HP Diagnostics

适用于 Windows®、 Unix 和 Linux 操作系统

软件版本：9.20

安装和配置指南

文档发行日期：2012 年 5 月
软件发布日期：2012 年 5 月

法律声明

担保

HP 产品和服务的唯一担保已在此类产品和服务随附的明示担保声明中提出。此处的任何内容均不构成额外担保。 HP 不会为此处出现的技术或编辑错误或遗漏承担任何责任。

此处所含信息如有更改，恕不另行通知。

受限权利声明

机密计算机软件。必须拥有 HP 授予的有效许可证，方可拥有、使用或复制本软件。按照 FAR 12.211 和 12.212，并根据供应商的标准商业许可的规定，商业计算机软件、计算机软件文档与商品技术数据授权给美国政府使用。

版权声明

商标声明

Java 是 Oracle 和 / 或其附属公司的注册商标。

Adobe® 和 Acrobat® 是 Adobe Systems Incorporated 的商标。

Microsoft®、 Windows®、 Windows® NT、 Windows® XP 和 Windows Vista® 是 Microsoft Corporation 在美国注册的商标。

Oracle® 是 Oracle 和 / 或其附属公司的注册商标。

UNIX® 是 The Open Group 的注册商标。

致谢

产品包括 Apache Software Foundation 开发的软件。 (http://www.apache.org/)。

产品包括 Spice Group (http://spice.codehaus.org) 开发的软件。

有关开放源和第三方许可证协议的信息，请参见产品安装介质上的 “文档”目录。

文档更新

此文档的标题页包含以下标识信息：

- 软件版本号，用于指示软件版本。
- 文档发行日期，该日期将在每次更新文档时更改。
- 软件发布日期，用于指示该版本软件的发布日期。

要检查是否有最新的更新，或者验证是否正在使用最新版本的文档，请访问：

http://support.openview.hp.com/selfsolve/manuals

需要注册 HP Passport 才能登录此站点。要注册 HP Passport ID，请访问：

http://h20229.www2.hp.com/passport-registration.html

或单击“HP Passport”登录页面上的“**New users - please register**”链接。

此外，如果订阅了相应的产品支持服务，则还会收到更新的版本或新版本。有关详细信息，请与您的 HP 销售代表联系。

支持

请访问 HP 软件支持网站：

http://support.openview.hp.com

此网站提供了联系信息，以及有关 HP 软件提供的产品、服务和支持的详细信息。

HP 软件联机支持提供客户自助解决功能。通过该联机支持，可快速高效地访问用于管理业务的各种交互式技术支持工具。作为尊贵的支持客户，您可以通过该支持网站获得下列支持：

- 搜索感兴趣的知识文档
- 提交并跟踪支持案例和改进请求
- 下载软件修补程序
- 管理支持合同
- 查找 HP 支持联系人
- 查看有关可用服务的信息
- 参与其他软件客户的讨论
- 研究和注册软件培训

大多数提供支持的区域都要求您注册为 HP Passport 用户再登录，很多区域还要求用户提供支持合同。要注册 HP Passport ID，请访问：

http://h20229.www2.hp.com/passport-registration.html

要查找有关访问级别的详细信息，请访问：

http://support.openview.hp.com/access_level.jsp

目录

第 III 部分：JAVA 代理和 .NET 代理的安装和设置

第 VI 部分：配置 DIAGNOSTICS 度量采集器

第 VII 部分：设置与其他 HP 软件产品的集成

第 VIII 部分：DIAGNOSTICS 服务器、JAVA 代理和 .NET 代理的高级配置

欢迎使用本指南

欢迎使用 《HP Diagnostics 安装和配置指南》 。本指南描述如何安装和配置 HP Diagnostics 组件，还概述了如何与其他 HP 软件产品集成。

本指南的结构

本指南包括以下部分：

第 I 部分　准备安装

提供有关计划和准备 Diagnostics 组件安装和配置的信息和说明。

第 II 部分　安装 Diagnostics 服务器和采集器

描述如何安装和配置 HP Diagnostics 服务器和 HP Diagnostics Collector。

第 III 部分　Java 代理和 .NET 代理的安装和设置

描述了如何安装和配置 Diagnostics Agent。

第 IV 部分　自定义插桩以监控 Java 和 .NET 应用程序

描述了如何控制 HP Diagnostics 应用于受监控应用程序的类和方法的插桩，以启用插桩从而收集性能度量。

第 V 部分　Diagnostics 服务器、 Java 代理和 .NET 代理的高级配置

描述了 Diagnostics 服务器、 Diagnostics .NET 和 Java 代理的高级配置。

第 VI 部分　配置通过代理服务器和防火墙进行的通信

描述了如何使用不同的通信通道对 Diagnostics 部署进行设置。

第 VII 部分　配置 Diagnostics 度量采集器

描述了度量捕获以及如何为 .NET 代理和 Java 代理配置度量采集器。

第 VIII 部分　设置与其他 HP 软件产品的集成

概述了如何设置 LoadRunner、 Performance Center 和 Business Service Management 以便与 HP Diagnostics 集成。

第 IX 部分　附录

描述了 Diagnostics 管理员要执行的诸如以下任务的管理任务，并提供了技术数据流图：

- 使用管理程序 UI 配置和管理 Diagnostics
- 设置用户、权限、授权和身份验证
- 启用组件之间的 HTTPS 安全通信
- 使用系统运行状况程序 UI
- 管理数据以及备份和恢复数据
- 升级 Diagnostics 和安装修补程序更新
- 使用数据导出功能
- 疑难解答和查找其他参考信息

HP Diagnostics 联机文档

HP Diagnostics 应用程序附带了以下文档：

- **《Diagnostics User's Guide and Online Help》。**介绍了如何使用 HP Diagnostics 来分析企业应用程序的性能。通过 Diagnostics UI 中的“帮助”按钮，或通过集成 HP 软件产品中的帮助菜单，可以访问“使用 HP Diagnostics”联机帮助。还可通过 Diagnostics 联机帮助主页、Windows 开始菜单（“开始”>“程序”>“HP Diagnostics Server”>“User Guide”）、HP Diagnostics 安装磁盘上的 Documentation 目录或 Diagnostics 服务器安装目录，来访问本指南的 PDF 版本。
- **《Diagnostics 安装和配置指南》。**介绍了如何安装和配置 Diagnostics 组件，以及如何配置 Diagnostics 以与其他 HP 软件产品集成。可通过 Diagnostics 联机帮助主页、Diagnostics 安装磁盘上的 Documentation 目录、Diagnostics 服务器安装目录或 Windows 开始菜单（“开始”>“程序”>“HP Diagnostics Server”>“Install Guide”）访问本指南。
- **Diagnostics 常见问题解答。**提供常见问题解答。可从 Diagnostics 联机帮助访问此 PDF。
- **Diagnostics 数据模型和查询 API。**描述 Diagnostics 数据模型以及用于访问数据的查询 API。可从 Diagnostics 联机帮助访问此 PDF。
- **自述文件。**提供了有关 HP Diagnostics 的最新技术信息和疑难解答信息。该文件位于 HP Diagnostics 安装磁盘根目录中。在此根目录中，还有一个升级和修补程序安装说明文档，其中包含有关安装升级或修补程序的详细信息。
- **《Diagnostics Java Agent Guide》。**描述了如何安装、配置以及使用 Diagnostics Java 代理和 Diagnostics Profiler for Java。可以通过代理系统的 \docs 目录、Java Diagnostics Profiler UI 联机帮助链接或 HP Diagnostics 安装磁盘的 Documentation 目录访问本指南。

➤ **《Diagnostics .NET Agent Guide》。**描述如何安装、配置和使用 Diagnostics .NET 代理及 Diagnostics Profiler for .NET。可以通过代理系统的 \docs 目录、.NET Diagnostics Profiler UI 联机帮助链接或 HP Diagnostics 安装磁盘的 Documentation 目录访问本指南。

注意：Diagnostics Agent 指南以 《Diagnostics 安装和配置指南》和《Diagnostics User's Guide》为基础。

其他联机资源

HP 软件网站。可访问 HP 软件网站。此站点提供了 HP 软件产品的最新信息，具体包括新软件版本、研讨会和展销会、客户支持等。要访问此网站，请选择 “帮助” > “HP 软件网站”。此网站的 URL 是 www.hp.com/go/software。

HP Software 支持。可访问 HP 软件支持网站。通过此网站，您不但可以浏览“自助解决”知识库，还可以搜索用户论坛并将信息发布到论坛、提交支持请求、下载修补程序和最新文档等。要访问此网站，请选择 “帮助” > “HP 软件支持”。此网站的 URL 是 http://support.openview.hp.com。

大多数提供支持的区域都要求您注册为 HP Passport 用户再登录，很多区域还要求用户提供支持合同。

要查找有关访问级别的详细信息，请访问：

http://support.openview.hp.com/access_level.jsp

要注册 HP Passport 用户 ID，请访问：

http://h20229.www2.hp.com/passport-registration.html

文档更新

HP 软件将不断更新其产品文档。

要检查是否有最新更新，或验证所使用的文档是否为最新版本，请访问 HP 软件产品手册网站 (http://support.openview.hp.com/selfsolve/manuals)。

第 I 部分

准备安装

本部分包括：

- 准备安装 HP Diagnostics

1

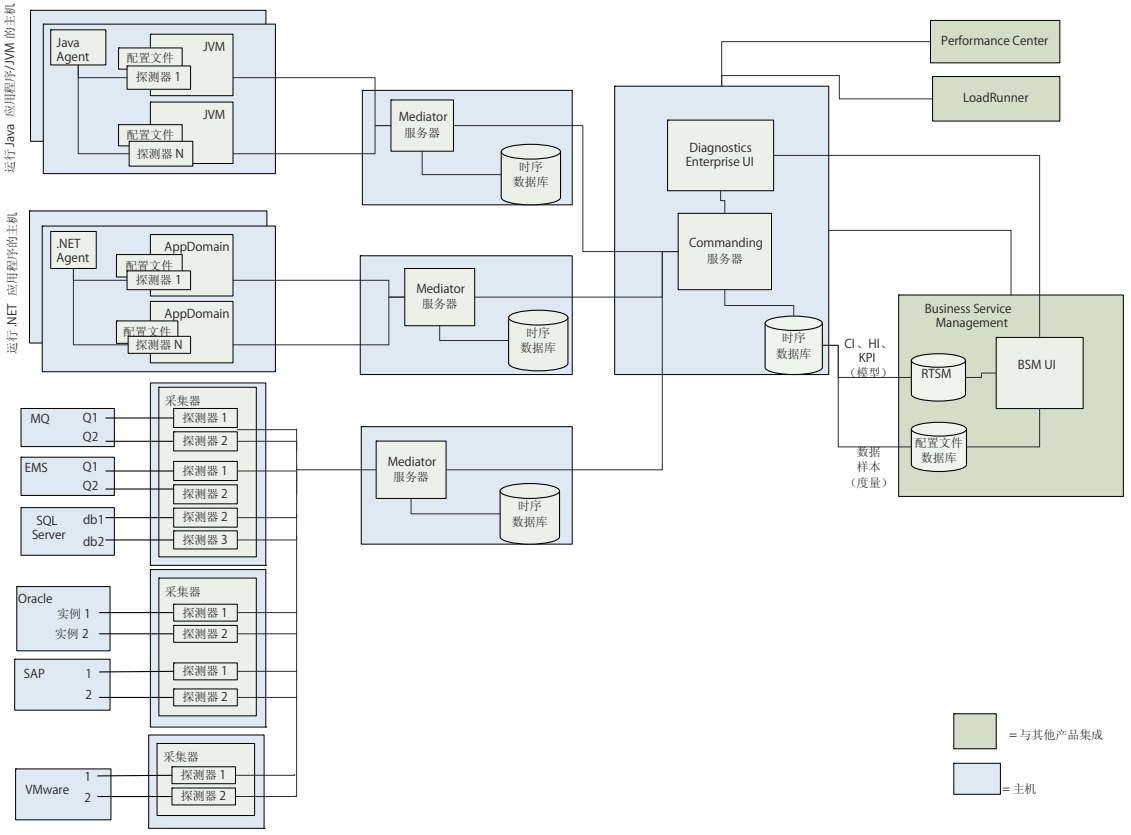
准备安装 HP Diagnostics

安装 HP Diagnostics 之前，请阅读以下信息和说明，以帮助您计划和准备对 Diagnostics 组件执行安装和配置。

本章包括：

HP Diagnostics 组件和数据流

下图显示了 Diagnostics 组件之间的数据流以及与其他 HP 软件产品的集成情况。



HP Diagnostics 包含以下组件：

➤ **Diagnostics Agent** 从 J2EE 和 .NET 应用程序捕获事件 （如方法调用、业务事务的开始和结束以及服务器请求），然后聚合性能度量以发送到 Diagnostics 服务器。

Diagnostics Agent 软件安装在要监控的目标系统上。借助 Java 代理，您可以插桩应用程序服务器以执行监控。借助 .NET 代理，您可以插桩应用程序域以执行监控。

插桩的每个应用程序服务器或应用程序域将生成一个由探测器实体表示的代理实例。通过在代理安装文件夹中使用多个不同的配置文件，可以控制这些探测器实体的数据收集设置。

- **Diagnostics Collector。**负责从外部环境收集数据，包括 Oracle 数据库、SQL 服务器系统、 IBM WebSphere MQ 消息系统、 TIBCO Enterprise Message Service、 Software AG webMethods 代理、 VMware vCenter 或 VMware ESX Server 以及 SAP NetWeaver - ABAP 系统。您可以安装 Diagnostics Collector，并定义要监控的这些系统的特定实例。每个监控的实例均在 Diagnostics 用户界面中表示为一个探测器实体。

- **Diagnostics 服务器。**负责使用代理、采集器以及其他用于捕获、处理和显示应用程序性能度量的 HP 软件产品。

  Diagnostics 服务器处理并进一步聚合接收的数据，然后格式化信息，以便在用户界面视图中显示这些信息。

  Diagnostics 部署中可包含一个或多个 Diagnostics 服务器。如果部署中仅有一个 Diagnostics 服务器，则将其配置为 Diagnostics Commander 服务器，并且必须执行 Commander 和 Mediator 角色。如果部署中有多个 Diagnostics 服务器，则必须将其中一个配置为 Diagnostics Commander 服务器，其余所有作为 （分布式） Mediator 运行。

  在典型部署中，Diagnostics Commander 服务器将连接到作为 Mediator 角色运行的一个或多个服务器。所有 Diagnostics Mediator Server 均配置为从安装有代理和采集器的系统接收数据。然后， Diagnostics Mediator Server 将筛选和聚合其接收到的事件。这些信息将发送到 Diagnostics Commander 服务器，后者在 UI 中显示经过处理的度量。

  Diagnostics Commander 服务器负责各个 Diagnostics 组件和与 Diagnostics 一起使用的其他产品组件之间的命令和控制功能。

  此 Commander 服务器跟踪其他 Diagnostics 组件的位置和状态，并作为其他组件之间的通信集线器。

此 Commander 服务器还负责在 Diagnostics 用户界面中显示受监控应用程序的性能信息。

用户界面。主 Diagnostics 用户界面 (Diagnostics Enterprise UI) 在图表和图形中显示性能数据，用于监控性能、隔离问题以及分析原因，以解决复杂的性能问题。

如果与其他 HP 软件产品一起使用 Diagnostics，则还可以从其他产品的用户界面访问 Diagnostics Enterprise UI。例如，可从 HP Business Service Management 访问 Diagnostics Enterprise UI。在预生产的负载测试期间，可从 HP LoadRunner 或 HP Performance Center 访问 Diagnostics Enterprise UI。

此外， Diagnostics 还提供 Java 和 .NET Profiler，它们显示在单独的用户界面 (Diagnostics Profiler UI) 中，您可直接在代理系统或从主 Diagnostics 用户界面向下搜索访问这些界面。

- **集成。** Diagnostics 与以下其他 HP 软件产品集成。有关详细信息，请参阅第 VII 部分，“设置与其他 HP 软件产品的集成”。另请参阅在线帮助或用户指南中“与其他 HP 软件产品集成”一节。
  - HP Business Service Management
  - HP LoadRunner
  - HP Performance Center
  - HP Sitescope
  - HP Continuous Delivery Automation (CDA)

受支持的应用程序服务器和环境

HP Diagnostics 支持对以下各项进行监控：

- **基于 Java EE 的应用程序服务器。**包括 WebLogic、WebSphere、Oracle、Sun Java 企业服务器、 JBoss 等。
- **基于 .NET 的应用程序服务器。** HP Diagnostics 支持 Microsoft IIS .NET Framework。

- **SAP NetWeaver–ABAP 系统**。
- **Oracle 数据库**。
- **SQL Server 数据库**。
- **IBM WebSphere MQ 系统**。
- **TIBCO Enterprise Message Service (EMS) 系统。**
- **VMware vCenter 或 VMware ESX Server**。
- **Software AG webMethods 代理**

有关受支持的环境的最新信息，请参阅 Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

Diagnostics 组件的系统要求

下面介绍建议的用于安装 HP Diagnostics 组件的系统配置。请参阅上一节中的部署图，以了解本节介绍的组件主机。

选择用于安装 Diagnostics 组件的计算机时，请确保计算机的系统配置支持负载处理功能和要监控的应用程序数。

本节包括以下主题：

- “Diagnostics .NET Profiler 用户界面主机的要求”（第 37 页）
- “Diagnostics Collector 主机的要求”（第 37 页）

支持 Diagnostics 组件的环境

有关与 Diagnostics 组件相关的受支持环境的最新信息，请参阅 Diagnostics Support Matrix，网址为：
http://support.openview.hp.com/sc/support_matrices.jsp。

Diagnostics 服务器和 Diagnostics Collector 使用 Java 1.6 JVM。

重要信息：对于 Diagnostics Linux 安装程序（用于服务器、代理和采集器的 32 位和 64 位安装程序），必须在 64 位 Linux 系统上安装修补程序 libXtst-1.0.1-3.1，才能在图形模式中运行安装程序。

Diagnostics Enterprise UI 的要求

将通过使用一个 Java 小程序在 Web 浏览器中显示 Diagnostics Enterprise UI，此 Java 小程序要求至少在用于访问 UI 的客户端系统上安装 JRE 1.6。受支持的浏览器包括 Microsoft Internet Explorer 7、8、9 以及 Mozilla Firefox 3.5、3.6、5 和 6。有关受支持的浏览器的最新信息，请参阅 Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

Diagnostics 服务器主机的要求

Diagnostics 服务器主机的系统配置要求取决于探测器的数量和向主机报告数据的 Mediator 服务器的数量。将一个服务器指定为 Diagnostics Commander 服务器之后，探测器数据通常存储在向 Diagnostics commander 服务器报告数据的每个 Mediator 服务器上。

如果在 SAN 存储设备上安装 Diagnostics 服务器，则 SAN 必须具有可与中高端驱动器相媲美的足够的读取和写入速度 （请参阅 “扩展性信息”（第 32 页））。

注意：下面表中的系统要求基于测试的指导原则，在这些测试中，探测器监控了具有平均服务器请求数和服务器请求深度的应用程序。所需的实际系统要求以及受支持的实际探测器数量受所监控环境的多个特征影响，包括服务器请求数、服务器请求深度（调用配置文件中的方法），已趋势化的方法数和出站调用次数。服务器请求的类型也会影响系统要求。例如， Web 服务需要更多的资源，并且不能对它们应用剪裁。

下表列出了从 Java 探测器接收数据的 Diagnostics 服务器 （通常为 Mediator 服务器）的主机系统的要求。

平台	项目	最多 50 个 Java 探测器	最多 100 个 Java 探测器	最多 200 个 Java 探测器
Windows	CPU	2x 2.4 GHz	2x 2.8 GHz	2x 3.4 GHz
Windows	内存	4 GB	4 GB	4 GB
Solaris	CPU	2x Ultra Sparc 3	2x Ultra Sparc 4	2x Ultra Sparc 4
Solaris	RAM	4 GB	4 GB	4 GB
Linux	CPU	2x 2.0 GHz	2x 2.4 GHz	2x 2.8 GHz
Linux	内存	2 GB	4 GB	4 GB
所有平台	堆大小	512 M	750 M	1280 M
所有平台	磁盘	每个探测器 4 GB		

关于测试环境的注意事项

- 每个服务器请求的调用配置文件 （方法调用的深度）：5
- 每个探测器的唯一服务器请求数：23

下表列出了从 .NET 探测器接收数据的 Diagnostics 服务器（通常为 Mediator 服务器）的主机系统的要求。

平台	项目	最多 10 个 .NET 探测器	最多 20 个 .NET 探测器	最多 50 个 .NET 探测器
Windows	CPU	1x 1.0 GHz	1x 2.0 GHz	2x 2.4 GHz
Windows	内存	768 MB	1 GB	3 GB
Solaris	CPU	1x Ultra Sparc 2	2x Ultra Sparc 2	2x Ultra Sparc 3
Solaris	RAM	1 GB	1.5 GB	3 GB
Linux	CPU	1x 1.0 GHz	1x 2.0 GHz	2x 2.4 GHz
Linux	内存	768 MB	1 GB	3 GB
所有平台	堆大小	350 M	700 M	1400 M
所有平台	磁盘	每个探测器 3 GB		

扩展性信息

以下扩展性数字派生自下面的参考硬件配置：

平台： Windows

操作系统： Windows Server 2008 （64 位）

CPU： Intel Xeon 5160， 3.00Ghz （四核处理器）

内存： 8 GB

磁盘 I/O： RAID 0 Smart Array P400i， 2SCSI 驱动器 (136 GB) [130 MB/S，顺序读取和写入]

Java 堆： 5.9 GB (-Xmx6096m)； 64 位 JVM

磁盘空间： 2-4 GB/ 探测器 （可以通过更改保存时间间隔调整总磁盘空间）

网络： 1 Gbps

注意：要获得最佳性能，建议将 64 位操作系统和 JVM 与 Diagnostics 一起使用。

之前参考的硬件的扩展性数字。

最多 100 个 Java 探测器： 每个探测器 100 个服务器请求，每 45 秒提取的每个调用配置文件对应 78 个方法 （默认）

最多 400 个 Java 探测器： 每个探测器 25 个服务器请求，每 45 秒提取的每个调用配置文件对应 78 个方法 （默认）

最多 150 个 Java 探测器： 每个探测器 150 个服务器请求，每 240 秒提取的每个调用配置文件对应 25 个方法

最多 230 个 Java 探测器： 每个探测器 100 个服务器请求，每 240 秒提取的每个调用配置文件对应 25 个方法

最多 40 个 Java 探测器： 75 个 Web 服务操作，每个 Web 服务操作对应 10 个唯一的用户，每 45 秒提取的每个调用配置文件对应 25 个方法 （默认）

请注意，此负载配置要求为每个探测器提供 7 GB 的磁盘空间。

另请参阅 “针对大型安装配置 Diagnostics 服务器”（第 428 页）。

注意：

- 在包含多个探测器的环境中，如果安装两个或更多 Diagnostics 服务器实例，并在每个服务器实例之间分布探测器，则可以获得更好的性能。
- 有关与 Diagnostics Commander 服务器主机上存储的 Diagnostics 性能数据相关的配置注意事项，请参阅 “预安装数据管理注意事项”（第 840 页）。
- 有关如何优化 Diagnostics 服务器以处理更多探测器的信息，请参阅 “在生产过程中优化 Diagnostics 服务器以处理 更多探测器”（第 453 页）。

Diagnostics Java 代理主机的要求

Diagnostics Java 代理在受监控系统上占用的开销非常低。下面是支持代理处理功能的建议的内存和磁盘空间：

平台：	所有平台
内存：	50 MB 的额外 RAM
可用硬盘空间：	初次安装 Java 探测器时，需要 200 MB 的可用磁盘空间。由于创建日志文件和类映射，所以在运行期间可能需要更多空间。对于大型应用程序，建议为每个探测器提供额外 200 MB 的可用空间，用于存储日志文件和类映射数据。

注意：额外内存必须分配给 JVM 的最大堆，方法是将 **-Xmx???m** 添加到应用程序启动脚本中的 Java 设置中。

调整堆大小。有关设置 Java 代理最大堆的信息，请参阅“在应用程序服务器中调整 Java 代理的堆大小”（第 235 页）。

调整 permgen 大小。通常，因添加 Diagnostics Agent 而产生的 permgen 大小增加均很小。然而，在某些情况下，没有代理的应用程序几乎会使用所有 permgen。因此，将需要调整 permgen 大小。例如，可通过向现有限制 * 1.05 添加 5 MB 来增加大小。要为热点 JVM 调整 permgen，请使用 -XX:MaxPermSize 选项，例如：-XX:MaxPermSize=240m。

Diagnostics Java Profiler 用户界面主机的要求

将通过使用 Java 小程序在 Web 浏览器中显示 Diagnostics Profiler 用户界面，该 Java 小程序要求至少在用于访问 UI 的客户端系统上安装 JRE 1.6 或更高版本。此计算机必须能够访问 Diagnostics Profiler URL：http://< 探测器主机 >:< 探测器端口 >/profiler。默认情况下，探测器会被分配到从端口 35000 开始的第一个可用端口。受支持的浏览器包括 Microsoft Internet Explorer 7、 8、 9 以及 Mozilla Firefox 3.5、3.6、5 和 6。有关受支持的浏览器的最新信息，请参阅 Diagnostics Support Matrix，网址为：
http://support.openview.hp.com/sc/support_matrices.jsp。

Diagnostics .NET 代理主机的要求

.NET 代理在受监控系统上占用的开销非常低。下面是支持代理处理功能的建议的内存和磁盘空间：

平台	所有支持的平台
内存	60 MB 额外 RAM
可用硬盘空间	200 MB 额外空间
.NET Framework	2.0 或更高版本

重要信息：如果必须支持 .NET Framework 1.1，则使用较早版本的 .NET 代理 (8.x)，将持续对此版本提供支持并通过修补程序对其进行更新。

WCF 要求和限制：如果要监控 .NET Windows Communication Foundation (WCF) 服务，则需要安装 .NET Framework 3.0 SP1 或更高版本。仅支持以下绑定：

- BasicHttpBinding
- WSHttpBinding
- NetTcpBinding

如果应用程序使用不受支持的绑定，则 .NET 探测器仅会为每个 WCF 方法创建常规服务器请求。这不是一种 Web 服务，也没有任何 XVM 关联。

Diagnostics .NET Profiler 用户界面主机的要求

.NET Diagnostics Profiler 用户界面通过 DHTML/XML/XSLT/JScript 技术显示，该技术要求安装 Internet Explorer 7 或更高版本。用于显示 UI 的计算机必须能够访问 .NET Diagnostics Profiler URL：http://< 探测器主机 >:< 探测器端口 >/profiler。探测器会分配到安装探测器时定义的范围内的第一个可用端口。默认端口范围是 35000 - 35100。

Diagnostics Collector 主机的要求

采集器可以安装在受支持的系统上，这些系统可以与从中收集数据的 SAP NetWeaver–ABAP、Oracle、SQL 服务器、IBM WebSphere MQ、TIBCO EMS、Software AG webMethods 代理或 VMware 应用程序的主机交互。

安装采集器，需要 350MB 的可用磁盘空间。由于创建日志文件或大型环境，在运行期间可能需要更多空间。

有关与 Diagnostics 组件相关的受支持环境的最新信息，请参阅 Diagnostics Support Matrix，网址为：
http://support.openview.hp.com/sc/support_matrices.jsp。

安装过程所需的信息

安装 Diagnostics 组件之前，需要仔细计划 Diagnostics 组件以及安装这些组件的计算机的配置。另外，还要考虑组件主机在网络结构中的位置。

以下各节中的表格有助于您在 Diagnostics 组件安装期间收集所需的信息。

注意：如果要将 Diagnostics 与 Business Service Management 一起安装，则在输入 Diagnostics 组件的主机名时，强烈建议使用完全限定主机名（即计算机名称加域名）。

Diagnostics 服务器

所需信息	描述
是否具有一个 Commander 服务器和一个或多个 Mediator？	在计划 Diagnostics 部署期间，这将根据受监控环境的大小和复杂性来决定。
对于 Diagnostics Commander 服务器，为安装此服务器的计算机生成的 HP Diagnostics 许可证的位置	联系 HP 软件支持人员，并申请许可证，然后将许可证放在可以从 Diagnostics 服务器安装程序访问的文件夹中。
对于 Diagnostics Mediator 服务器，Diagnostics Commander 服务器的 URL	安装 Diagnostics Commander 服务器之后，可用于打开 Diagnostics 视图的 URL 可用。
是否将在 SaaS 环境中使用 Diagnostics？	如果在 SaaS（由 HP 托管）环境中部署 Diagnostics，则将显示其他安装程序选项。
是否将在 Business Service Management 环境中集成 Diagnostics？	如果将 Diagnostics 部署到 Business Service Management 环境中，则将需要在安装程序中选择此选项。

Java 代理

➤ **HP 软件产品和 Diagnostics 服务器信息**

所需信息	查找位置	值
安装代理的模式	根据产品许可证进行选择。	➤ **仅 Profiler（不连接到服务器）** ➤ **仅与 LoadRunner/ Performance Center 一起使用 （AD 许可证）** ➤ **将企业模式 （AM 许可证）与以下一个或两个应用程序一起使用：** ➤ Diagnostics ➤ TransactionVision
Diagnostics 服务器名称	Diagnostics 服务器主机的完全限定主机名或 IP 地址。 在独立模式中使用 Java Diagnostics Profiler 时则不需要。	如果运行代理的部署中仅有一个 Diagnostics 服务器，是 Diagnostics Commander 服务器。 在包含一个 Commander 服务器和多个 Mediator 服务器的分布式环境中，则是要从代理接收事件的 Diagnostics Mediator 服务器。
Diagnostics 服务器端口	使用默认端口 2006 或为访问 Diagnostics 所配置的端口。 在独立模式中使用 Java Diagnostics Profiler 时则不需要。	**默认值**：2006

➤ **插桩的应用程序服务器和代理信息**

所需信息	查找位置	值
Java 代理名称	唯一的字符串； 由用户创建。 代理名称将指定为默认探测器实体名称。 代理名称应指明要监控的应用程序和探测器设备的类型，以帮助区分其他应用程序和探测器类型。 多个探测器可以使用一个 Java 代理配置。此时，您可以稍后为每个受监控的应用程序配置唯一的探测器名称。	例如： WebLogic_MedRec_java
Java 代理组	在安装代理时由用户定义。 所输入的代理组名称将用作探测器组名称。 探测器组是向相同的 Diagnostics 服务器进行报告的探测器的逻辑分组。	**默认值：** Default
将插桩以执行监控的应用程序服务器类型	主机系统管理员。	

所需信息	查找位置	值
应用程序服务器配置属性	主机系统管理员。 根据要监控的应用程序服务器，详细信息会有所不同。	
JRE 可执行文件的位置	主机系统管理员。 取决于要监控的应用程序服务器类型。请参阅 “准备应用程序服务器以使用 Java 代理进行监控”（第 161 页）。	

.NET 代理

➤ **Diagnostics 服务器信息**

所需信息	查找位置	值
安装代理的模式	根据产品许可证进行选择。	➤ **仅 Profiler（不连接到服务器）** ➤ **仅与 LoadRunner/ Performance Center 一起使用 （AD 许可证）** ➤ **将企业模式 （AM 许可证）与以下一个或两个应用程序一起使用：** ➤ Diagnostics ➤ TransactionVision
Diagnostics 服务器名称	Diagnostics 服务器主机的完全限定主机名或 IP 地址。 在独立模式中使用 .NET Diagnostics Profiler 时则不需要。	如果运行代理的部署中仅有一个 Diagnostics 服务器，是 Diagnostics Commander 服务器。 在包含一个 Commander 服务器和多个 Mediator 服务器的分布式环境中，则是要从代理接收事件的 Diagnostics Mediator 服务器。
Diagnostics 服务器端口	使用默认端口 2006 或为访问 Diagnostics 所配置的端口。 在独立模式中使用 .NET Diagnostics Profiler 时则不需要。	**默认值：**2612

➤ **代理和端口信息**

所需信息	查找位置	值
代理组	在安装代理时由用户定义。 所输入的代理组名称将用作探测器组名称 探测器组是向相同的 Diagnostics 服务器进行报告的探测器的逻辑分组。	**默认值：** 默认值
最小 Web 端口	系统管理员。 代理系统上可分配到探测器的端口范围内的最小端口号。	**默认值：**35000
最大 Web 端口	系统管理员。 代理系统上可分配到探测器的端口范围内的最大端口号。	**默认值：**35100

预安装注意事项

注意：在 Windows 计算机上安装任何 Diagnostics 组件之前，请确保未打开“服务”窗口 （可从 “管理工具”访问）。

Diagnostics 服务器

- 只有在使用有效许可证为 Diagnostics Commander 服务器 授予许可后，才能显示 HP Diagnostics 性能度量。有关获取许可证和其他许可问题的详细信息，请参阅第 3 章，“HP Diagnostics 许可”。

注意：要获得最佳的 Diagnostics 视图显示效果，屏幕分辨率至少应为 1024 x 768。

Diagnostics Java 代理

- Java 代理必须与处于测试阶段的 Java 应用程序安装在相同的系统上。
- 在 Diagnostics Profiler for Java 能够连接到已正确获得许可的 Diagnostics Commander 服务器之前， Java Diagnostics Profiler 会在具有负载限制的未许可模式下运行。有关获取许可证和其他许可问题的详细信息，请参阅第 3 章，“HP Diagnostics 许可”。
- Diagnostics 不支持代理名称的本地化。

Diagnostics .NET 代理

- .NET 代理必须与处于测试阶段的 .NET 应用程序安装在相同的系统上。
- 在 Diagnostics Profiler for .NET 能够连接到已正确获得许可的 Diagnostics Commander 服务器之前，Java Diagnostics Profiler 会在具有负载限制的未许可模式下运行。有关获取许可证和其他许可问题的详细信息，请参阅第 3 章，“HP Diagnostics 许可”。
- Diagnostics 不支持代理名称的本地化。

LoadRunner 和 Performance Center 主机

- 如果已安装 LoadRunner，则在安装 LoadRunner Diagnostics 插件之前，确保关闭控制器和 LoadRunner 主窗口。
- Performance Center 不需要 LoadRunner Diagnostics 插件。
- Diagnostics 组件的主机计算机的时间和时区设置必须一致。如果未正确设置时间，则会遇到时间差问题。

建议的安装顺序

仔细计划和准备 HP Diagnostics 组件安装有助于避免复杂化和错误，并能快速完成安装和配置步骤。

注意：建议按以下顺序安装产品和组件。如果不遵循此顺序，可能会增加安装过程的复杂程度，并产生不可预知的结果。

在开始之前，请查看以下信息以了解整个安装和配置过程。

建议的安装顺序：

1 查看系统要求和安装注意事项。

请参阅 “Diagnostics 组件的系统要求”（第 29 页）。

2 安装 Diagnostics 服务器。

有关详细信息，请参阅第 2 章，“安装 Diagnostics 服务器”和第 3 章，“HP Diagnostics 许可”。

3 安装 Diagnostics Agent 和 Collector。

有关 Java 环境，请参阅第 5 章，“安装 Java 代理”。

有关 .NET 环境，请参阅第 8 章，“安装 .NET 代理”。

有关 Oracle 数据库、 SAP NetWeaver-ABAP、 SQL 服务器数据库、 VMware vCenter 或 VMware ESX Server、WebSphere MQ、TIBCO EMS 以及 Software AG webMethods 代理环境，请参阅第 4 章，“安装 Diagnostics Collector”。

4 对于 Java 代理，插桩将由 Diagnostics 监控的应用程序服务器；这会导致生成在 Diagnostics 中表示为探测器实体的代理实例。

有关详细信息，请参阅第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”。

5 通过在代理安装文件夹中使用多个不同的配置文件，可以自定义插桩并控制数据收集设置。

有关详细信息，请参阅“自定义插桩以监控 Java 和 .NET 应用程序”和“Diagnostics 服务器、 Java 代理和 .NET 代理的高级配置”中的各节

6 如果将 HP Diagnostics 与 LoadRunner、Performance Center 或 Business Service Management 集成，则需要配置其中的每个产品，以使用 HP Diagnostics。

有关 Business Service Management，请参阅第 21 章，“设置 Business Service Management 和 Diagnostics 之间的集成”。

对于 LoadRunner 集成，请安装 LoadRunner Diagnostics 插件（请参阅第 22 章，“安装 LoadRunner Diagnostics 插件”）并设置 LoadRunner，以使用 Diagnostics（请参阅第 23 章，“设置 HP LoadRunner 与 HP Diagnostics 集成”）。

有关 Performance Center，请参阅第 24 章，“设置 Performance Center 以使 Diagnostics”。

许可 HP Diagnostics

要在 Diagnostics 视图中查看应用程序的度量，必须获取 Diagnostics Commander 服务器的有效许可证。有关获取许可证和其他许可问题的详细信息，请参阅第 3 章，“HP Diagnostics 许可”。

从 Diagnostics 较早版本升级

如果已安装产品的上一个版本，或需要升级其他 HP 软件产品才能访问 Diagnostics 功能，则在安装 HP Diagnostics 时，请按照附录 G，“升级和修补程序安装说明”。中的说明执行操作。这些说明可指导您升级当前的 HP 软件产品和 Diagnostics 组件。

第 II 部分

安装 Diagnostics 服务器和采集器

本部分包括：

- 安装 Diagnostics 服务器
- HP Diagnostics 许可
- 安装 Diagnostics Collector

2

安装 Diagnostics 服务器

本节介绍如何在 Windows 和 UNIX 计算机上安装 Diagnostics 服务器。

本章包括：

安装 Diagnostics 服务器

本章提供有关安装 Diagnostics 服务器的详细说明，将适用于以下环境：

- Windows 环境
- 使用图形安装程序的大多数 UNIX 环境。还可使用控制台模式命令行界面在 UNIX 上安装。

有关受支持的平台的最新信息，请参阅“Diagnostics Support Matrix”，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。请与 HP 支持联系，获取有关支持列表中未列出的其他平台的安装帮助。

注意：如果计算机上安装了 Diagnostics 服务器的较早版本，请参阅附录 G，“升级和修补程序安装说明”。

根访问权限要求。如果 Diagnostics Commander 服务器将与 Business Service Management 9.00 或更高版本集成，则在 Diagnostics 服务器安装期间要求具有根访问权限。要安装 OM 代理和 IAPA 组件，需要根访问权限。

如果需要在没有根访问权限的情况下安装 Diagnostics 服务器，则可以选择不安装 OM 代理和 IAPA 组件，并在稍后手动安装它们。当看到对话框时：OM 代理和 IAPA 组件安装不选中此框，并在稍后安装 （请参阅 “手动安装 OM 代理和 IAPA 组件”（第 72 页））。

此外，在 Solaris 和 Linux 系统上设置服务器在系统启动时自动启动，也需要根权限。

HP 软件即服务 (SaaS)。可将 HP Diagnostics 部署到 HP 软件即服务 (SaaS) 环境中。在 SaaS 部署中， Diagnostics Java 代理安装在您公司的 IT 环境中，Diagnostics Commander 服务器和 Mediator 服务器由 HP 安装在 HP 自己的 SaaS 系统上。因此，对于在 SaaS 环境中使用 Diagnostics 的客户，通常不用在公司系统上安装任何 Diagnostics 服务器，因而可忽略本章内容。在 SaaS 部署中，客户只需安装 Java 代理，用以连接由 HP SaaS 管理员在 HP premises 上设置的服务器。有关详细信息，请联系 SaaS 管理员。

本节包括：

启动 Diagnostics 服务器安装程序

根据您的计算机环境，启动 Windows 安装程序或 UNIX 安装程序。另请参阅“Diagnostics 服务器静默安装”（第 66 页）。

注意：临时目录的可用空间约为 400 MB。

要访问 Windows 安装程序，请执行以下操作：

1 Diagnostics DVD (Autorun.exe) 中将显示安装菜单页面。从菜单中选择 **Diagnostics 服务器 32 位**，安装 32 位 Windows 版本的 Diagnostics 服务器。选择 **Diagnostics 服务器 64 位**，安装 64 位版本的 Diagnostics 服务器。

2 或者，通过双击 **Diagnostics_Servers** 目录中的可执行文件 **HPDiagServer_< 版本号 >_win32.exe**（32 位）或 **HPDiagServer_< 版本号 >_win64.exe**（64 位，使用 64 位 JVM 运行），可直接运行相应的安装程序。

按照 “运行安装”（第 56 页）中的说明继续操作。

要访问 UNIX 安装程序，请执行以下操作：

1 从 Diagnostics 安装位置访问 **Diagnostics_Servers** 目录。将相应的安装程序 **HPDiagServer_<版本号>_<平台>.bin** 复制到要安装 Diagnostics 服务器的计算机中。

2 更改安装程序文件的模式以使其可执行。

3 运行安装程序。

➤ 要在图形模式中运行安装程序，请在 UNIX 命令提示符处输入安装程序 **HPDiagServer_<版本号>_<平台>.bin** 文件名，例如：

```
./HPDiagServer_9.00_linux.bin
```

➤ 要在控制台模式中运行安装程序，请在 UNIX 命令提示符处输入安装程序 **HPDiagServer_<版本号>_<平台>.bin** 文件名与 **-console** 选项，例如：

```
./HPDiagServer_9.00_linux.bin -console
```

按照 “运行安装”（第 56 页）中的说明继续操作。

要从 HP 软件下载中心下载安装程序，请执行以下操作：

1 转至 HP 软件网站的软件下载中心。

2 查找 **Diagnostics** 信息，并选择下载 Diagnostics 服务器软件相应的链接。

3 按照网站上的下载说明进行操作。

按照 “运行安装”（第 56 页）中的说明继续操作。

要从 Business Service Management Diagnostics “下载”页面下载安装程序，请执行以下操作：

1 在 Business Service Management 中，从顶部菜单中选择“管理”>“Diagnostics”，然后单击 “下载”选项卡。

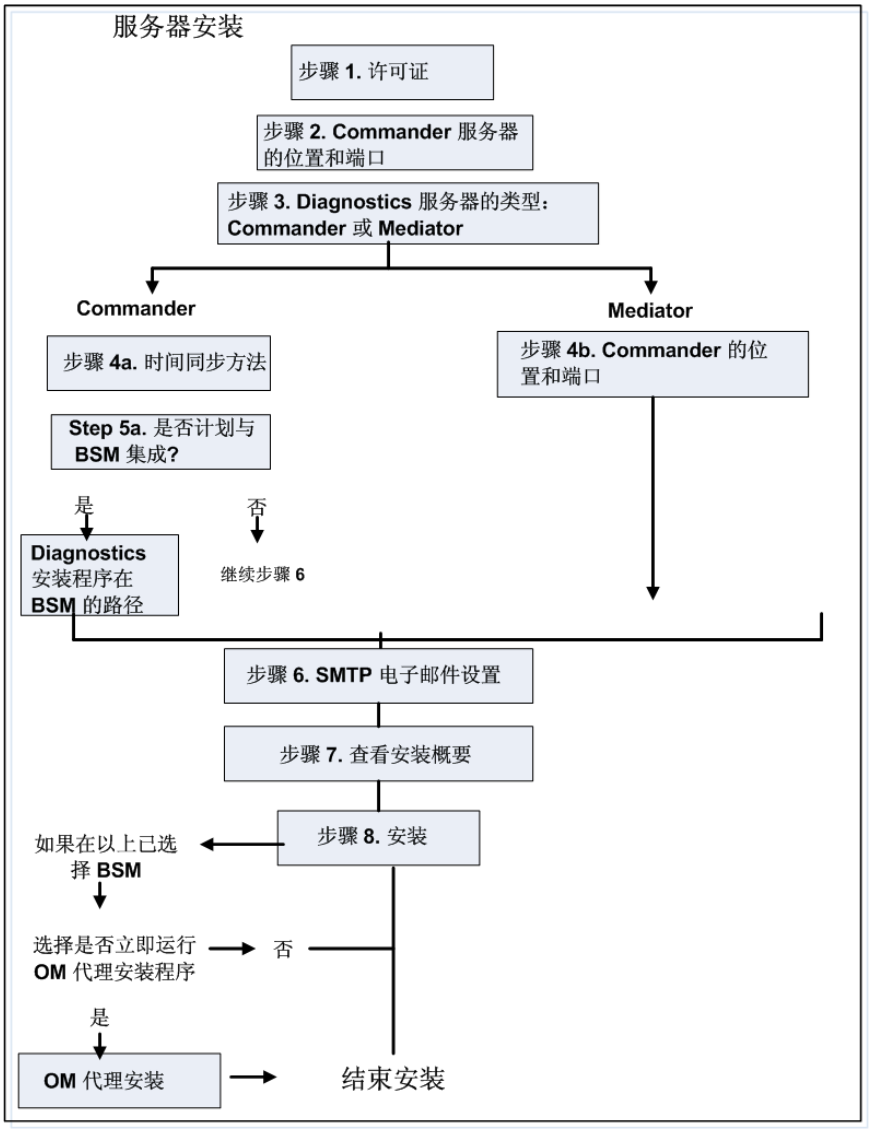
2 在 “下载”页面上单击链接，下载相应的 Diagnostics 服务器安装程序。

注意：只有将 Java 代理安装程序置于 Business Service Management 能够访问的目录下时，才能在 Business Service Management 中使用此程序。您可以在 Diagnostic 服务器安装期间启用此程序，也可以将服务器安装程序从安装光盘手动复制到所需位置。

按照 “运行安装”（第 56 页）的说明继续操作。

运行安装

下图概述了 Diagnostics 服务器的安装步骤；有关每个步骤的详细信息，请参阅本节其余内容。



请注意，HP SaaS 管理员在 HP 自有系统上进行 SaaS 服务器部署需要执行其他步骤 （HP 内部文档）。

重要信息：如果主机上存在预安装的 Java 代理，则必须按照说明升级此代理系统，而不遵循这些安装说明，请参阅 “升级和修补程序安装说明”（第 851 页）。

启动安装程序之后，将打开软件许可证协议。

在 Ubuntu 或 CentOS Linux 上，如果自解压缩存档（安装文件）失败，则需要安装 32 位 glibc 库。

要在 RHEL 6/CentOS 6 中安装 32 位 glibc 包，请运行以下命令：
yum install glibc.i686。

要在 Ubuntu 11 中安装 32 位 glibc 包，请运行以下命令：
sudo apt-get install libc6-i386。

然后再次尝试运行安装。

要开始安装并选择安装位置和模式，请执行以下操作：

1 接受软件许可证协议。

此时将显示软件许可证协议。

阅读协议并接受协议中的条款。

选择 “下一步”继续。

注意：对于 UNIX 控制台模式安装程序，可以在阅读时按 ENTER 键移动到下一页文字内容，或键入 q 跳到许可证协议的结尾。

---

2 指定 Diagnostics 服务器的安装位置。

可接受默认安装目录，也可键入其他位置的路径。在 Windows 安装程序（或 UNIX 图形模式安装程序）中，单击 “浏览”导航到其他目录。

选择 “下一步”继续。

注意：在 UNIX 控制台模式安装程序中，按 1 键选择 “下一步” 、按 2 键选择 “上一步”、按 3 键选择 “取消”或按 4 键选择 “重新显示屏幕”。

3 为要安装的 Diagnostics 服务器指定模式。

要设置的 Diagnostics 部署可以包含一个或多个 Diagnostics 服务器。如果在部署中仅有一个 Diagnostics 服务器，则该 Diagnostics 服务器会以 Commander 模式安装，并且可以执行 commander 和 mediator 角色。如果在部署中有多个 Diagnostics 服务器，则会将其中一个配置为 Commander 模式，将其余所有 Diagnostics 服务器配置为向 Commander 服务器报告的 Mediator 模式。

- 如果部署中仅有一个 Diagnostics 服务器，请选择 “Commander 模式”。
- 如果在部署中有多个 Diagnostics 服务器，并且要将当前安装的 Diagnostics 服务器配置为 Commander 模式，则请选择“Commander 模式”。否则，请选择 “Mediator 模式”。

请忽略 “此服务器将在 HP 软件即服务 (SaaS) 环境中使用”复选框，因为此复选框用于 HP SaaS 管理员在 HP 自有系统上安装 Diagnostics 服务器（Commander 或 Mediator）。

在此阶段中，安装过程会有所不同，具体取决于是以 Commander 模式还是 Mediator 模式安装 Diagnostics 服务器。

- 要安装 Diagnostics Commander 服务器，请按照 “安装 Commander 模式的 Diagnostics 服务器”（第 59 页）中的说明继续操作。
- 要安装 Diagnostics Mediator 服务器，请按照“安装 Mediator 模式的 Diagnostics 服务器”（第 63 页）中的说明继续操作。

安装 Commander 模式的 Diagnostics 服务器

如果要在 Commander 模式中安装 Diagnostics 服务器，请继续执行如下操作：

4 选择时间同步方法。

要正确关联 Diagnostics 数据， Diagnostics 部署方案中的所有组件必须同步时间。可选择以下时间同步方法之一：

- **与 NTP 服务器同步。**此方法仅在 Diagnostics 服务器可以访问防火墙外部的 NTP 服务器时才适用。这是默认方法。
- **与注册的 Business Service Management 服务器同步。**如果 Diagnostics 服务器要在 Business Service Management 环境中工作，请选择此选项来与 Business Service Management 服务器同步。
- **与系统时间同步。**如果 Diagnostics 服务器不在 Business Service Management 环境中工作，并且无法访问 NTP 服务器，请选择此方法。

选择 “下一步”继续。

5 选择 Diagnostics 服务器的可选配置。

此服务器将用于 HP Business Service Management (BSM)。如果 Diagnostics Commander 服务器将与 Business Service Management 集成，请选中此框。

如果要与 Business Service Management 9.00 或更高版本集成，则选中此选项意味着将安装其他 OM 代理和 IAPA 组件，用于向 Business Service Management 发送运行状况指标。IAPA 是 Diagnostics 用于与 Business Service Management 进行通信的 OMi 代理的集成适配器策略激活组件。

Diagnostics Commander 服务器安装结束时，系统会提示您确认是否要安装这些组件。

有关与 BSM 集成所需的其他安装后配置，请参阅 “设置 Business Service Management 和 Diagnostics 之间的集成”（第 693 页）。此外，如果需要设置向 OM 服务器和 BSM 服务器发送报告，请参阅 “Diagnostics 和 OM 服务器共存”（第 712 页）获取说明。

选择要应用于此 Diagnostics 服务器的选项，然后选择 “下一步”继续。

如果选择 HP Business Service Management 选项，则将显示其他对话框；在其中，您可提供 HP Diagnostics 安装 DVD 中 Diagnostics 安装程序所在目录的路径。

注意：此步骤中需要使用 Diagnostics 安装光盘。

要从 Business Service Management 中的 “Diagnostics 配置”页面下载 Diagnostics Agent 和 Diagnostics Collector 安装程序，必须指定 Diagnostics 安装光盘中包含有这些安装程序的目录的路径 (**\Diagnostics_Installers)**。

输入 Diagnostics 安装程序在 Diagnostics 安装光盘中的路径，并选择“下一步”继续。

系统会将安装程序自动复制到 Business Service Management 可以访问的 Diagnostics 服务器安装目录。 **\Diagnostics_Installers** 目录的大小约为 1.85 GB，因此复制操作可能需要几分钟才能完成。

注意：可以跳过此步骤，而始终直接从 Diagnostics 安装光盘访问 Agent 和 Collector 安装程序。另外，也可以在以后手动执行此步骤，方法是将 Agent 和 Collector 安装程序从 Diagnostics 安装光盘 (**/Diagnostics_Installers**) 复制到 Business Service Management 可访问的 Diagnostics 服务器安装目录（**<Diagnostics 服务器安装目录>/html/opal/downloads**）。

---

6 指示电子邮件警报的 SMTP 设置 （可选）。

在安装过程中，这些 SMTP 设置是***可选***的。配置 SMTP 设置之后，可在 Diagnostics 服务器出现问题时发送电子邮件警报。如果希望以后配置（或以后修改）此设置，可跳过此对话框，并使用 Diagnostics 服务器的 “警报属性”页面配置这些设置 （有关详细信息，请参阅 《HP Diagnostics User's Guide》中有关警报的章节）。

- **SMTP 服务器**。 SMTP 服务器的主机名或 IP 地址。
- **SMTP 端口。** SMTP 服务器的端口号。
- **发件人电子邮件地址。**发送电子邮件消息的电子邮件地址。
- **管理警报电子邮件地址。**如果希望 Diagnostics 管理员在 Diagnostics 服务器出现问题时接收电子邮件警报，则可为管理员指定逗号分隔的电子邮件地址列表。当出现某些问题时，会发送警报，例如探测器生成大量要发送给服务器的服务器请求，服务器磁盘空间问题或来自 Commander 服务器的许可证检查警报。

  为 Diagnostics 管理员确定这些警报类型的阈值是在服务器 **server.properties** 文件中出厂配置的。有关详细信息，请参阅此文件中的注释，了解各种 **watchdog** 属性。另请参阅 “服务器上的磁盘空间问题”（第 840 页）。

7 检查预安装概要信息。

此时将显示已选择的安装设置。检查信息是否准确。

注意：对 Commander 模式 Diagnostics 服务器安装的估计总大小不包括代理安装程序的大小（如果在安装过程中为 Business Service Management 提供了这些安装程序）。

可以返回前面的安装步骤以更改设置。对于 Windows 环境，请单击“返回”。对于 UNIX 环境，请选择 “上一步”。

要启动 Diagnostics 服务器安装，请选择 “下一步”。

8 安装开始。

服务器安装开始。安装完成后，将看到安装后概要信息，或者如果在以上步骤 5a 中选择与 Business Service Management 集成，则会显示另一个对话框。

检查安装后概要并选择“完成”，可退出安装，或者如果选择与 BSM 集成，则继续下一个步骤。

9 OM 代理和 IAPA 组件安装复选框。

如果已选择在 Business Service Management 中使用 Diagnostics Commander 服务器，则会显示另一个对话框，以进行 OM 代理和 IAPA 组件安装。选中此框将安装这些组件。请注意，您可以不选中此复选框，并在稍候手动安装这些组件。有关详细信息，请参阅 “手动安装 OM 代理和 IAPA 组件”（第 72 页）。

将 Diagnostics 与 Business Service Management 9.00 或更高版本集成时，必须在 Diagnostics Commander 服务器上安装 OM 代理和 IAPA 组件。Diagnostics 将使用这些组件向 Business Service Management 网关服务器发送运行状况指标状态事件。如果要与较早版本的 Business Availability Center 集成，则不需要安装这些组件。

如果发生错误，将在 **<Diagnostics 安装目录 >/server/log.txt** 文件中报告错误。如有任何安装问题，请参阅 “OM 代理疑难解答”（第 875 页）。

要安装 OM 代理和 IAPA 组件，必须具有**根目录访问权限**。如果需要在没有根访问权限的情况下安装 Diagnostics 服务器，则可以选择不安装这两个组件，并在稍后手动安装它们。

如果已经在系统中安装了 OM 代理，则当 OM 代理组件版本较早时，这些安装程序会更新这些组件。

在 Windows 系统上安装这些组件时需要一段时间。

您可以不选中此框，跳过 OM 代理和 IAPA 组件安装，并在稍后安装它们。

不论是否选中此框都将始终存储安装程序位以供将来使用，以便以后在需要时从 **<Diagnostics 安装目录 >/server/setup/ovo-agent** 和 **/ovoiapa** 安装组件。有关详细信息，请参阅 “手动安装 OM 代理和 IAPA 组件”（第 72 页）。

安装这些组件之后，还需要在 Business Service Management 中执行其他一些配置步骤。有关详细信息，请参阅 “在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）。

在安装完成后，检查安装后概要信息，以确保安装已成功完成。

选择 “完成”退出安装。

注意：在 Windows 计算机上， Diagnostics 服务器将尝试自动启动。如果任何其他应用程序正在使用 Diagnostics 服务器默认端口，则 Diagnostics 服务器不会启动。有关如何手动启动 Diagnostics 服务器的说明，请参阅 “启动和停止 Diagnostics 服务器”（第 68 页）。

安装 Mediator 模式的 Diagnostics 服务器

如果要安装 Diagnostics Mediator 服务器，请继续执行如下操作。

如果要在 Mediator 模式中安装 Diagnostics 服务器，请继续执行如下操作：

4 提供 Commander 模式下 Diagnostics 服务器的位置。

提供用于将 Diagnostics Mediator 服务器连接到 Diagnostics Commander 服务器的信息。

- 输入 Diagnostics Commander 服务器的主机名或 IP 地址。
- 输入 Diagnostics Commander 服务器的端口。

Diagnostics Commander 服务器的默认端口是 **2006**。如果在安装 Diagnostics 服务器后更改了端口，请在此处指定该端口号而不是默认端口号。有关更改 Diagnostics 服务器端口的信息，请参阅 “更改默认 Diagnostics 服务器端口”（第 433 页）。

- 要使安装程序能够检查与指定的主机和端口的连接性，请选择 “检查 Diagnostics 服务器的连接性”。

如果不希望在此阶段检查是否存在连接性问题，请清除 “检查对 Diagnostics 服务器的连接”选项，以便继续安装。

选择 “下一步”继续。

如果需要安装程序执行连接性测试，则安装程序会在此时测试连接性。如果结果为负值，则安装程序会报告这些结果，然后继续执行下一个安装步骤。

5 指示电子邮件警报的 SMTP 设置 （可选）。

在安装过程中，这些 SMTP 设置是***可选***的。配置 SMTP 设置之后，可在 Diagnostics 服务器出现问题时发送电子邮件警报。如果希望以后配置（或以后修改）此设置，可跳过此对话框，并使用 Diagnostics 服务器的 “警报属性”页面配置这些设置 （有关详细信息，请参阅 《HP Diagnostics User's Guide》中有关警报的章节）。

- **SMTP 服务器**。 SMTP 服务器的主机名或 IP 地址。
- **SMTP 端口。** SMTP 服务器的端口号。
- **发件人电子邮件地址。**发送电子邮件消息的电子邮件地址。
- **管理警报电子邮件地址。**如果希望 Diagnostics 管理员在 Diagnostics 服务器出现问题时接收电子邮件警报，则可为管理员指定逗号分隔的电子邮件地址列表。当出现某些问题时，会发送警报，例如探测器生成大量要发送给服务器的服务器请求，服务器磁盘空间问题或来自 Commander 服务器的许可证检查警报。

  为 Diagnostics 管理员确定这些警报类型的阈值是在服务器 **server.properties** 文件中出厂配置的。有关详细信息，请参阅此文件中的注释，了解各种 **watchdog** 属性。另请参阅 “服务器上的磁盘空间问题”（第 840 页）。

6 检查预安装概要信息。

此时将显示已选择的安装设置。检查信息是否准确。

可以返回前面的安装步骤以更改设置。对于 Windows 环境，请单击“返回”。对于 UNIX 环境，请选择 “上一步”。

要启动 Diagnostics 服务器安装，请选择 “下一步”。

7 安装开始。

在安装完成后，检查安装后概要信息，以确保安装已成功完成。

选择 “完成”退出安装。

注意：在 Windows 计算机上， Diagnostics 服务器将尝试自动启动。如果任何其他应用程序正在使用 Diagnostics 服务器默认端口，则 Diagnostics 服务器不会启动。有关如何手动启动 Diagnostics 服务器的说明，请参阅 “启动和停止 Diagnostics 服务器”（第 68 页）。

验证 Diagnostics 服务器安装

要验证 Diagnostics 服务器安装是否正确，可检查 **<Diagnostics 服务器安装目录>/log/server.log** 文件，查找错误和警告。

也可启动 Diagnostics Enterprise UI，以验证此服务器是否正在运行。转至 **http://<Diagnostics_commander_server>:2006/**。此时，可使用默认的用户 / 密码即 “admin/admin”，或给定的登录名 （如果已为您设置此登录名）。但请注意，在企业中部署 Diagnostics 之后，应更改 “admin/admin”的默认登录名 （要配置用户、角色、权限和身份验证时，请参阅附录 B，“用户身份验证和授权”）。

请注意，要在 UI 中查看代理数据，还需要安装和配置 Diagnostics Agent 和 / 或 Collector 软件，以收集和向服务器报告性能数据，用以显示在 UI 中。

此外，还可检查系统运行状况视图，查找有关 Diagnostics 服务器及托管服务器的计算机的信息。

要访问系统视图，请执行以下操作：

1. 以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。
2. 在查询页面的列表中查找 Mercury 系统用户，并选择此链接打开 Diagnostics。
3. 登录 Diagnostics 并在 “应用程序” 窗口中选择 “整个企业”，然后选择任何链接打开 Diagnostics 视图。
4. 在 “视图” 窗格中，将显示 “系统视图” 视图组。打开视图组，并选择 “系统运行状况” 视图或 “系统容量” 视图。

Diagnostics 服务器静默安装

“静默安装” 是指无需用户交互的自动执行的安装。静默安装会在每个安装步骤接受来自响应文件的输入，以代替用户输入。

例如，需要在多台计算机上部署组件的系统管理员可以创建一个包含所有必要配置信息的响应文件，然后在多台计算机上执行静默安装。这样便无需在安装过程期间执行任何手动输入。

在多台计算机上执行静默安装之前，必须生成用于在安装过程期间提供输入的响应文件。此响应文件可用于需要在安装期间输入相同内容的所有静默安装。

重要信息：对于每个新版本的 Diagnostics，应在多台计算机上执行静默安装之前重新记录 Diagnostics 服务器的静默安装响应文件。

响应文件的后缀为 **.rsp**。可以使用标准的文本编辑器编辑响应文件。

要生成响应文件，请执行以下操作：

➤ 使用以下命令行选项执行常规安装：

```
<installer> -options-record <responseFileName>
```

此命令可创建其中包含在安装期间提交的所有信息的响应文件。

要执行静默安装，请执行以下操作：

➤ 使用相应的响应文件执行静默安装。

使用 **-silent** 命令行选项执行静默安装，如下所示：

```
<installer> -silent -options <responseFileName>
```

执行静默安装时，可以在指定响应文件名后指定下列两个附加选项。

➤ 通过指定 **-is:log < 日志文件路径 >** 选项，可以创建日志文件。

➤ 通过指定 **-is:tempdir< 临时目录路径 >** 选项，可以将临时目录更改为用户指定的目录。

启动和停止 Diagnostics 服务器

Diagnostics 服务器自动启动。如果需要手动启动或停止 Diagnostics 服务器，请按照以下说明执行操作。

适用于 Windows 计算机的说明

要在 Windows 计算机上启动 Diagnostics 服务器，请执行以下操作：

选择 “开始” > “所有程序” > “HP Diagnostics Server” > “Start HP Diagnostics Server”。

要在 Windows 计算机上停止 Diagnostics 服务器，请执行以下操作：

选择 “开始” > “所有程序” > “HP Diagnostics Server” > “Stop HP Diagnostics Server”。

适用于 Solaris 或 Linux 计算机的说明 （使用 Nanny）

nanny 是一个作为守护程序运行的进程，可以确保 Diagnostics 服务器始终处于运行状态。 nanny 还可以启动 LoadRunner 代理，允许对 LoadRunner 或 Performance Center 执行脱机数据整理。

下面是使用 nanny 启动和停止 Diagnostics 服务器的过程。

但请注意， m_daemon_setup 脚本不会配置服务器在系统启动之后自动重新启动。要执行此操作，需要将此启动与启动序列集成，或在每次系统启动之后手动执行启动。

要在 Solaris 或 Linux 计算机上启动 Diagnostics 服务器，请执行以下操作：

1 确保 **M_LROOT** 环境变量被定义为 Diagnostics 服务器 nanny 的根目录。例如，可以在 ksh 中输入以下内容：

```
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
```

在示例中，<platform> 是 Solaris、Linux 或 hpux。如果 **M_LROOT** 环境变量未定义为根目录，则会显示以下错误：

```
Warning: MDRV: cannot find lrun root directory. Please check your M_LROOT
Unable to format message id [-10791]
m_agent_daemon (is down)
```

2 将目录更改为 **$M_LROOT/bin**。

3 运行带有 **-install** 选项的 **m_daemon_setup**，例如：

```
cd $M_LROOT/bin
$./m_daemon_setup -install
```

在 Linux 中，如果遇到错误，则可能需要安装 libstdc++.so.5 共享库。

要在 UNIX 或 Linux 计算机上停止 Diagnostics 服务器，请执行以下操作：

1 将目录更改为 **$M_LROOT/bin**。

2 运行带有 **-remove** 选项的 **m_daemon_setup**，例如：

```
cd $M_LROOT/bin
$./m_daemon_setup -remove
```

适用于 Solaris 或 Linux 计算机的说明 （不使用 Nanny）

下面是未使用 nanny 启动和停止 Diagnostics 服务器的过程。

要在 Solaris 或 Linux 计算机上启动 Diagnostics 服务器，请执行以下操作：

运行 **<Diagnostics 服务器安装目录 >/bin/server.sh**。

要在 Solaris 或 Linux 计算机上停止 Diagnostics 服务器，请执行以下操作：

使用 **kill** 等实用程序终止进程。

向 Diagnostics 软件授予许可

Diagnostics 软件随附即时许可证，可使用它立即启动 Diagnostics。但是最终需要安装永久许可证密钥，可在 Diagnostics Commander 服务器上完成此步骤。有关请求和上载有效许可证文件的说明，请参阅第 3 章，“HP Diagnostics 许可”。

有关如何配置 Diagnostics 服务器的更多信息

将使用默认配置安装 Diagnostics 服务器，这样 Diagnostics 服务器可以立即开始工作。

不过，也可能需要更改该默认配置，以提高 Diagnostics 服务器的性能或允许 Diagnostics 服务器在特殊或复杂的情况下工作。有关 Diagnostics 服务器高级配置的信息，请参阅第 11 章，“Diagnostics 服务器高级配置”。

有关与 BSM 集成所需的其他安装后配置，请参阅“设置与其他 HP 软件产品的集成”（第 691 页）。此外，如果需要设置向 OM 服务器和 BSM 服务器发送报告，请参阅 “Diagnostics 和 OM 服务器共存”（第 712 页）获取说明。

确定 Diagnostics 服务器的版本

在请求支持服务时，您必须知道 Diagnostics 服务器的版本。在 Diagnostics Enterprise UI 中，“关于”对话框显示 Diagnostics 服务器的版本。通过从 Diagnostics Enterprise UI 工具栏的“帮助”菜单中选择“关于 HP Diagnostics”，可访问 “关于”对话框。

卸载 Diagnostics 服务器

下一节包含有关如何卸载 Diagnostics 服务器的说明。

重要信息：请注意，如果 OM 代理被其他产品使用，则不会将其与 Diagnostics 服务器一起卸载。要卸载 OM 代理和 IAPA 组件，必须在卸载服务器之前将这些组件卸载，因为它们的卸载程序位于服务器目录下面。

要从 Windows 计算机卸载 Diagnostics 服务器，请执行以下操作：

1 选择 “开始” > “所有程序” > “HP Diagnostics Server” > “Uninstall HP Diagnostics Server”，即可卸载 Diagnostics 服务器。

此外，也可以运行位于 **<Diagnostics 服务器安装目录>_uninst** 目录中的 **uninstaller.exe**。

2 在卸载过程中，会显示一条消息，询问您是否要删除特定文件。请执行以下操作：

- 要完全卸载 Diagnostics 服务器以及任何属性设置，请单击 “是” 或 “全是”。
- 如果计划重新安装 Diagnostics 服务器，并需要保留正在卸载的 Diagnostics 服务器的自定义属性设置，请将 **etc** 目录中的属性文件备份到新位置。

  如果已经备份这些文件，请单击 “是” 或 “全是”。

  如果未备份这些文件，请选择 “否” 或 “全部否”。

要从 UNIX 计算机卸载 Diagnostics 服务器，请执行以下操作：

可以在控制台模式或图形模式中卸载 Diagnostics 服务器。

1 停止 Diagnostics 服务器。有关说明，请参阅“启动和停止 Diagnostics 服务器”（第 68 页）。

2 将目录更改为根目录。

3 在 UNIX 命令提示符处输入以下内容：

➤ **在控制台模式中：**

```
<diagnostics_server_install_dir>/Server/_uninst/uninstaller.bin -console
```

➤ **在图形模式中：**

在图形模式中运行之前，导出显示内容。

```
export DISPLAY=< 主机名 >.0.0
```

```
<diagnostics_server_install_dir>/Server/_uninst/uninstaller.bin
```

手动安装 OM 代理和 IAPA 组件

用于 Diagnostics Commander 服务器的安装程序包含对 OM 代理和 IAPA 组件的安装，这些组件用于将运行状况指标状态事件发送到 Business Service Management 9.00 或更高版本。（如果要与较早版本的 Business Availability Center 集成，则不需要安装这些组件）。

在 Windows 上安装 OM 代理和 IAPA 组件的过程需要一段时间才能完成。在 Diagnostics 服务器安装期间，可以选择跳过安装这些组件，然后在 Diagnostics Commander 服务器上手动安装这些组件，具体操作如下：

如有任何安装问题，请参阅 “OM 代理疑难解答”（第 875 页）。

注意：在安装 Diagnostics 服务器期间，无论该服务器是 Commander 服务器还是中介服务器，都将存储 OM 代理安装程序和 IAPA 安装程序位以供将来使用，即使您选择稍后安装这些组件也是如此。

要在 Windows 系统上手动安装 OM 代理，请执行以下操作：

1 对于 Windows 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-agent/< 平台 >**，其中 < 平台 > 是 **win32** 或 **win64**。

2 在此目录的命令行中执行

cscript.exe opc_inst.vbs

要在 Linux 或 Solaris 系统上手动安装 OM 代理，请执行以下操作：

1 对于 Linux 或 Solaris 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-agent/< 平台 >**，其中 < 平台 > 是 **Linux 32**、**Linux 64** 或 **solaris**。

2 以根用户身份在此目录的命令行中执行

./opc_inst

要在 Windows 系统上手动安装 IAPA 组件，请执行以下操作：

1 对于 Windows 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-iapa/< 平台 >**，其中 < 平台 > 是 **win 32** 或 **win 64**。

2 在 win 32 目录的命令行中执行

cscript.exe < 安装目录 >/server/bin/install_ovo_iapa.vbs /i HPOprIAPA-09.00.111-WinNT4.0-release.msi < 日志文件 >

其中 < 日志文件 > 是记录安装结果的文件，路径可选。

或在 win 64 目录的命令行中执行

cscript.exe < 安装目录 >/server/bin/install_ovo_iapa.vbs /i HPOprIAPA-09.00.111-Win5.2_64-release.msi < 日志文件 >

其中 < 日志文件 > 是记录安装结果的文件，路径可选。

要在 Linux 或 Solaris 系统上手动安装 IAPA 组件，请执行以下操作：

1 对于 Linux 或 Solaris 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-iapa/< 平台 >**，其中 < 平台 > 是 **Linux 32**、**Linux 64** 或 **solaris**。

2 以根用户身份在 Linux32 目录的命令行中执行

rpm -ivh HPOprIAPA-09.00.111-Linux2.6-release.rpm

或者，以根用户身份在 Linux32 目录的命令行中执行

rpm -ivh HPOprIAPA-09.00.111-Linux2.6_64-release.rpm

或者，以根用户身份在 solaris 目录的命令行中执行

pkgadd -a ./noask_pkgadd -d HPOprIAPA-09.00.111-SunOS5.10-release.sparc HPOprIAPA

要完成 OM 代理安装，还必须执行以下操作：

➤ 要完成 OM 代理配置，必须完成向 Business Service Management 注册 Diagnostics 的步骤。有关 OM 代理的详细信息，请参阅“在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）。

手动卸载 OM 代理和 IAPA 组件

卸载 Diagnostics 服务器时，不会卸载 OM 代理和 IAPA 组件。要卸载 OM 代理和 IAPA 组件，必须在卸载服务器之前将这些组件卸载，因为它们的卸载程序位于服务器目录下面。并且必须按以下顺序卸载这些组件：首先卸载 IAPA 组件，然后再卸载 OM 代理。

要在 Windows 系统上手动卸载 IAPA 组件，请执行以下操作：

1 对于 Windows 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-iapa/< 平台 >**，其中 < 平台 > 是 **win 32** 或 **win 64**。

2 在 win 32 目录的命令行中执行

cscript.exe < 安装目录 >\server\bin\install_ovo_iapa.vbs /x HPOprIAPA-09.00.111-WinNT4.0-release.msi uninstall.log

或在 win 64 目录的命令行中执行

cscript.exe < 安装目录 >\server\bin\install_ovo_iapa.vbs /x HPOprIAPA-09.00.111-Win5.2_64-release.msi uninstall.log

要在 Linux 或 Solaris 系统上手动卸载 IAPA 组件，请执行以下操作：

1 对于 Linux 或 Solaris 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-iapa/< 平台 >**，其中 < 平台 > 是 **Linux 32**、 **Linux 64** 或 **solaris**。

2 以根用户身份在 Linux32 或 Linux 64 目录的命令行中执行

rpm -e HPOprIAPA

或者，以根用户身份在 solaris 目录的命令行中执行

pkgadd HPOprIAPA

要在 Windows 系统上手动卸载 OM 代理，请执行以下操作：

1 对于 Windows 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-agent/< 平台 >**，其中 < 平台 > 是 **win32** 或 **win64**。

2 在此目录的命令行中执行

cscript.exe opc_inst.vbs -r

要在 Linux 或 Solaris 系统上手动卸载 OM 代理，请执行以下操作：

1 对于 Linux 或 Solaris 系统，将目录更改为 **<Diagnostics 安装目录 >/server/setup/ovo-agent/< 平台 >**，其中 < 平台 > 是 **Linux 32**、 **Linux 64** 或 **solaris**。

2 以根用户身份在此目录的命令行中执行

./opc_inst -r

3

HP Diagnostics 许可

HP Diagnostics 要求用户向 Diagnostics 命令服务器上载有效的许可证。

本章包括：

关于 HP Diagnostics 许可

可使用上载到 Diagnostics 命令服务器的文件为 Diagnostics 授予许可。您可以从 HP 软件客户支持代表处获取此许可证。

当 Java 代理、.NET 代理和 Diagnostics Mediator 服务器首次与 Diagnostics 命令服务器连接时，会根据 Diagnostics 命令服务器上安装的许可证为它们授予许可。

如果希望 Diagnostics 管理员接收许可证检查警报，则在安装 Commander 服务器时，请在 “SMTP 设置”安装对话框中指定逗号分隔的管理警报电子邮件地址。或者，也可使用 Commander 服务器的 “警报属性”页面在安装后设置管理员地址。

许可证类型

在安装期间，您将获得一个产品随附的 “即时启动许可证” 。使用该即时启动许可证，可以安装 Diagnostics 组件，开始监控应用程序，并可以处理性能度量。即时启动许可证在从安装时刻开始或从首次使用产品开始的固定时间段内有效。

在此时间段内，您必须获取 “永久许可证”，或申请 “评估许可证”以延长评估期。 Diagnostics 使用评估许可证提供许可证密钥，以此来扩展客户对产品的评估。评估许可证在固定时间段内有效。

如果在获得永久许可证之前即时启动许可证 （或已延长的评估许可证）到期了，则 Diagnostics 服务器将发出提醒消息。

注意：完整的 Diagnostics 产品附带有即时启动许可证。在您提供有效的许可证文件之前，各独立 Diagnostics Profiler 中存在负载限制。

永久许可证通常为特定容量 （请参见 “基于当前连接的探测器许可证信息”（第 83 页））。一旦安装许可证密钥，Diagnostics 就会针对此容量计算使用情况。

对于 Diagnostics，有两种类型的 LTU （要使用的许可证）：

- AM 许可证 - 在应用程序管理 / 企业模式中使用产品时，通常用于生产环境。使用 AM 许可证的代理也可与 LoadRunner/Performance Center 配合使用。
- AD 许可证 - 在预生产加载测试环境中的 Diagnostics 模式下使用产品时，用于 LoadRunner/Performance Center 运行。

  随 Diagnostics 一同接收的即时启动许可证存在以下时间和容量限制：AM - 60 天且容量为 50， AD - 14 天且容量为 50。

  超出限制后将看到提醒消息。有关 AD 和 AM 许可证的详细信息，请参见 “基于当前连接的探测器许可证信息”（第 83 页）。

对 Commander 模式下的 Diagnostics 服务器授予许可

从 HP 软件客户支持代表处获取 Diagnostics 许可证。在无需手动从每个系统检索信息的情况下确定所需的许可证数目时，下述 “许可证管理”页面所包含的信息十分有用。该信息仅对 Diagnostics 8.00 或更高版本的探测器可用。

您将从 HP 接收许可证证书以验证许可证购买条款。在输入与软件产品购买关联的销售订单号（对每个订单而言是唯一的）之后，将发出许可证密钥 / 密码。此订单号显示在许可证换领表单上，同时还显示在与订单运输和包装相关的所有文本中。

将许可证文件存储在可通过 Diagnostics 命令服务器的“许可证管理”页面访问的目录中。然后，如以下步骤所述将其上载到 Diagnostics Commander 服务器中。

重要信息：对于许可证版本低于 Diagnostics 9.10 的客户，旧许可证仍然可以使用 9.10，不过以下部分将描述如何使用新的许可过程来购买新的 Diagnostics 9.10 或更高版本。

要为 Diagnostics 部署授予许可，请执行以下操作：

1. 通过访问 Diagnostics Enterprise UI 来访问 Diagnostics 命令服务器的“许可证管理”页面 (http://<Diagnostics_ 服务器 >:2006)
2. 输入登录名和密码。可以使用默认登录名，也可以使用已创建和已分配的登录名。默认登录名为 admin，默认密码为 admin。
3. 选择 “配置诊断程序”。
4. 选择 “License”链接。此时 “许可证管理”页面将打开，其中将提供以下信息：
   - 有关当前许可证的信息。
   - 用于上载从 HP 软件支持处收到的许可证的实用程序。
   - 受监控环境中应用程序服务器实例总数以及当前连接的探测器数的信息。您还可以找到有关 Diagnostics AD 和 AM 许可证容量使用情况的信息。


Diagnostics
许可证管理
AM 许可证信息
属性
值
许可证:
HP Diagnostics AM Implicit InstantOn License
类型:
Instant-on
开始日期:
2012年6月8日 星期五
到期时间:
2012年8月7日 星期二
持续时间 (天):
60
剩余天数:
34
容量:
50
自动传递许可证上载
注意:
上载的文件将被添加到 "DiagnosticsLicFile.txt"中。
许可证文件:
Browse...
上载
服务器许可证上载（过时）
注意:
您仅可以上载以 ".lic" 结尾的文件。上载的文件将被重命名为 "DiagnosticsServer.lic"。
许可证文件:
Browse...
上载
基于当前连接的探测器许可证信息
客户名称: Default Client
属性
值
应用程序服务器实例总数:
1
Load Runner/Performance Center (AD 许可证) 实例:
0
应用程序管理/企业模式 (AM 许可证) 实例:
1
.NET 进程:
0
Java 探测器:
1
旧 .NET 探测器:
0
未知探测器:
0
采集器实例:
1
刷新

5 收到适用于 Diagnostics 部署的许可证文件后，使用 “许可证管理”页面的 “自动传递许可证上载”部分上载此文件。

“服务器许可证上载 (过时)”部分已过时，只有当在服务器上安装了 Diagnostics 之前使用过的许可证密钥类型 (.lic file) 或仅安装了即时启动许可证时，才会显示此部分。此上载功能针对其许可证版本低于 Diagnostics 9.10 版的现有客户而提供，允许上载旧许可证。

注意：请不要尝试将许可证文件直接复制到 Diagnostics 服务器安装目录下，请始终使用 “许可证管理”页面的 “自动传递许可证上载”部分上载此文件。

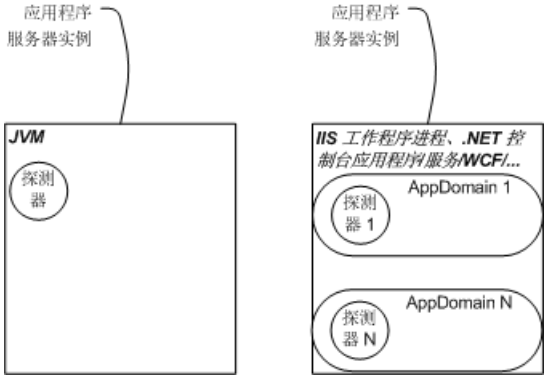
键入许可证文件的存储位置的路径，或者单击 “Browse”以导航到许可证文件位置。单击 “上载”，将许可证文件应用到 Diagnostics 服务器。

如果成功 （许可证文件中的密钥有效且未过期），则上载过程会将许可证添加为 **DiagnosticsLicFile.txt** 并存储在 Diagnostics Commander 服务器的 **<Diagnostics 安装目录 >/etc** 目录中。使用自动传递许可，您就可以上载已添加到许可证文件的增量许可证 （混合使用旧许可证时不能执行此操作）。

查看许可证信息

“许可证管理”页面中将会报告有关当前许可证的信息。您可以查看许可证类型、到期日期 （如果有）、最大应用程序服务器数 （JVM 和 .NET 进程）以及许可证中包含的采集器实例数。

您还可以查看基于当前连接的探测器许可证信息。此信息包含当前受 Diagnostics 监控的应用程序服务器实例数、 .NET 进程数、 JVM 实例数和采集器实例数。



基于当前连接的探测器许可证信息

在 “许可证信息”部分中，您将看到基于当前连接的探测器的计数 （请参见下例）。在需要不手动从每个系统检索信息的情况下确定所需的许可证数目时，该信息十分有用。该信息仅对 Diagnostics 8.00 或更高版本的探测器可用。

基于当前连接的探测器许可证信息

客户名称: Default Client

属性	值
应用程序服务器实例总数：	0
Load Runner/Performance Center (AD 许可证) 实例：	0
应用程序管理/企业模式 (AM 许可证) 实例：	0
.NET 进程：	0
Java 探测器：	0
IB .NET 探测器：	0
未知探测器：	0
采集器实例：	0
	刷新
	详细信息

➤ **应用程序服务器实例总数。**应用程序服务器实例是 Java 代理实例或 .NET 代理实例。Java 代理实例 =“探测器”。.NET 代理实例 = .NET 工作程序进程。此值是“Load Runner/Performance Center 实例”和“应用程序管理 / 企业模式实例”的总和。许可证容量必须涵盖此总值。

LoadRunner/Performance Center（AD 许可证）实例。在 AD 模式中配置以及在 LoadRunner 或 Performance Center 运行中主动调用的 Java 探测器 /.NET 进程实例数。对于 HP Diagnostics AD 许可证容量，只会计算活动 AD 模式探测器 / 进程。不会计算运行以外的探测器 / 进程。

在安装代理时，系统将提示您指定是否在 AD 模式下配置代理，以用于 LoadRunner 和 Performance Center 运行。如果选择 AD 许可证选项，则将在 Diagnostics 中设置以下值：

对于 Java 代理 - 在安装 Java 代理时， **etc/probe.properties** 文件中 **active.products** 属性的值设置为 **AD** 模式 （请参阅 “设置活动产品模式”（第 461 页））。可以通过在安装后修改此属性来更改模式值。

对于 .NET 代理 - 在安装 .NET 代理时，**probe_config.xml <modes>** 元素的值设置为 **ad** 模式 （请参见 “<modes> 元素”（第 548 页））。可以通过在安装后修改此元素来更改模式值。

在 AD 模式下运行探测器的好处在于，您只需要可容纳当前在 LoadRunner 或 Performance Center 测试运行中的探测器数的许可证容量。例如，如果在测试系统上安装了 20 个代理，但每次测试运行中只会有 5 个探测器，那么就只需要可容纳 5 个探测器的 AD 许可证容量。

➤ **应用程序管理 / 企业模式（AM 许可证）实例。**在生产环境中安装且设置为企业或 AM 模式的 Java 代理和 .NET 进程实例数。它们将纳入 HP Diagnostics AM 许可证容量计算。

在安装代理时，系统将提示您指定是否要在应用程序管理 / 企业模式 （AM 许可证）下配置代理，以便在生产环境中使用 Diagnostics 服务器。如果选择此模式，则将在 Diagnostics 中设置以下值：

对于 Java 代理 - 在安装 Java 代理时， **etc/probe.properties** 文件中 **active.products** 属性的值设置为 **Enterprise** 模式 （请参阅 “设置活动产品模式”（第 461 页））。可以通过在安装后修改此属性来更改模式值。

对于 .NET 代理 - 在安装 .NET 代理时，**probe_config.xml <modes>** 元素的值设置为 **enterprise** 模式（请参见 “<modes> 元素”（第 548 页））。可以通过在安装后修改此元素来更改模式值。

对于设置为企业模式的代理，将针对 HP Diagnostics AM 许可证容量计算探测器。

- **.NET 进程。**执行插桩以便由一个或多个 .NET 探测器监控的所有进程 （应用程序域）。例如， IIS 工作进程或 .NET 控制台应用程序 / 服务 /WCF。在许可证报告中，可看到低于 8.00 版本的旧 .NET 探测器的数目。
- **Java 探测器。**受监控的 java 或 javaw 进程，或嵌入 JVM 的任何其他进程。这相当于 Java 探测器。
- **采集器实例。**采集器实例数显示在许可证页面上，但不会针对任何许可证计算这些实例。采集器实例包括以下几种：
  - Oracle - （所执行的） Oracle 软件 （Oracle 进程）中的实例，以及它们使用的内存 (SGA) 中的实例。一个 SID 可标识一个实例。使用 **oracle-config.xml** 中的 <oracleInstance> 条目配置的要监控的实例将包括在内。
  - SQL Server - 主要应用于数据库引擎及其支持组件的实例。使用 **sqlserver-config.xml** 中的 <sqlserverInstance> 条目配置的要监控的实例将包括在内。
  - WebSphere MQ - 使用 **mq-config.xml** 中的 <mqInstance> 条目配置的要监控的实例将包括在内。
  - TIBCO EMS - 使用 **tibco-ems-config.xml** 中的 <emsInstance> 条目配置的要监控的实例将包括在内。
  - WebMethods 代理 - 使用 **wm-broker-config.xml** 中的 <WmBrokerInstace> 条目配置的要监控的实例将包括在内。

- SAP/ABAP - 搜寻到的每个对话实例 （SAP ABAP 探测器）将包括在内。
- VMware - **vmware-config.xml** 文件中指定的 vSphere 服务器数将包括在内。

- 版本低于 8.0x 的所有探测器将列在旧探测器下。

许可证详细信息

选择“许可证”页面底部的“详细信息”链接将显示每个 .NET 探测器、Java 探测器和采集器的详细信息。详细信息包括探测器名称、主机名和端口或 PID、运行 ID （用于 LoadRunner/Performance Center 加载测试运行中的探测器）、探测器版本和产品模式。

以下为 “许可证管理详细信息”页面：

Diagnostics

许可证管理

以下对象的详细信息 Default Client

.NET Probes

.NET processes (hostname:pid)	Probe Name	Mode	Run ID	Version

Java Probes:

Probe Name	Mode	Run ID	HostName:Port	Version
QCJavaAgent	Enterprise, PRO		win-7x12evcu1zb:35002	9.20.57.1197

Old .NET probes not reporting their PID:

Probe Name	Mode	Run ID	CLR Info	Version

Unknown Probes (no PID or VM info missing):

Probe Name	Mode	Run ID	VM Info	Version

Collectors:

Name	Instances	Mode	Run ID	Version
RegCollector1	1	Enterprise		9.20.40.770
SQLCollecotr	1	Enterprise		9.20.40.770
collector1	0	Enterprise		9.20.40.770
Collector2-Ada	0	Enterprise		9.20.40.770

HP Diagnostics 服务器 "server-BSMVM0111JA"，版本 9.20.60.1257

为其他 Diagnostics 组件授予许可

作为 Mediator 运行的 Diagnostics 服务器和 Diagnostics Agent 没有独立的许可证，它们的许可证基于 Diagnostics 命令服务器的许可证。这些组件首次连接到已授予许可的 Diagnostics 命令服务器时， Diagnostics Agent 和 Diagnostics Mediator 服务器将自动获得许可。

安装 Java 代理或 .NET 代理时，将会自动安装 Diagnostics Profiler。您可以在探测器实体的上下文中查看 Diagnostics Profiler。 Diagnostics Profiler 是独立的 UI，您可在装有代理的系统上直接访问它，也可以通过 HP Diagnostics UI 访问它。

在探测器能够连接到已获得正确许可的 Diagnostics 命令服务器之前， Diagnostics Profiler 可在具有负载限制的未许可模式下运行。在未许可模式下， Profiler 只能通过 5 个并发线程捕获数据。

4

安装 Diagnostics Collector

可以在 Windows 和 UNIX 计算机上安装 Diagnostics Collector。

本章包括：

➤ 确定 Diagnostics Collector 的版本 （第 128 页）

➤ 卸载 Diagnostics Collector （第 128 页）

关于安装 Diagnostics Collector

Diagnostics Collector 可从远程系统收集数据。可以将 Collector 配置为从以下类型的活动系统收集性能数据：

- SAP NetWeaver–ABAP
- Oracle 数据库 （包括 Oracle RAC）
- IBM WebSphere MQ
- TIBCO Enterprise Message Service (EMS)
- Software AG webMethods 代理
- SQL Server 数据库
- VMware vCenter 或 VMware ESX Server

在安装 Collector 期间，可以选择监控这些活动系统中的任意一种。安装之后，可以定义要监控的 Oracle 数据库、SQL Server 系统、VMware vCenter 或 VMware ESX Server、IBM WebSphere MQ 消息传递系统、TIBCO EMS 系统、Software AG webMethods 代理和 SAP NetWeaver–ABAP 系统的实例。每个受监控的实例都表示为一个探测器实体。可以为每个 Collector 配置多个探测器。

注意：可以在任何计算机上安装 Diagnostics Collector，而不必将其安装在 SAP、Oracle、MQ、Tibco EMS、webMethods 代理、VMware 或 SQL Server 应用程序的主机上。有关 Collector 主机的要求，请参见 “Diagnostics Collector 主机的要求”（第 37 页）。

访问 Collector 安装程序

可以从 Diagnostics 安装磁盘启动安装，也可以将可执行安装文件复制到其他位置，然后运行该文件，或从 Business Service Management 的 Diagnostics “下载”页面中选择该文件。

要从 Diagnostics 安装位置访问安装程序，请执行以下操作：

- 对于 Windows，Diagnostics 安装 DVD (Autorun.exe) 将显示安装菜单页面。从菜单中选择 “Diagnostics Collector”启动安装程序。
- 或者，也可以运行适当的安装程序，方法是在安装位置查找 **HPDiagCollector_<版本号>_win.exe** 文件（用于 Windows）或 **HPDiagCollector_<版本号>_<平台>.bin** 文件 （用于 Unix），将文件复制到新的安装位置并双击 .exe 文件，或运行 .bin 安装程序。

  按照 “安装 Collector”（第 92 页）中的说明继续操作。

要从 HP 软件下载中心下载安装程序，请执行以下操作：

1. 转至 HP 软件网站的软件下载中心。
2. 查找 **Diagnostics** 下载信息，并选择下载 Diagnostics Collector 软件的相应链接。
3. 按照网站上的下载说明进行操作。

   按照 “安装 Collector”（第 92 页）中的说明继续操作。

要从 Business Service Management Diagnostics “下载”页面下载安装程序，请执行以下操作：

1 在 “Business Service Management”的顶部菜单中选择 “管理” > “Diagnostics”，然后单击 “下载”选项卡。

2 在 “下载”页面上单击相应链接，下载相应的 Collector 安装程序。

注意：如果将 Collector 安装程序放置到 Business Service Management 访问所需的目录中，则可以在 Business Service Management 中找到该安装程序。通过提供 Diagnostics Agent 和 Collector 安装程序的路径，可以在 Diagnostic 服务器安装期间启用此下载，或者可以手动将文件从安装光盘复制到 Diagnostics 服务器安装目录的 **<Diagnostics 服务器安装目录 >/html/opal/downloads** 文件夹。请参见第 2 章，“安装 Diagnostics 服务器”（第 51 页）

按照 “安装 Collector”（后续）的说明继续操作。

安装 Collector

以下步骤提供了在 Windows 或 Unix 系统上安装 Collector 的详细说明。

有关其他安装类型的信息，请参见：

- 有关对除 Linux 和 Solaris 外的平台使用常规 Unix 安装程序的说明，请参见“使用常规安装程序安装 Diagnostics Collector”（第 102 页）
- 有关静默安装的信息，请参见“静默安装 Diagnostics Collector”（第 101 页）。

注意：临时目录的可用空间约为 400 MB。

注意：如果主机上存在预安装的 Collector，则必须按照说明升级此 Collector，而不遵循这些安装说明，请参见 “升级和修补程序安装说明”（第 851 页）。

图形模式中的 Windows 安装程序和 Unix 安装程序显示相同屏幕。如果在控制台模式下运行 Unix 安装程序，则会显示提示而非屏幕，但是流程仍然与本节中的说明一致。

对于 Unix 安装程序，如有需要，可更改安装程序文件的模式以使其可执行。

➤ 要以图形模式运行 Unix 安装程序，请在 UNIX 命令提示符处输入 <安装程序> 可执行文件，其中的 < 安装程序 > 可以是如下的示例内容：

HPDiagCollector_< 版本号 >_sol.bin

HPDiagCollector_< 版本号 >_linux.bin

安装程序所显示的屏幕与 Windows 安装程序所显示的屏幕相同。

➤ 要以控制台模式运行 Unix 安装程序，请在 UNIX 命令提示符处输入 < 安装程序 > -console。

以下说明假定您了解 UNIX 控制台屏幕和命令。有关 UNIX 屏幕和命令的更多信息，请参见 “使用 UNIX 命令”（第 881 页）。在控制台模式下运行的安装程序将显示一系列提示。

启动安装程序之后，将打开软件许可证协议。

要安装 Collector，请执行以下操作：

1 接受软件许可证协议。

阅读协议并选择 “我接受许可证协议条款”。在控制台模式下，按 **Enter** 键继续处理许可证协议，在出现提示时输入 **1** 接受协议。

选择 “下一步”继续。

2 指定 Collector 的安装位置。

在 “安装目录名”框中，接受默认目录 **C:\MercuryDiagnostics\Collector**，或键入要安装 Collector 的目录的名称。也可单击 “浏览”导航到其他目录。在本文档中指的是 <Collector 安装目录 >。

如果目录中包含需要升级的已有 Collector，请取消此安装，然后按照附录 G，“升级和修补程序安装说明”中所述的 Collector 升级步骤操作。

选择 “下一步”继续。

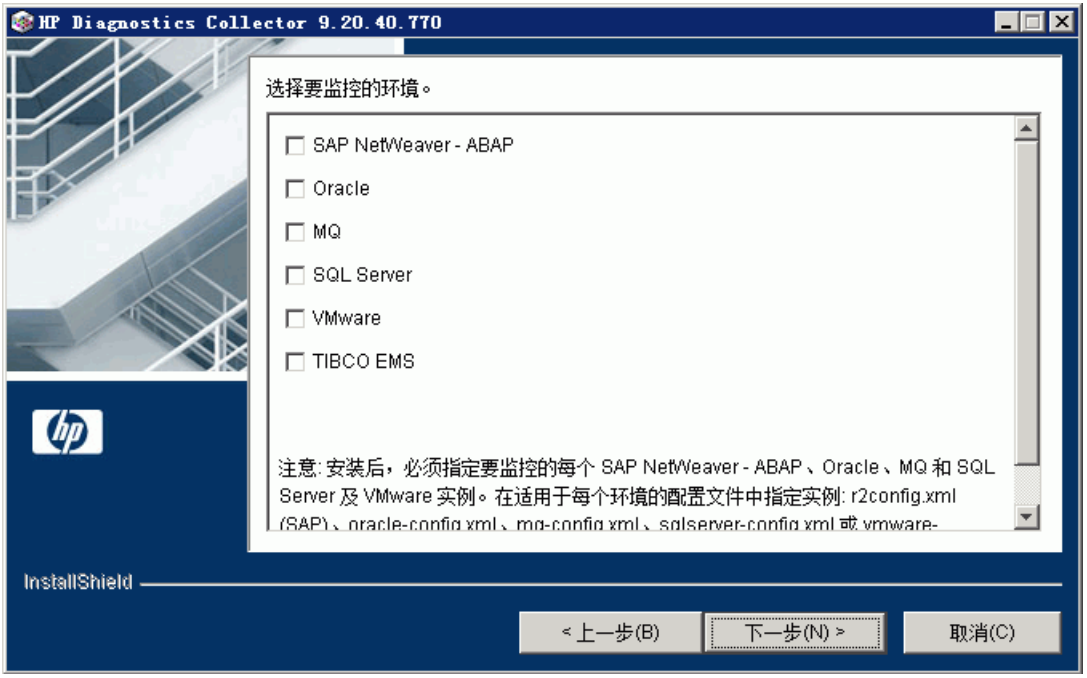
3 为 Collector 分配唯一的名称。

为 Collector 分配一个可唯一标识此特定 Collector 的名称。

可以在名称中使用 “-”、“_” 及所有字母数字字符。

选择 “下一步” 继续。

4 选择要监控的环境。



选择要应用于此 Collector 的选项，可以选择一个或多个选项。

- 要收集 SAP NetWeaver–ABAP 环境中的数据，请选择 “SAP NetWeaver–ABAP”。
- 要收集 Oracle 10g 数据库服务器中的数据，请选择 “Oracle”。
- 要收集 MQ 系列环境中的数据，请选择 “MQ”。
- 要收集 SQL Server 数据库中的数据，请选择 “SQL Server”。
- 要收集 VMware vCenter 或 VMware ESX Server 中的数据，请选择 “VMware”。
- 要收集 TIBCO EMS 环境中的数据，请选择 “TIBCO EMS”。
- 要收集 webMethods 代理系统中的数据，请选择 “WebMethods 代理”。

重要信息：在安装之后，必须指定每个要监控的 SAP NetWeaver–ABAP、Oracle、MQ、TIBCO EMS、SQL Server、webMethods 代理和 VMware 实例。您可以在安装程序附带的 XML 文件中手动定义这些实例。有关详细信息，请参阅 “配置活动系统属性文件”（第 104 页）。

选择 “下一步” 继续。

5 提供有关 Diagnostics Mediator 服务器的信息。

提供用于启用与 Diagnostics Mediator 服务器的通信的详细信息。

如果在将运行 Collector 的 Diagnostics 部署方案中只有一个 Diagnostics 服务器，请输入该 Diagnostics 服务器的主机名以及其事件端口信息。

如果在部署方案中有多个 Diagnostics 服务器，请输入用于从 Collector 收集事件的 Diagnostics Mediator 服务器的信息。

a 在 “Diagnostics 服务器 Mediator 主机”框中，键入 Diagnostics Mediator 服务器主机的主机名或 IP 地址。

注意：必须指定完全限定主机名。在包含 UNIX 的混合操作系统环境中，必须指定完全限定主机名，才能保证网络路由正常工作。

---

b 在 “Diagnostics 服务器 Mediator 端口”框中，键入 Diagnostics 服务器用于侦听 Collector 通信的端口号。默认端口号是 **2006**。如果在安装 Diagnostics 服务器后更改了端口，请指定该端口号而不是默认端口号。

c 要确保 Diagnostics 服务器正在运行并可从安装主机对其进行访问，请选中 “检查对 Diagnostics Server Mediator 主机和端口的连接”。

选择 “下一步”继续。

如果在选中“检查对 Diagnostics Server Mediator 主机和端口的连接”后遇到了连接问题，请查看安装程序提供的连接性检查结果。如果您不希望在此阶段解决这些问题，请清除“检查对 Diagnostics Server Mediator 主机和端口的连接”复选框，然后继续安装，并在稍后再解决问题。

6 如果在步骤 4 中选择了 “SAP NetWeaver–ABAP”，请提供 SAP Java 连接器的位置。

在 “SAP Java 连接器安装目录” 框中，输入 SAP Java 连接器的安装目录路径。在安装了此 Collector 的系统上，安装程序会将必要的文件复制到 **<Collector 安装目录>\lib** 目录中。

此目录必须包含以下文件：

- sapjco.jar
- librfc.dll 或 librfc32.dll 或 librfccm.so
- sapjcorfc.dll 或 libsapjcorfc.so

如果不知道 SAP Java 连接器目录的名称或缺少任何以上文件，请与 SAP 代表联系。

7 请记住在安装后将需要的文件复制到 Collector 系统中。

如果选择了 Tibco EMS，则将看到一个提醒，告诉您在安装后复制以下第三方 jar：**tibjms.jar**、**tibjmsadmin.jar**。这些文件通常位于 **<Tibco_EMS>/ems/<version>/lib** 目录的 TIBCO EMS 安装中，您可将其复制到安装了 Collector 的系统上的 **<Collector 安装目录 >\lib** 目录中。

如果选择了 webMethods 代理，则将看到一个提醒，告诉您在安装后复制以下第三方 jar：**wm-brokerclient.jar**、 **wm-g11nutils.jar**。这些文件通常位于 **<SoftwareAG>/common/lib** 目录的 Software AG 安装中，您可将其复制到安装了 Collector 的系统上的 **<Collector 安装目录 >\lib** 目录中。

8 检查预安装概要。

此时将显示已选择的安装设置。检查信息是否准确。

要选择不同的安装设置，请单击 “返回”（或在控制台模式下选择 “上一步”）。

要开始安装，请单击 “下一步”。

9 安装完成。

安装完成后将显示一条消息，确认 Collector 已成功安装。选择 “完成”退出安装程序。

10 为活动系统配置 XML 文件。

在步骤 4 中，已选择了要监控的活动系统。必须为这些要监控的活动系统配置相关属性，以使 Collector 主机和活动系统主机能够进行通信。

有关用于配置相关活动系统属性的说明，请参见 “配置活动系统属性文件”（第 104 页）。

11 验证是否已正确安装并正在运行 Collector。

安装完成后，Diagnostics Collector 会自动运行。通过检查 **collector.log** 文件中的错误，可以验证 Collector 安装。有关详细信息，请参见 “验证 Diagnostics Collector 安装”（第 125 页）。

在 Diagnostics UI 中，会将每个 Collector 实例表示为以下系统类型的探测器实体：Oracle 探测器、SAP 探测器、MQ 探测器、EMS 探测器、WM 探测器或 SQL Server 探测器。

静默安装 Diagnostics Collector

“静默安装”是指无需用户交互的自动执行的安装。静默安装会在每个安装步骤接受来自响应文件的输入，以代替用户输入。

例如，需要在多台计算机上部署组件的系统管理员可以创建一个包含所有必要配置信息的响应文件，然后在多台计算机上执行静默安装。这样便无需在安装过程期间执行任何手动输入。

在多台计算机上执行静默安装之前，必须生成用于在安装过程期间提供输入的响应文件。此响应文件可用于需要在安装期间输入相同内容的所有静默安装。

重要信息：对于每个新版本的 Diagnostics，应该先重新记录 Diagnostics Collector 的静默安装响应文件，然后在多台计算机上执行静默安装。

响应文件的后缀为 **.rsp**。可以使用标准的文本编辑器编辑响应文件。

要生成响应文件，请执行以下操作：

➤ 使用以下命令行选项执行常规安装：

```
<installer> -options-record <responseFileName>
```

此命令可创建其中包含在安装期间提交的所有信息的响应文件。

要执行静默安装，请执行以下操作：

➤ 使用相应的响应文件执行静默安装。

使用 **-silent** 命令行选项执行静默安装，如下所示：

```
<installer> -silent -options <responseFileName>
```

执行静默安装时，可以指定两个附加选项。

➤ 通过在响应文件名后指定 **-is:log < 日志文件路径 >** 选项，可以创建日志文件。

➤ 通过在响应文件名后指定 **-is:tempdir < 临时目录路径 >** 选项，可以将临时目录更改为用户指定的目录。

使用常规安装程序安装 Diagnostics Collector

可使用 Diagnostics Collector 安装程序在 Windows、Linux 和 Solaris 系统上安装 Diagnostics Collector。在安装光盘中提供的常规 Unix 安装程序可用于在 HP-UX 和 AIX 等其他平台上安装 Collector。

要使用常规安装程序安装 Diagnostics Collector，请执行以下操作：

1 在 HP Diagnostics 安装光盘的 Diagnostics 安装程序文件夹中找到 **HPDiagCollector_< 发行版本 >_unix.zip**。

2 在要安装 Collector 的系统中解压缩该文件。

3 然后，必须对每个要监控的活动系统配置相关属性，以使 Collector 主机和活动系统主机能够进行通信。有关用于配置活动系统属性的说明，请参见 “配置活动系统属性文件”（第 104 页）。

如何在安装 Collector 之后手动添加其他收集类型

在 Collector 的初始安装期间，可选择要监控的不同收集类型或活动系统类型，例如 SAP 或 Oracle。在安装 Collector 后，可以手动添加其他收集类型或活动系统类型。

要手动添加其他活动系统类型，请执行以下操作：

1 手动将任何需要的文件复制到 **<Collector 安装目录 >\lib** 目录。SAP、Tibco EMS 和 webMethods 代理活动系统需要这些文件。有关所需文件的详细信息，请参见安装说明。其他收集类型 （如 Oracle 或 SQL Server）不需要此步骤。

2 在安装了 Collector 的系统上，在 **<Collector 安装目录>\etc\collector.properties** 文件中，编辑 **active.systems** 属性以添加其他收集类型。有效值 （区分大小写）包括 SAP_R3、 Oracle、 MQ、 SQL_Server、 VMWARE、 EMS、 WM_BROKER。

配置活动系统属性文件

在安装 Collector 时，系统会要求您指明收集类型 （Collector 要监控的活动系统）。在安装之后，可定义要监控的活动系统的实例。您可以在 Collector 安装程序附带的 XML 文件中手动定义这些实例。XML 文件中的实例定义将视为活动系统的探测器实体。有关配置说明，请参阅以下各节：

- “针对 SAP NetWeaver–ABAP 进行配置”（第 104 页）
- “针对 Oracle 进行配置”（第 108 页）
- “针对 SQL Server 的配置”（第 111 页）
- “针对 MQ 的配置”（第 115 页）
- “针对 TIBCO EMS 的配置”（第 118 页）
- “针对 webMethods 代理的配置”（第 119 页）
- “针对 VMware 的配置”（第 121 页）

针对 SAP NetWeaver–ABAP 进行配置

SAP NetWeaver–ABAP 系统部署可以包括一个或多个 SAP NetWeaver–ABAP 应用程序实例。这些实例一起组成了 SAP NetWeaver–ABAP 系统。

根据用户权限的不同，用户可直接访问系统或系统中的应用程序实例，也可能需要通过 SAP 消息服务器连接才能访问它们。对于每个 SAP NetWeaver–ABAP 探测器实体，必须了解所使用的连接选项。

可以将 Collector 配置为收集要监控的每个活动 SAP NetWeaver–ABAP 系统实例的数据。可在 **<Collector 安装目录 >\etc\r3config.xml** 文件中配置要监控的 SAP NetWeaver–ABAP。 **<Collector 安装目录 >\etc\r3config.xsd** 中对该 xml 文件的布局、元素和属性进行了介绍。

要配置 SAP NetWeaver–ABAP 监控，请执行以下操作：

1 打开 **Collector\etc\r3config.xml**。

2 如果要定义通过 SAP 消息服务器访问 SAP NetWeaver–ABAP 实例的 SAP NetWeaver–ABAP 探测器实体，请找到以下列注释为前导内容的代码部分：

```
<!--
Template to be used with the message server connection option.
-->
```

如果要定义可直接访问 SAP NetWeaver–ABAP 实例的 SAP NetWeaver–ABAP 探测器实体，请找到以下列注释为前导内容的代码部分：

```
<!--
Template to be used with the direct connection option.
-->
```

3 复制注释以及注释下方的模板代码，然后将其粘贴到文件末尾。

4 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉原始的模板代码。

5 在文件末尾复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
r3system name	在 Diagnostics UI 中，在其下显示此 SAP NetWeaver–ABAP 探测器实体的探测器组的逻辑名称。	用户定义。
systemId	SAP NetWeaver–ABAP 系统的 ID。仅包含 3 个字符。	格式：[XXX] 可以从 SAP 系统管理员处获取。

属性	描述	值
client	SAP NetWeaver–ABAP 系统的客户端名称。	可以从 SAP 系统管理员处获取。
user	连接到 SAP NetWeaver–ABAP 系统的用户的名称。 此用户至少必须拥有 S_RFC 授权对象，才能查询对话信息。目标系统用户的授权配置文件中必须有此对象，才能使用 RFC 连接目标系统。 但是对于系统 R/3 4.7 及更早版本，仅拥有 S_RFC 授权对象是不够的。解决方法是，首先在与 ABAP 主机时间同步的计算机上安装 Collector，然后通过设置属性 timesynch.interval.secs = 0（在 Collector\etc\r3.properties 中），来禁用 Collector 中的时间同步功能。	可以从 SAP 系统管理员处获取。
password	连接到 SAP NetWeaver–ABAP 系统的用户的密码（纯文本）。	可以从 SAP 系统管理员处获取。
encrypted-password	连接到 SAP NetWeaver–ABAP 系统的用户的密码（加密）。	可使用 EncryptPassword.jsp 实用程序（请参阅 “密码模糊”（第 123 页））对密码进行加密。
messageServerHost （仅限消息服务器连接）	SAP 消息服务器主机的名称。	可以从 SAP 系统管理员处获取。

属性	描述	值
r3Name （仅限消息服务器连接）	仅包含 3 个字符。	格式：[XXX] 可以从 SAP 系统管理员处获取。
group （仅限消息服务器连接）	SAP 应用程序服务器组。	可以从 SAP 系统管理员处获取。
dialogInstance	指定要监控的对话实例的列表。 默认情况下将自动发现和监控 ABAP 系统（群集）的所有对话实例。 但是，如果对话实例过多（且过于繁忙），一个 Collector 无法处理（可能内存不足），您可以使用此属性以只监控部分对话实例，用其他 Collector 监控其余对话实例。	SAP 对话实例

针对 Oracle 进行配置

可以将 Collector 配置为收集要监控的每个活动 Oracle 系统实例的数据。可在 **<Collector 安装目录 >\etc\oracle-config.xml** 文件中配置 Oracle 监控。**<Collector 安装目录 >\etc\oracle-config.xsd.** 中对该 xml 文件的布局、元素和属性进行了介绍。

要配置 Oracle 监控，请执行以下操作：

1 打开 **<Collector 安装目录 >\etc\oracle-config.xml**。

2 复制包含在注释标记 （<!-- 和 -->）之间的模板代码，将其粘贴到文件末尾。

使用模板中的 **oracleInstance** 元素收集 Oracle 10g 和 11g 实例中的数据。如果要收集多个实例的数据，请添加 oracleInstance 元素的单独条目。

要从 Oracle RAC (Real Application Cluster) 进行收集，请指定 **oracleRac** 元素。在 **oracle-config.xml** 文件中，oracleRac 配置必须位于 oracleInstance 配置之后。

3 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
hostName	Oracle 数据库服务器主机的名称。必须使用完全限定主机名。 在 oracleRAC 配置中，此名称为群集别名。	可以从 Oracle 管理员处获取。
portNumber	Oracle 数据库服务器用于侦听请求的端口。	默认值：**1521**
instanceName	用于 oracleInstance 元素（不适用于 oracleRAC 元素）。在安装 Oracle 数据库服务器期间指定给 Oracle 实例的名称。	默认值：**Orcl** 可以从 Oracle 管理员处获取。

属性	描述	值
serviceName	用于 oracleRac 元素 （不适用于 oracleInstance 元素）。serviceName 与群集别名 hostName 可将客户端与 RAC 安装的更改隔离开来。	可以从 Oracle 管理员处获取。
userId	连接到 Oracle 数据库服务器的用户的 ID。 **注意：**用户至少需要具有 **CREATE SESSION** 和 **SELECT ANY DICTIONARY** 才能收集性能度量。	可以从 Oracle 管理员处获取。
password	连接到 Oracle 数据库服务器的用户的密码 （纯文本）。	可以从 Oracle 管理员处获取。
encrypted-password	连接到 Oracle 数据库服务器的用户的密码 （加密）。	可使用 EncryptPassword.jsp 实用程序 （请参阅 “密码模糊”（第 123 页））对密码进行加密。

属性	描述	值
probeName	用于 oracleInstance。用于在 Diagnostics UI 中表示此 Oracle 实例的逻辑名称。此名称必须唯一。 在 oracleRac 配置中，可能有多个探测器，因此不必输入 probeName。	用户定义。如果未定义此值，则会使用指定给 **instanceName** 的值。 在 Oracle RAC 配置中，每个 Oracle 实例的探测器名称将在运行时从 GV$INSTANCE 视图的 INSTANCE_NAME 列检索。
probeGroupName	在 Diagnostics UI 中，在其下显示此探测器的探测器组的逻辑名称。此探测器组可以是现有的探测器组，也可以是新定义的探测器组。 对于 oracleRac 元素而言为可选参数，如果忽略，则探测器组将设置为 serviceName。	由用户定义；例如： 现有：**Default** 新建：**Oracle**

要收集其他度量，请执行以下操作：

1. 如果 Collector 遇到未配置为要收集的度量，则会记录包含未识别度量 ID 和名称的警告。如果该度量是计数、百分比、字节或者是厘秒度量，则可以选择通过将度量 ID 添加到 **<Collector 安装目录 >\etc\oracle.properties** 来收集该度量。
2. 找到与希望 Collector 收集的度量类型相对应的属性名称，然后添加度量。属性名称为：
   - oracle.metrics.count
   - oracle.metrics.percent

- oracle.metrics.bytes
- oracle.metrics.centiseconds （Collector 转换成毫秒）

3 重新启动 Collector。

针对 SQL Server 的配置

可以将 Collector 配置为收集要监控的每个活动 SQL Server 系统实例的数据。在 **<Collector 安装目录 >\etc\sqlserver-config.xml** 文件中配置 SQL Server 监控。**<Collector 安装目录 >\etc\sqlserver-config.xsd.** 中对该 xml 文件的布局、元素和属性进行了介绍。

要配置 SQL Server 监控，请执行以下操作：

1 打开 **<Collector 安装目录 >\etc\sqlserver-config.xml**。

2 复制模板代码，并将其粘贴到文件的末尾。

3 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉模板代码。

4 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
hostName	SQL Server 数据库主机的名称。必须使用完全限定主机名。	可以从 SQL Server 管理员处获取。
portNumber	SQL Server 数据库用于侦听请求的端口号。	默认值：**1433**

属性	描述	值
instanceName	在安装 SQL Server 数据库期间指定给 SQL Server 实例的名称。 指定实例名称之后， Diagnostics 将自动搜寻该实例中的所有 SQL Server 数据库。要从集合中排除其中某些数据库（例如，系统数据库），请在 **<Collector 安装目录>\etc\sqlserver.properties** 文件的 **exclude.db.list** 属性中指定一个以逗号分隔的列表。	默认值：**默认值** 可以从 SQL Server 管理员处获取。

属性	描述	值
integratedSecurity	如果设置为 **true**，则不会指定用户名 / 密码。JDBC 驱动程序将在本地计算机凭据缓存中搜索已在登录计算机或网络时提供的凭据。 从 **HP Diagnostics Collector** 服务运行 Collector 时，必须将用于连接到 SQL Server 的 Windows 用户凭据设置为服务的 **logon** 属性。要执行此操作，请运行 Windows 服务管理器 （在运行对话框中运行 **services.msc**，或单击“我的电脑”>“管理”>“服务和应用程序”>“服务”）。打开 **HP Diagnostics Collector** 服务的“Properties”对话框，选择“Log On”选项卡，然后将“Log on as:”设置为对 SQL Server 实例具有访问权限的授权用户。该用户必须是域帐户。重新启动服务。 使用 Windows 身份验证时，需要使用域帐户（非本地帐户）才能连接到 SQL Server 实例。 如果设置为 **false**，则必须提供用户名和密码。如果不指定此值，则默认值为 False。	默认值：**false**

属性	描述	值
userId	连接到 SQL Server 数据库的用户的 ID。 **注意：**用户至少需要具有 **VIEW SERVER STATE** 才能收集性能度量数据。 可按以下方式创建具有 VIEW SERVER STATE 的用户： ➤ CREATE LOGIN diag WITH PASSWORD = '<pwd>'; ➤ USE master; ➤ GRANT VIEW SERVER STATE TO diag; ➤ GO 在 SQL Server Management Studio GUI 中右键单击实例名，然后选择属性。	可以从 SQL Server 管理员处获取。
password	连接到 SQL Server 数据库的用户的密码（纯文本）。	可以从 SQL Server 管理员处获取。
encrypted-password	连接到 SQL Server 数据库的用户的密码（加密）。	可使用 EncryptPassword.jsp 实用程序（请参阅“密码模糊”（第 123 页））对密码进行加密。

属性	描述	值
probeName	用于在 HP Diagnostics UI 中将此实例表示为探测器的名称。 如果实例中存在 n 个数据库，则您实际上将拥有 n+1 个探测器：额外的一个探测器用于其中包含等待事件等度量的实例总数。 该额外的探测器在 UI 中显示为 probeName。每个数据库的探测器将显示为 probeName_databaseName。	用户定义。 如果未定义此值，则会使用指定给 **instanceName** 的值。
probeGroupName	在 Diagnostics UI 中，在其下显示此探测器实体的探测器组的逻辑名称。 此探测器组可以是现有的探测器组，也可以是新定义的探测器组。	由用户定义；例如： 现有：**Default** 新建：**SQL Server**

针对 MQ 的配置

可以将 Collector 配置为收集要监控的每个活动 MQ 系统实例的数据。可在 **<Collector 安装目录>\etc\mq-config.xml** 文件中配置 MQ 监控。**<Collector 安装目录>\etc\mq-config.xsd.** 中对该 xml 文件的布局、元素和属性进行了介绍。

MQ 探测器需要以下权限：

setmqaut -m <queue_manager_name> -n ** -t queue -g <OS_group_name> +dsp +get

setmqaut -m <queue_manager_name> -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g <OS_group_name> +dsp +get +put

setmqaut -m <queue_manager_name> -n ** -t channel -g <OS_group_name> +dsp
setmqaut -m <queue_manager_name> -t qmgr -g <OS_group_name> +connect +dsp +inq

可以限制 MQ 探测器从其中收集度量数据的队列类型，以分离出最需要的应用程序度量数据。默认情况下， MQ 探测器仅会从预定义（或非动态）的队列收集度量。可以通过在 **<Collector 安装目录 >\etc\mq.properties** 文件中设置属性，来指定要收集或者要忽略的队列类型。

要限制从中收集度量的队列，请执行下列操作：

1. 打开 **<Collector 安装目录 >\etc\mq.properties** 文件。
2. 找到对应于不希望 Collector 从其收集度量数据的 MQ 队列定义类型的属性名称。下表中列出了属性名称及其对应的 MQ 队列定义类型。

属性	MQ 队列定义类型
collect.predefined.queues	MQQDT_PREDEFINED
collect.permanent.dynamic.queues	MQQDT_PERMANENT_DYNAMIC
collect.temporary.dynamic.queues	MQQDT_TEMPORARY_DYNAMIC
collect.shared.dynamic.queues	MQQDT_SHARED_DYNAMIC

3. **collect.predefined.queues** 属性默认设置为 true，其他三个属性默认设置为 false。对于不希望 Collector 从其收集度量数据的类型，指定 **false**，然后保存 **mq.properties** 文件。

注意：这些属性仅受 MQ 6.x 及更高版本支持。

MQ jar 文件是 Diagnostics Collector 随附的，但如果一定要覆盖这些文件也是可以的。Collector 附带提供的 MQ jar 文件位于 **<Collector 安装目录>\lib** 目录中，可以被本地 MQ 安装中提供的 MQ jar 文件覆盖。通常可以在本地 WebSphere 的 MQ 安装的 **\lib** 目录中找到 jar 文件，该目录还包含 **com.ibm.mq.jar** 文件。如果找不到这些文件，请与 WebSphere MQ 管理员联系。

要配置 MQ 监控，请执行以下操作：

1 打开 **<Collector 安装目录 >\etc\mq-config.xml**。

2 复制模板代码，并将其粘贴到文件的末尾。

3 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉模板代码。

4 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
hostName	主机名。	可以从 MQ 管理员处获取。
portNumber	端口的编号	（可选）。
queueManagerName	要连接的 MQ 管理器。	可以从 MQ 管理员处获取。
channelName	要通过其连接到队列管理器的通道。	可以从 MQ 管理员处获取。
securityExit	用于可插入安全提供程序的一个 IBM 术语 （提供 MQ 安全接口的一段代码）。 如果要使用它作为 MQ 网关，请指定完整的类名称作为参数，并确保安全 Exit 类在类路径中可用。	
probeName	用于在 HP Diagnostics UI 中将此实例表示为探测器的名称。 此名称必须唯一。	用户定义。如果未定义此值，则会使用队列管理器名称作为默认值。
probeGroupName	在 Diagnostics UI 中，在其下显示此探测器实体的探测器组的逻辑名称。此组可以是现有的探测器组，也可以是新定义的探测器组。	由用户定义；例如： 现有：**Default** 新建：**MQ**

针对 TIBCO EMS 的配置

可以将 Collector 配置为收集要监控的每个活动 TIBCO Enterprise Message Service (EMS) 系统实例的数据。

可在 **<Collector 安装目录>\etc\tibco-ems-config.xml** 文件中配置 TIBCO EMS 监控。 **<Collector 安装目录 >\etc\tibco-ems-config.xsd** 中对该 xml 文件的布局、元素和属性进行了介绍。

除了下述配置，还必须将以下 TIBCO EMS jar 文件从 TIBCO EMS 安装的 **Tibco_EMS>/ems/<version>/lib** 目录复制到安装了 Collector 的系统上的 **<Collector 安装目录 >\lib** 目录中：

- tibjms.jar
- tibjmsadmin.jar

要配置 TIBCO EMS 监控，请执行以下操作：

1. 打开 **<Collector 安装目录 >\etc\tibco-ems-config.xml**。
2. 复制模板代码，并将其粘贴到文件的末尾。
3. 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉模板代码。
4. 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
emsServerUrl	EMS 服务器 URL	默认为 tcp://localhost:7222
username	EMS 服务器用户名。用户必须具有以下权限：“view-destination”和 “view-server”	
password	EMS 服务器密码 （纯文本）	

属性	描述	值
obfuscated-password	EMS 服务器模糊密码（可选）。如果同时定义了两个密码，则此属性将首先采用纯文本密码。如果没有定义任何密码，则密码为空。	可使用 EncryptPassword.jsp 实用程序（请参阅“密码模糊”（第 123 页））对密码进行加密。
probeName	用于在 HP Diagnostics UI 中将此实例表示为探测器的名称。 此名称必须唯一。	用户定义。
probeGroupName	在 Diagnostics UI 中，在其下显示此探测器实体的探测器组的逻辑名称。 此探测器组可以是现有的探测器组，也可以是新定义的探测器组。	由用户定义，例如： 现有：**Default** 新建：**TIBCO**

通过在 **<Collector 安装目录 >\etc\tibco-ems.properties** 文件中设置属性，可以自定义 TIBCO 数据收集。

- 收集数据的频率
- 如果没有建立连接，则尝试重新连接的频率
- 启用或禁用服务器级、队列级和顶级度量收集
- 包括或排除全局、静态或临时队列和主题
- 选择单个度量

针对 webMethods 代理的配置

可以将 Collector 配置为收集要监控的 webMethods 代理系统的数据。

可在 **<Collector 安装目录 >\etc\wm-broker-config.xml** 文件中配置 webMethods 代理监控。**<Collector 安装目录 >\etc\wm-broker-config.xsd** 中对该 xml 文件的布局、元素和属性进行了介绍。

除了下述配置，还必须将以下 webMethods 代理 jar 文件从 webMethods 代理安装的 **<SoftwareAG>/common/lib** 目录复制到安装了 Collector 的系统上的 **<Collector 安装目录 >\lib** 目录中：

- wm-brokerclient.jar
- wm-g11nutils.jar

要配置 webMethods 代理监控，请执行以下操作：

1. 打开 **<Collector 安装目录 >\etc\wm-broker-config.xml**。
2. 复制模板代码，并将其粘贴到文件的末尾。
3. 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉模板代码。
4. 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
主机名	代理服务器主机名。	必需。例如：localhost。
brokerName	要连接的代理的名称。如果忽略，则连接到代理服务器中定义的默认代理。	可选

属性	描述	值
clientGroup	要使用的客户端组的名称。如果忽略，则连接到 “admin” 客户端组。	可选
probeGroupName	在 Diagnostics UI 中，在其下显示此探测器实体的探测器组的逻辑名称。 此探测器组可以是现有的探测器组，也可以是新定义的探测器组。	可选，默认为 “默认值”。 可以输入用户定义的名称。

通过在 **<Collector 安装目录 >\etc\wm-broker.properties** 文件中设置属性，可以自定义 webMethods 数据收集。

- 收集数据的频率
- 如果没有建立连接，则尝试重新连接的频率
- 启用或禁用服务器级、队列级度量收集
- 选择单个度量

针对 VMware 的配置

可以将 Collector 配置为收集每个要监控的 VMware 节点的数据。可在 **<Collector 安装目录 >\etc\vmware-config.xml** 文件中配置 VMware 监控。**<Collector 安装目录 >\etc\vmware-config.xsd** 中对该 xml 文件的布局、元素和属性进行了介绍。对 **vmware-config.xml** 文件的更改将动态获取。

Collector 要求在 vCenter Server 上安装修补程序：（有关详细信息，请参见 <http://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKC&docType=kc&externalId=1024596&sliceId=1&docTypeID=DT_KB_1_1&dialogID=139216791&stateId=1 0 139218894>）。

应当在 VMware 访客上安装最新的 VMware Tools。可以使用 vSphere 客户端安装这些工具。您需要在 Diagnostics UI 中从 VMware 访客向下搜索至主机才能找到最新工具，因为 VMware 工具是通过 vCenter 使 VMware Collector 能够使用访客的 FQND 的。

在启动 Collector 时将创建 VMware 主机 / 访客关联。新 VMware 主机和 VMware 访客可能需要 15 分钟的时间才能在 Diagnostics 中显示出来。删除的或迁移的 VMware 访客可能需要 5 分钟的时间才能在 Diagnostics 中显示出来。

要配置 VMware 监控，请执行以下操作：

1. 打开 **<Collector 安装目录 >\etc\vmware-config.xml**。
2. 复制模板代码，并将其粘贴到文件的末尾。
3. 通过在模板代码上方的空行中键入 **<!--**，并在后面的空行中键入 **-->**，来注释掉模板代码。
4. 在所复制的代码中，按下表所述更改每个属性的值，然后保存文件。

属性	描述	值
serverURL	用于通过 VMware 基础结构 vSphere Web 服务 API 连接到 VMware ESX 或 vCenter Server 的 URL。 例如： https://<myVM.myCo.com>/sdk	
userId	VMware ESX 或 vCenter 用户 ID。 至少用户必须具有只读访问权且位于用户组中。	

属性	描述	值
encrypted-password	对应于用户 ID 的加密 VMware 密码。首先检查加密密码，如果密码不为空则使用它；否则使用纯文本密码。如果纯文本密码不存在或为空，则将密码留空。	可选
password	对应于用户 ID 的纯文本 VMware 密码。	可选

通过在 **<Collector 安装目录>\etc\vmware.properties** 文件中设置属性，可以自定义 VMware 数据收集。

- 可以限制查询间隔和重新连接时间。查询间隔只是对 Collector 的提示，因为实际采样间隔必须是 VMware Server 上配置的间隔的倍数。
- 您还可以按 VMware 主机 (ESX Server) 和 VMware 访客 （虚拟机）进行筛选。如果 VMware Collector 无法处理整个 vCenter 的负载，则主机和访客筛选器可能允许您在最重要的 vCenter 部分上使用 VMware Collector ，或在多个 VMware Collector 之间对 vCenter 进行分区。

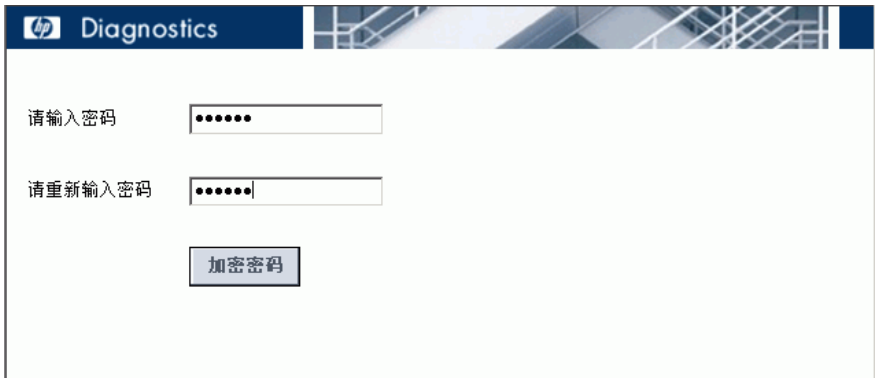
密码模糊

可使用 Diagnostics 中包含的 Web 应用程序创建模糊密码。访问 “安全”页面 (http://< 主机名 >:2006/security) 并选择页面底部的 “加密密码”。需要使用其中安装了 Diagnostics 的计算机的名称替换 < 主机名 >。

生成的模糊密码可用于以下不同收集类型的 xml 文件：

- **r3config.xml** 文件，用于配置 SAP NetWeaver–ABAP Collector

- **oracle-config.xml** 文件，用于配置 Oracle Collector
- **vmware-config.xml** 文件，用于配置 VMware Collector
- **tibco-ems-config.xml** 文件，用于配置 TIBCO EMS Collector
- **sqlserver-config.xml** 文件，用于配置 SQL Server Collector。



请输入纯文本密码，然后重新输入密码进行确认，再选择“加密密码”按钮。此时，将显示模糊的密码。复制此页面中的完整模糊密码（包括开头的“OBF:”），然后，将该密码粘贴到相应的属性文件（**r3config.xml**、**oracle-config.xml**、**vmware-config.xml**、**tibco-ems-config.xml** 或 **sqlserver-config.xml**）中。

注意：您可以继续使用纯文本密码属性。

---

security.encrypted-password 属性也可以用于下列属性文件中的 mercury 用户密码：**collector.properties**、**dispatcher.properties**、**server.properties**。*mercury* 用户用于进行各种 Diagnostics 组件之间的身份验证。以下是这些属性文件中受影响部分的副本：

```
##############################################################################
# Remote Server Authentication Properties
##############################################################################

#
# This user name and password is used for communication between Diagnostics
# components (probes, and servers). You may want to change this password
# every so often to keep your system secure inside your enterprise. If you
# do change this password, you must first use
# http://<host name>:2006/security and select Encrypt Password to encrypt the
# password.
# Plaintext passwords can be used by replacing the security.encrypted-password
# with security.password. You must also change the encrypted password in the
# <install-dir>/etc/.htaccess file, as well as all the Diagnostics probe, and
# servers, that communicate with each other in your enterprise.
#
security.username=mercury
security.encrypted-password=OBF:1c431jg81hv41k1d1l161wu81z0d1pyl1wmt1n6h1y
m71n511wnd1pw11z0h1wu61kxw1jyl1hse1jd21c2z
```

验证 Diagnostics Collector 安装

安装完成后，Diagnostics Collector 会自动运行。通过检查 **collector.log** 文件中的错误，可以验证 Collector 安装。

在启动 Collector 探测器实例之后，则可以启动 Diagnostics Enterprise UI 以验证该探测器是否正在工作。转至 **http://<Diagnostics Commander 服务器 >:2006/**。此时，可使用默认的用户 / 密码即“admin / admin”，或给定的登录名（如果已为您设置此登录名）

此外，还可检查“系统运行状况”视图，查找有关 Collector 部署及托管该 Collector 的计算机的信息。

要访问系统视图，请执行以下操作：

1 以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。

2 在查询页面的列表中查找 Mercury 系统用户，并选择此链接打开 Diagnostics。

3 登录 Diagnostics 并在 “应用程序”窗口中选择 “整个企业”，然后选择任何链接打开 Diagnostics 视图。

4 在 “视图”窗格中，将显示 “系统视图”视图组。打开视图组，并选择 “系统运行状况”视图或 “系统容量”视图。

启动和停止 Diagnostics Collector

适用于 Windows 计算机的说明

要在 Windows 计算机上启动 Collector，请执行以下操作：

➤ 选择 “开始” > “所有程序” > “HP Diagnostics Collector” > “Start HP Diagnostics Collector”。或者，在命令行中输入 **net start "HP Diagnostics Collector"**。

要在 Windows 计算机上停止 Collector，请执行以下操作：

➤ 选择 “开始” > “所有程序” > “HP Diagnostics Collector” > “Stop HP Diagnostics Collector”。或者，在命令行中输入 **net stop "HP Diagnostics Collector"**。

适用于 UNIX 计算机的说明 （使用 Nanny）

nanny 是一个作为守护程序运行的进程，可以确保 Collector 始终处于运行状态。下面是使用 nanny 启动和停止 Collector 的过程。

要在 UNIX 计算机上启动 Collector，请执行以下操作：

1 确保 **M_LROOT** 环境变量被定义为 Collector 的根目录。例如，可以在 ksh 中输入以下内容：

```
export M_LROOT=<collector_install_dir>/nanny/solaris
```

如果 **M_LROOT** 环境变量未定义为根目录，则会显示以下错误：

```
Warning: MDRV: cannot find lrun root directory. Please check your M_LROOT
Unable to format message id [-10791]
m_agent_daemon (is down)
```

2 将目录更改为 **$M_LROOT/bin**。

3 使用 **-install** 选项运行 **m_daemon_setup**，如以下示例所示：

```
cd $M_LROOT/bin
./m_daemon_setup -install
```

要在 UNIX 计算机上停止 Collector，请执行以下操作：

1 将目录更改为 **$M_LROOT/bin**，如上面的启动过程设置所述。

2 使用 **-remove** 选项运行 **m_daemon_setup**，如以下示例所示：

```
cd $M_LROOT/bin
./m_daemon_setup -remove
```

适用于 UNIX 计算机的说明 （不使用 Nanny）

下面是未使用 nanny 启动和停止 Collector 的过程。

要在 UNIX 计算机上启动 Collector，请执行以下操作：

➤ 运行 **<Collector 安装目录 >/bin/collector.sh**。

要在 UNIX 计算机上停止 Collector，请执行以下操作：

➤ 使用 **kill** 等实用程序终止进程。

确定 Diagnostics Collector 的版本

在请求支持时，如果知道 Diagnostics Collector 的版本将十分有用。可以在 **<Collector 安装目录 >\version.txt** 文件中找到 Collector 版本号。

卸载 Diagnostics Collector

要卸载 Collector，请执行以下操作：

➤ 在 Windows 计算机上，选择 “开始” > “所有程序” > “HP Diagnostics Collector” > “Uninstall Diagnostics Collector”。

或者，可运行 **<Collector 安装目录 >_uninst** 目录中的 **uninstaller.exe**。

➤ 在 Linux 或 Solaris UNIX 计算机上，运行 **<Collector 安装目录 >/_uninst** 目录中的 **uninstall***。

➤ 在其他 UNIX 计算机上，选择 1.5 或更高版本的 JVM，并运行 **Java -jar <Collector 安装目录 >/_uninst/uninstall.jar**，以卸载 Collector。

第 III 部分

Java 代理和 .NET 代理的安装和设置

本部分包括：

- 安装 Java 代理
- 准备应用程序服务器以使用 Java 代理进行监控
- 准备应用程序服务器以使用 Java 代理进行客户端监控
- 安装 .NET 代理

5

安装 Java 代理

本节描述如何安装 Java 代理，以及如何设置和配置 Java 代理。

本章包括：

Java 代理安装概述

HP Diagnostics/TransactionVision Java 代理安装程序用于安装 Java 代理，以收集 Diagnostics 或 TransactionVision 或同时收集这两者的数据； TransactionVision 可向 Business Service Management 中的事务管理应用程序提供数据。有关安装 Java Agent for TransactionVision 的详细信息，请参阅 Business Service Management 文档库中的 《TransactionVision Deployment Guide》。

此代理安装在您要监控的应用程序所在的计算机上。

在使用 Java 代理监控 HP Diagnostics 中的应用程序之前，您必须：

- 安装 Java 代理。
- 运行 Java 代理安装模块，该模块在安装程序完成后会自动启动。
- 后续步骤将检测应用程序服务器使用的 JRE，并配置应用程序服务器的 JVM 参数以调用 Java 代理。根据您的应用程序服务器，您可以使用 JRE 自动检测选项，而不是手动运行 JRE Intrumenter 实用程序。

  临时目录的可用空间约为 400 MB。有关建议的用于承载 Java 代理的系统配置信息，请参阅“Diagnostics Java 代理主机的要求”（第 34 页）。我们为 Windows 平台以及多个 UNIX 平台提供了 Java 代理安装程序。可以在图形模式下运行安装程序 （显示如同 Windows 安装程序一样的安装屏幕），或使用控制台模式命令行界面。如果无法使用正规 UNIX 安装程序，则可以如 “使用常规安装程序安装 Java 代理”（第 154 页）中所述使用常规安装程序。

重要信息：默认情况下，**<探测器安装目录>/log** 目录设置为 777。这可确保 Java 代理从任意用户运行的受监控应用程序中收集度量。
根据组织的安全要求，您可以进一步限制对此目录的访问；例如：
chmod 775 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/log

---

注意：如果在主机上预先安装了 Java 代理，请参阅 “升级和修补程序安装说明”（第 851 页），了解有关如何升级代理系统的重要说明。

HP 软件即服务 (SaaS)。可将 HP Diagnostics 部署到 HP 软件即服务 (SaaS) 环境中。在 SaaS 部署中，Diagnostics Java 代理安装在您公司的 IT 环境中，Diagnostics Commander 服务器和 Mediator 服务器由 HP 安装在 HP 自己的 SaaS 系统上。在安装 Java 代理期间，可以选择针对具有 SaaS 托管的调解器（安装在 HP 自有系统上）的 Diagnostics 配置代理。

请参阅访问 Java 代理安装程序开始安装。

访问 Java 代理安装程序

您可以从 Diagnostics 安装光盘安装 Java 代理，或者将可执行安装文件复制到其他位置并运行该文件，也可以选择从 Business Service Management 中的“Diagnostics 下载”页面安装 Java 代理。

在安装 Profiler 试用软件时，从 HP 软件网站启动安装程序。

要从 Diagnostics 安装位置访问安装程序，请执行以下操作：

➤ 对于 Windows，Diagnostics 安装 DVD (Autorun.exe) 将显示安装菜单页面。在菜单中选择 “Diagnostics Agent for Java”，启动安装程序。

OR

➤ 可以直接运行适当的安装程序，方法是在安装位置查找 **HPDiagTVJavaAgt_<版本号>_< 平台 >.bin** 文件（用于 Unix）或 **HPDiagTVJavaAgt_< 版本号 >_win.exe** 文件 （用于 Windows），将文件复制到新的安装位置并双击 .exe 文件，或运行 .bin 安装程序。

按照 “安装 Java 代理”（第 135 页）中的说明继续操作。

要从 HP 软件下载中心下载安装程序，请执行以下操作：

1 转至 HP 软件网站的软件下载中心。

2 查找 **Diagnostics**（或 TransactionVision）下载信息，并选择下载 Diagnostics Agent 软件的相应链接。请注意，还可以通过下载中心获得 Diagnostics Profiler 试用 / 评估版软件。

3 按照网站上的下载说明进行操作。

按照 “安装 Java 代理”（第 135 页）中的说明继续操作。

要从 Business Service Management 的 Diagnostics “下载”页面下载安装程序，请执行以下操作：

1 在 Business Service Management 中，从主菜单中选择“管理”>“Diagnostics”，然后单击 “下载”选项卡。

2 在 “下载”页面上单击链接，下载相应的 Java 代理安装程序。

注意：只有将 Java 代理安装程序置于 Business Service Management 能够访问的目录下时，才能在 Business Service Management 中使用此程序。您可以在 Diagnostic 服务器安装期间启用此程序，或者将 Java 代理安装程序从安装光盘手动复制到所需位置。

继续进行安装 Java 代理。

安装 Java 代理

本节提供首次在 Windows 或 UNIX 系统中安装 Java 代理的详细说明。

重要信息：如果主机上存在预安装的 Java 代理，则必须按照说明升级此代理系统，而不遵循这些安装说明，请参阅“升级和修补程序安装说明”（第 851 页）。

有关其他安装类型的信息，请参阅：

- 对于 z/OS，参阅 “在 z/OS 大型计算机上安装 Java 代理”（第 152 页）。
- 有关使用常规安装程序进行安装的详细信息，请参阅 “使用常规安装程序安装 Java 代理”（第 154 页）。

- 有关静默安装的信息，请参阅 “静默安装 Java 代理”（第 155 页）。
- 如有需要，可更改 UNIX 安装程序文件的模式使其可执行。有关 UNIX 命令的详细信息，请参阅 “使用 UNIX 命令”（第 881 页）。
- 要以控制台模式运行安装程序，请在命令提示符处输入以下内容：

```
./<installer> -console
```

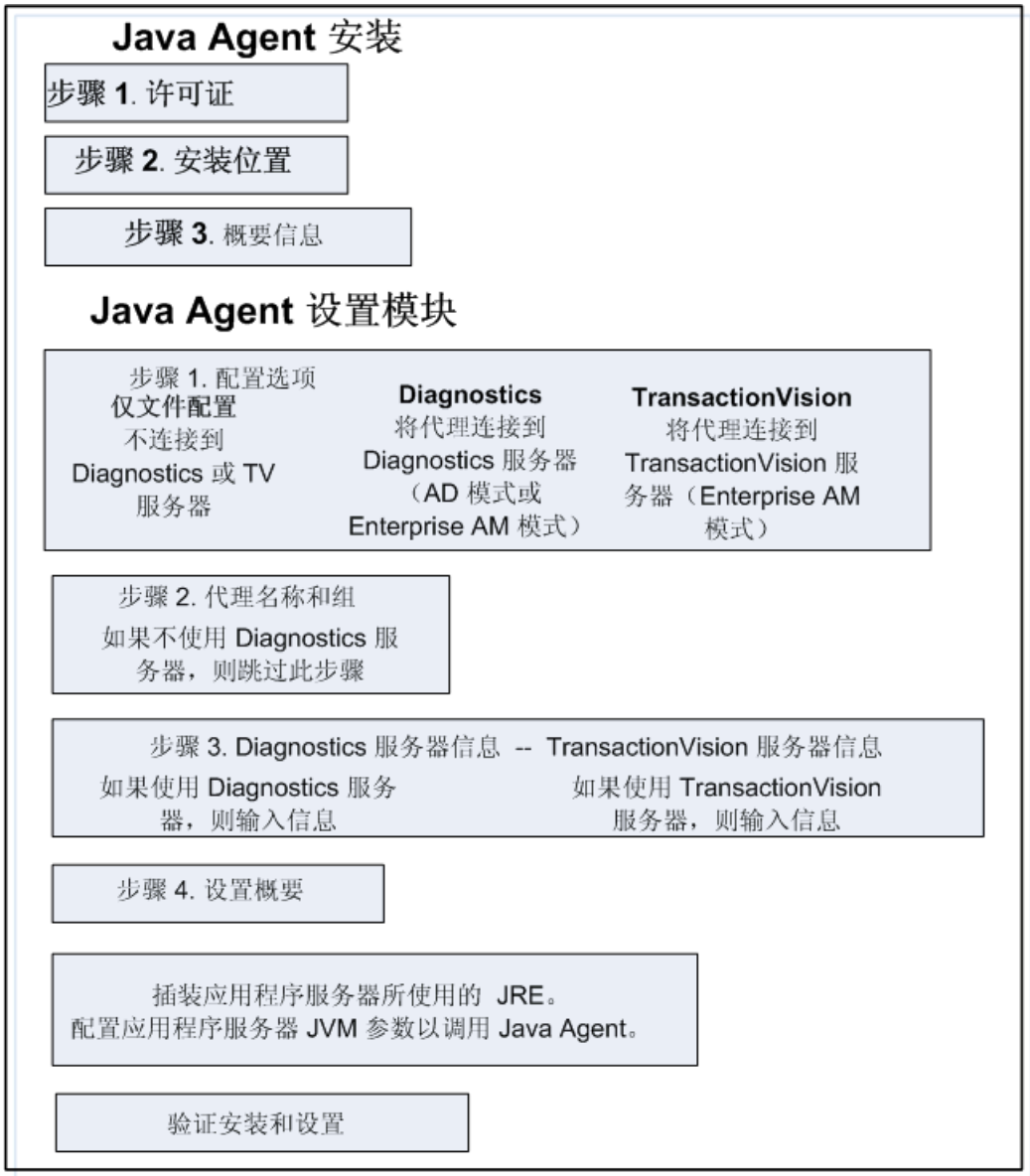
安装程序在控制台模式下将显示安装提示。

- 要以图形模式运行安装程序，请在命令提示符处输入以下内容：

```
./<installer>
```

安装程序所显示的屏幕与 Windows 安装程序所显示的屏幕相同。

下图概述了 Java 代理的安装步骤；有关每个步骤的详细信息，请参阅本节其余内容。



从 “步骤 1. 最终用户许可证协议”（第 138 页）开始安装 Java 代理。

步骤 1. 最终用户许可证协议

接受最终用户许可证协议。

阅读协议并选择 “我接受许可证协议条款”。

在控制台模式界面中，按 **Enter** 键可移动到文本下一页（而非选择“下一步”），或键入 q 跳到许可证协议的结尾处。

选择 “下一步”（在控制台模式中选择 **Enter**）可继续下一个步骤。

步骤 2. 指定安装位置

指定代理的安装位置。

您可以接受默认安装目录，也可以指定其他位置，方法是在 “安装目录名”框中键入安装目录的路径，或单击 “浏览”导航到安装目录。

在控制台模式界面中，在 “安装目录名”提示符处，您可以接受方括号中显示的默认安装位置，也可以输入其他位置的路径。

注意: 此位置将变为 **< 探测器安装目录 >**。默认情况下，此位置是 Windows 上的 **C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent** 和 UNIX 上的 **/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent**。

如果发生错误，请检查 JavaAgent 目录是否已存在，并将其删除。如果安装和卸载了之前的 Java 代理，但没有同时删除 JavaAgent 子目录，则可能会出现此问题。

选择 “下一步”（在控制台模式中选择 **Enter**）可继续下一个步骤。

步骤 3. 检查预安装概要信息

检查安装概要信息。

安装目录和大小要求均已列出。

如果满足列出的条件，则选择“下一步”（在控制台模式中选择 **Enter**）开始安装。安装过程会持续几分钟。

安装完成后将启动 Java 代理安装模块。继续运行 Java 代理安装模块的下一部分。

运行 Java 代理安装模块

可以在未连接 Diagnostics 服务器的情况下将 Java 代理配置为 Profiler（或者配置为与 Diagnostics 服务器和 / 或 TransactionVision 服务器配合使用的代理）。如果初始情况下仅将代理配置为 Profiler，则可稍后通过重新运行 Java 代理安装模块将代理配置为与 Diagnostics 服务器配合使用。

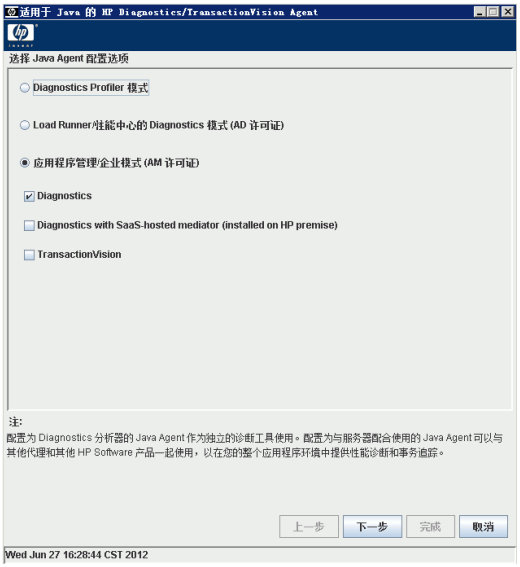
通过使用 Java 代理安装模块配置 Java 代理，该模块会在 Java 代理安装结束时自动启动，您也可以选择“开始”>“所有程序”>“HP Java Agent”>“Setup Module”随时启动它。对于 UNIX，可通过运行 **< 探测器安装目录 >/bin/setupModule.sh** 随时启动它。

Java 代理安装模块包括以下步骤，选择步骤 1. 配置选项以开始：

- “步骤 1. 配置选项”（第 140 页）
- “步骤 2. 代理名称和组”（第 142 页）
- “步骤 3. Diagnostics 服务器信息”（第 143 页）
- “步骤 4. 安装后概要”（第 147 页）

步骤 1. 配置选项

指定是否将 Java 代理作为独立 Profiler 安装且不建立与服务器的任何连接（例如，如果安装的是 Diagnostics Java Profiler 试用版软件，或安装与 LoadRunner/ Performance Center 配合使用的代理，或与 Diagnostics 和/或 TransactionVision 服务器配合使用的代理）。



选择适用于代理使用环境的方式。

- **应用程序管理 / 企业模式（AM 许可证）**。选择此选项将安装在企业（或生产）环境中与 Diagnostics 服务器和 / 或 TransactionVision 服务器配合使用的代理。

  然后指定以下哪些服务器需要配置代理：

  - Diagnostics 服务器（本地安装）或在 HP SaaS 系统（在 HP Premise 上）中托管的 Diagnostics 服务器
  - TransactionVision 服务器
  - 本地安装的 Diagnostics 服务器和 TransactionVision 服务器

  如果选择 HP SaaS 模式，则 HP SaaS 管理员将为您提供将 Java 代理连接到 HP SaaS 托管的 Diagnostics Mediator 服务器的信息。

  如果选择 TransactionVision，则参阅 Business Service Management 文档库中的《HP TransactionVision Deployment Guide》，获取有关特定于 TransactionVision 的设置选项的详细信息。

通过应用程序管理 / 企业模式 （AM 许可证）选项，如果选择 Diagnostics 服务器，则 **etc/probe.properties** 文件中 **active.properties property** 的值将设置为 **Enterprise** 模式。如果在安装 Java 代理时选择 TransactionVision 服务器，则会设置为 **TV** 模式 （请参阅 “设置活动产品模式”（第 461 页））。

对于那些设置为企业模式的代理，将针对 HP Diagnostics AM 许可证容量计算代理。

➤ 选择 “Diagnostics Profiler 模式”以将代理配置为 Diagnostics Java Profiler 且不连接 Diagnostics 服务器。在购买 HP Diagnostics 产品之前安装 Diagnostics Java Profiler 试用版软件时，通常会使用 Diagnostics Profiler 模式。

如果选择 Diagnostics Profiler 模式，在安装 Java 代理 时，**etc/probe.properties** 文件中 **active.products** 属性的值将设置为 **PRO** 模式 （请参阅 “设置活动产品模式”（第 461 页））。

如果选择 Diagnostics Profiler 模式，但没有其他配置选项，则可以选择 “完成”以完成配置。此时将显示 “安装后概要”对话框。

➤ **LoadRunner/ 性能中心的 Diagnostics 模式（AD 许可证）。**选择此选项以安装在负载测试 （或预生产）环境中与 Diagnostics 服务器配合使用的代理，在这种环境下只会在 LoadRunner 或 Performance Center 运行时使用探测器。

代理将在 AD 许可证模式下安装，这意味着当代理在 LoadRunner 或 Performance Center 测试运行中时，只会针对 HP Diagnostics AD 许可证容量计算代理。有关 AD 许可证容量的详细信息，请参阅“基于当前连接的探测器许可证信息”（第 83 页）。

在 AD 模式下，代理仅在 LoadRunner 或 Performance Center 运行时捕获数据，并将结果存储在用于该运行的特定 Diagnostics 数据库中，例如 Default Client:21。当代理处于 AD 模式下时，除非探测器参与了 LoadRunner/Performance Center 运行，否则代理不会将任何数据发送到服务器。

如果选择此 AD 许可证选项，在安装 Java 代理时，**etc/probe.properties** 文件中 **active.properties property** 属性的值将设置为 **AD** 模式 （请参阅 “设置活动产品模式”（第 461 页））。

在 AD 模式内运行探测器的好处在于，只有当探测器在 LoadRunner 或 Performance Center 测试运行中时，才针对许可证容量计算 AD 模式下的探测器。例如，如果在 LoadRunner/Performance Center AD 模式下安装了 20 个探测器，但每次测试运行中只会有 5 个探测器，那么就只需要可容纳 5 个探测器的 AD 许可证容量。

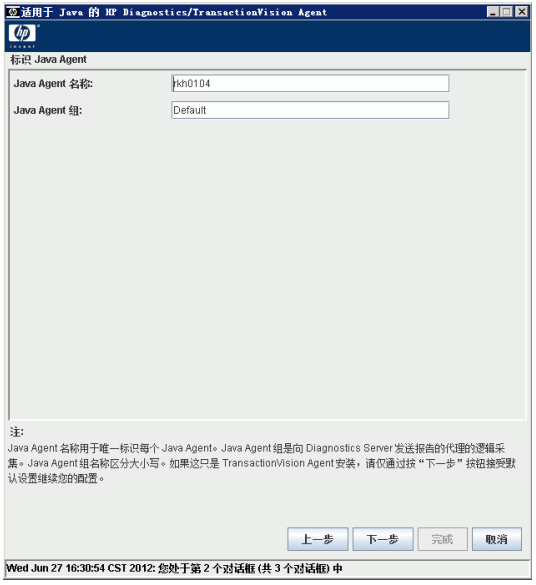
在控制台模式界面中，输入 X 选择安装模式。

选择 “下一步”（在控制台模式中选择 **Enter**）可继续下一个步骤。

步骤 2. 代理名称和组

如果代理不向 Diagnostics 服务器进行汇报，则可跳过此步骤。

指定 Java 代理的名称和所属组。



➤ 对于 Java 代理名称，请输入在 HP Diagnostics 中唯一标识代理的名称。可以在名称中使用 -、 _ 及所有字母数字字符。代理名称将指定为默认探测器实体名称。如果您在系统上安装了一个代理，而又打算监控多个应用程序服务器或应用程序域，则可以稍后为每个受监控应用程序配置唯一的探测器名称。

向代理分配名称时，请选择可帮助您识别要监控的应用程序的名称以及要安装代理的系统（例如，如果在带有 WebLogic 应用程序服务器的系统 ovrserver130 上安装，则可以使用代理名称 WL10_MedRec_ovrserver130）。

➤ 对于 Java 代理组名称，请输入现有组或待建新组的名称。代理组名称区分大小写，并将用作探测器组名称。

探测器组是向相同的 Diagnostics 服务器进行报告的探测器的逻辑分组。探测器组的性能度量是可以跟踪的，并且可在多个 Diagnostics 视图上显示。

例如，可以将一个特定企业应用程序的所有探测器分配到一个探测器组中，这样既可以监控组级别的性能，又可以监控基于各个探测器实体的性能。

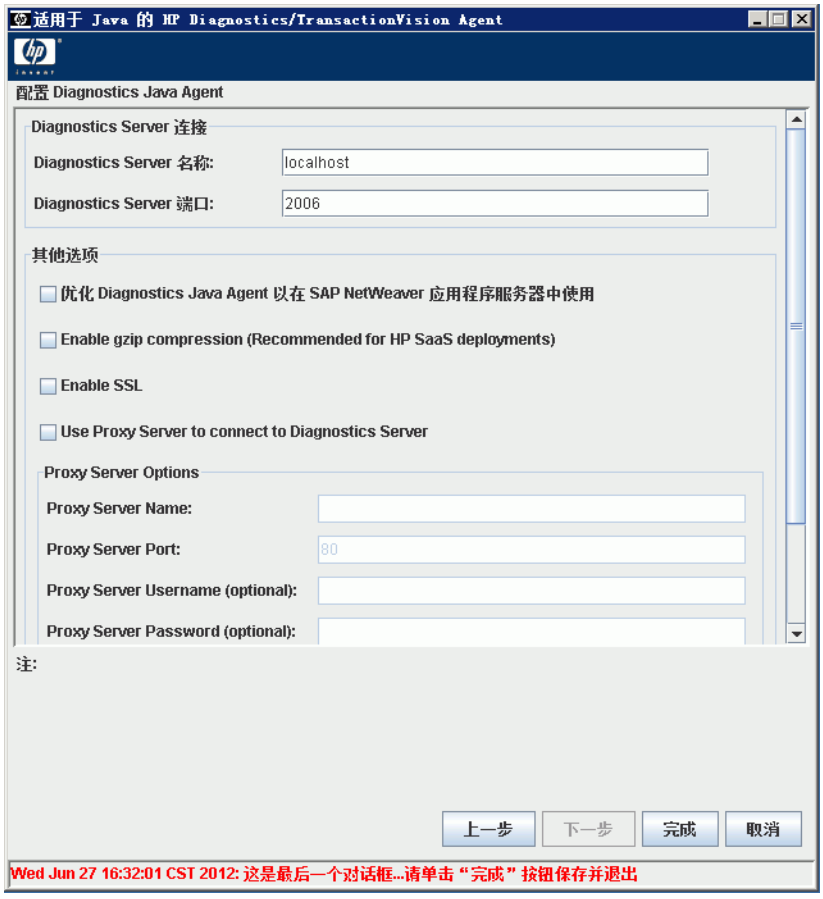
选择 “下一步”（在控制台模式中选择 **Enter**）可继续下一个步骤。

步骤 3. Diagnostics 服务器信息

如果代理不向 Diagnostics 服务器进行汇报，则可跳过此步骤。

输入 Diagnostics 服务器和其他选项的配置信息。

在控制台模式界面中，为每个选项输入 **X** 表示 “是”，输入 **O** 表示 “否”。



在 “Diagnostics Server 名称” 框中，输入应连接此代理的 Diagnostics 服务器主机的主机名或 IP 地址。应指定完全限定主机名而不只是简单的主机名。在包含 UNIX 的混合操作系统环境中，必须指定完全限定主机名，才能保证网络路由正常工作。

Commander 服务器。如果在要运行代理的 Diagnostics 部署中只有一个 Diagnostics 服务器，请在此处输入 Diagnostics Commander 服务器主机名和端口信息。

Mediator 服务器。在包含一个 Commander 服务器和多个 Mediator 服务器的分布式环境中，输入从代理接收数据的 Diagnostics Mediator 服务器的信息。

如果使用 “HP 软件即服务 (SaaS)”，则 HP 会将 Diagnostics Mediator 安装在 HP 自有的 HP SaaS 系统中。 HP SaaS 管理员将为您提供有关要使用的主机名和端口的信息。另请注意，在 HP SaaS 环境中，“Enable gzip”选项将自动选中，您将看不到 “Enable SSL”选项，因为该选项在 HP 自有系统的 Diagnostics Commander/Mediator 上已配置。

➤ 在 “Diagnostics Server 端口”框中，输入 Diagnostics 服务器的端口号。

Diagnostics 服务器的默认端口是 **2006**。对于与服务器的 SSL 通信，本地安装的服务器的端口通常设置为 **8443**。

如果在 SaaS 环境中安装代理，则默认端口为 **443** （SaaS 管理员将为您提供详细信息）。

如果在 Diagnostics 服务器安装后更改了端口，则请在此处指定新端口号，以代替默认端口号。

➤ 要允许此代理支持 SAP NetWeaver 应用程序服务器，请选中“优化 Diagnostics Java 代理以在 SAP NetWeaver 应用程序服务器中使用”复选框。

➤ 如果需要压缩 Java 代理和调解器之间的数据，请设置“Enable gzip compression”复选框。这是对带宽和探测器性能开销之间的权衡。在 HP SaaS 环境中，通常要求您启用 gzip 压缩。有关详细信息，请咨询 SaaS 管理员。

- 当选中“Enable SSL”或当 Diagnostics 服务器为 HP 自有托管 SaaS 时，Java 代理通过 SSL 连接到 Diagnostics 服务器。选中 “Enable SSL”复选框将指示代理在 SSL 模式下连接到 Diagnostics 服务器，并尝试从服务器下载所需的证书链。因此， **server.properties** 信任证书将包含该证书。有关安全通信的详细信息，请参阅 “在组件之间启用 HTTPS”（第 797 页）。
- 如果代理服务器用于与 Diagnostics Mediator 服务器通信，请选中 “Use Proxy Server to connect to Diagnostics Server”复选框并输入相应的选项。在 HP SaaS 环境中，如果您的公司需要代理以与外部服务器通信，则您可以选中此选项。您也可以在代理系统上的 **dispatcher.properties** 文件中设置这些选项，方法是将 proxy.enabled 设置为 true 并输入其他选项。请参阅 “针对 HTTP 代理配置 Diagnostics 服务器和代理”（第 627 页）。

  Proxy Server Options：

  - **Proxy Server Name。**代理服务器的主机名。
  - **Proxy Server Port。**代理服务器的端口。
  - **Proxy Server Username (optional)。**用于对代理服务器进行身份验证的用户。
  - **Proxy Server Password (optional)。**用于对代理服务器进行身份验证的密码。
- 建议您更改默认的 “本地 Profiler 密码” (admin)。

TransactionVision 信息

如果选择为 TransactionVision 配置此代理，则将看到用于为 TransactionVision 配置代理的其他对话框。有关这些安装选项的详细信息，请参阅《HP TransactionVision Deployment Guide》。

安装过程开始

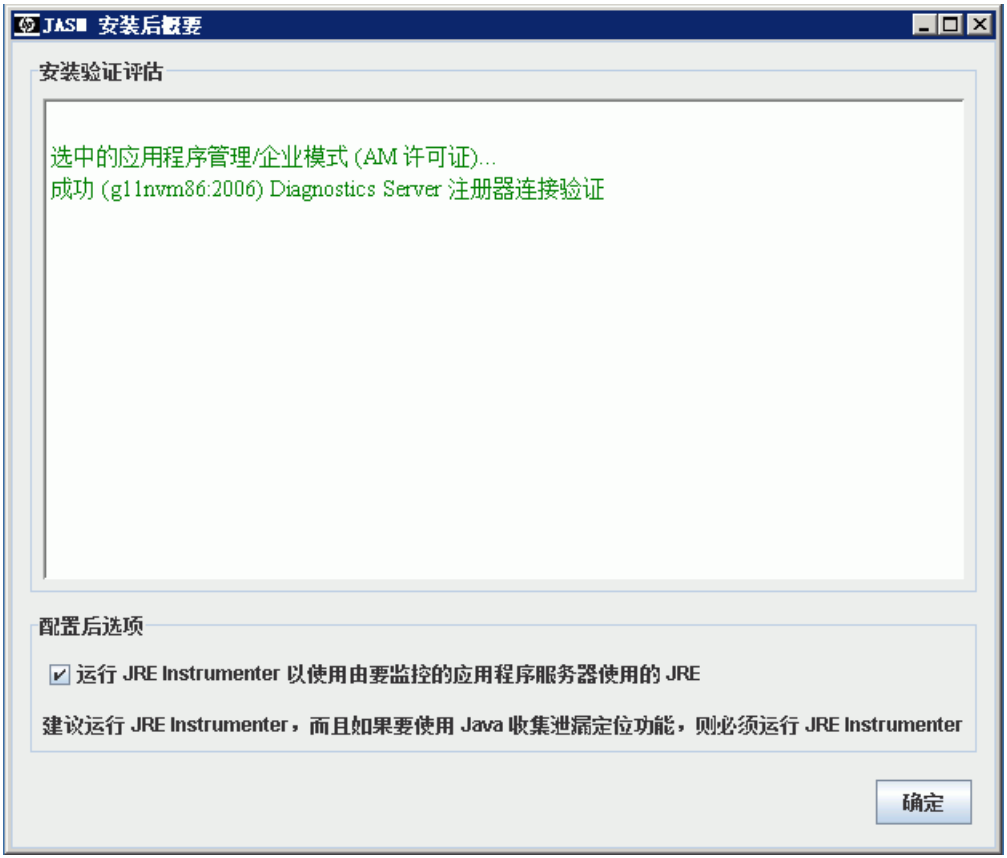
Java 代理安装过程开始。在图形模式下，进度条表示配置的实际进度。

测试 Diagnostics 服务器的连接性。如果遇到连接问题，“设置程序”会显示连接性检查的结果。

继续执行下一步。

步骤 4. 安装后概要

检查 “安装后概要”并单击 “确定”。



您应根据自己的应用程序服务器使用 JRE 自动插桩选项，而不是手动运行 JRE Intrumenter 实用程序。因此，用于运行 JRE Instrumenter 实用程序的复选框默认留空。请参阅下面的有关准备应用程序服务器以进行监控以继续。

有关准备应用程序服务器以进行监控

后续步骤将插桩应用程序服务器使用的 JRE，并配置应用程序服务器的 JVM 参数以调用 Java 代理。

有关如何插桩应用程序服务器使用的 JRE，以及如何配置用于特定应用程序服务器的 JVM 参数以调用 Java 代理的信息，请遵循“准备应用程序服务器以使用 Java 代理进行监控”（第 161 页）中的说明。

在准备好应用程序服务器以供 Java 代理进行监控后，您需要重新启动应用程序服务器，然后将调用 Java 代理以开始监控应用程序。

有关客户端监控功能的详细信息，请参阅第 7 章，“准备应用程序服务器以使用 Java 代理进行客户端监控”。

使用 Diagnostics 服务器注册代理

配置代理以连接到 Diagnostics Commander 服务器。

Diagnostics Commander 服务器的功能之一是跟踪 Diagnostics 组件，以帮助执行这些组件之间的通信，并让您随时了解这些组件的状态和运行状况。

要配置代理以向 Diagnostics 服务器注册，请使用 **registrar.url** 属性来设置主机的名称和端口，您可在以下属性文件中找到该属性：**< 探测器安装目录 >\etc\dispatcher.properties**。

下面是 **dispatcher.properties** 中的一段摘录，显示了 **registrar.url** 属性。

```
## the URL of the registrar
registrar.url=http://host01.company.com:2006/registrar/
```

验证 Java 代理安装

在启动探测器之后，代理才会使用 Diagnostics 服务器进行注册。已插桩的应用程序服务器启动时，探测器也会一同启动。因此，在检测应用程序服务器使用的 JRE 并配置应用程序服务器 JVM 参数以调用 Java 代理之前，您无法检查代理是否正常工作。

在启动 Java 探测器实例之后，便可以启动 Diagnostics Enterprise UI 以验证该探测器是否正在工作。转至 **http://<Diagnostics Commander 服务器 >:2006/**。此时，可使用默认的用户名 / 密码 **admin/admin**，或给定的登录名（如果已为您设置了不同登录名）。

此外，还可检查“系统运行状况”视图，查找有关 Java 代理部署及托管代理的计算机的信息。

要访问系统视图，请执行以下操作：

1. 以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。
2. 在查询页面的列表中查找 Mercury 系统用户，并选择此链接打开 Diagnostics。
3. 登录 Diagnostics 并在 “应用程序”窗口中选择 “整个企业”，然后选择任何链接打开 Diagnostics 视图。
4. 在 “视图”窗格中，将显示 “系统视图”视图组。打开视图组，并选择 “系统运行状况”视图或 “系统容量”视图。

您也可以检查 < **探测器安装目录** >\log\<probe_id>\probe.log 文件中的条目。如果文件中没有条目，则表明您尚未插桩 JRE，或者输入的 Java 参数（如 **Xbootclasspath**）不正确。在 **probe.log** 文件中查找错误，并找到“Successfully downloaded first command”条目，该条目表示探测器和服务器之间已建立通信。

继续下一部分以执行安装后任务。

关于其他配置和自定义插桩

还可以执行一些其他配置以及可选的自定义插桩。请参阅以下说明：

- “配置 SOAP 消息处理程序”（第 150 页）
- “将探测器属性指定为 Java 系统属性”（第 151 页）
- “可选的高级配置”（第 151 页）
- “可选的自定义插桩”（第 152 页）
- 有关使用代理配置环境的信息，请参阅 “针对 HTTP 代理配置 Diagnostics 服务器和代理”（第 627 页）；有关防火墙的信息，请参阅 “配置 Diagnostics 以在防火墙环境中工作”（第 631 页）；有关启用 HTTPS，请参阅 “在组件之间启用 HTTPS”（第 797 页）。

配置 SOAP 消息处理程序

要支持以下功能，必须使用 Diagnostics SOAP 消息处理程序：

- 收集 SOAP 错误的负载。
- 根据 SOAP 标头、正文或包络确定 SOA 用户 ID。

对于大多数应用程序服务器，写入插桩点和代码段是为了自动为正在监控的 Web 服务配置 Diagnostics 处理程序。

对于 WebSphere 5 JAX-RPC 和 Oracle 10g JAX-RPC，必须手动执行某些步骤才能配置 SOAP 处理程序。请参阅 “加载 Diagnostics SOAP 消息处理程序”（第 237 页）。

将探测器属性指定为 Java 系统属性

可以在应用程序服务器的启动命令行中将所有探测器属性指定为 Java 系统属性，但 **dynamic.properties** 属性文件中定义的探测器属性除外。当多个 JVM 使用单个探测器时，此操作将十分有用。

要将属性指定为 Java 系统属性，请将字母 **D** 和属性文件名的第一部分放在属性名前面。以下示例对此进行了说明。

- 要从启动命令设置 **probe.properties** 中的 **id** 属性，请将 **D** 和属性文件名中的 **probe** 串联，然后添加指定的属性名称，即 **id**，如下所示：

```
-Dprobe.id=SomeId
```

- 要从启动命令设置 **probe.properties** 中的 **active.products** 属性，请将 **D** 和属性文件名中的 **probe** 串联，然后添加指定的属性名称，即 **active.products**，如下所示：

```
-Dprobe.active.products=Enterprise,TV
```

- 要从启动命令设置 **dispatcher.properties** 中的 **registrar.url** 属性，请将 **D** 和属性文件名中的 **dispatcher** 串联，然后添加指定的属性名称，即 **registrar.url**，如下所示：

```
-Ddispatcher.registrar.url=http:/</host01.company.com>:2006/commander/registrar
```

可选的高级配置

请确定适用于您的环境的高级探测器配置，请参阅第 12 章，“高级 Java 代理与应用程序服务器配置”。

可选的自定义插桩

如有需要，还可以配置自定义插桩。有关详细信息，请参阅第 9 章，“Java 应用程序的自定义插桩”。

在 z/OS 大型计算机上安装 Java 代理

本节提供有关从 Diagnostics 安装光盘中的 .tgz 文件安装 Java 代理的说明。

在安装 Java 代理并将其配置为 z/OS 环境中的 Java 代理之前，请注意以下事项：

- Diagnostics Java 代理已安装且大量使用 z/OS 上的 Unix 系统服务环境 (USS)。
- 在 z/OS 环境中执行安装后， Java 代理会认为 Diagnostics 属性文件是 EBCDIC 格式，而不是 ASCII 格式。可使用 EBCDIC 编辑器更新属性文件，并使用相同格式存储更新后的文件。
- 系统不会为 z/OS 捕获系统度量，但您可以对 Diagnostics Java 代理进行配置，以便捕获有限数量的系统级别度量。

  有关如何捕获 z/OS 中系统度量的详细信息，请参阅 “启用 z/OS 系统度量捕获”（第 677 页）。

在 z/OS 上从 Diagnostics 安装光盘安装 Java 代理

Diagnostics 安装光盘上的 .tgz 文件包含 Java 代理文件，用于在 z/OS 大型机上安装 Java 代理。

要在 z/OS 大型计算机上安装 Java 代理，请执行以下操作：

1 将 HP Diagnostics 安装光盘上 **Diagnostics_Installers** 文件夹中的 **HPDiagTVJavaAgt_< 版本号 >_zos.tgz** 上载到 z/OS 系统上的目录，在此目录中解压缩安装程序。

2 如下例所示，使用 gzip 解压缩 HPDiagTVJavaAgt_< 版本号 >_zos.tgz：

```
gzip -d HPDiagTVJavaAgt_9.10_zos.tgz
```

此命令将创建解压缩文件 HPDiagTVJavaAgt_< 版本号 >_zos.tar。

3 要解压缩 .tar 文件，请如下例所示运行 tar 命令：

```
tar -xfp HPDiagTVJavaAgt_9.10_zos.tar
```

此命令将创建解压缩后的目录 **JavaAgent**。

4 确保路径中包含 Java 可执行文件，然后运行 Java 代理安装模块以将 Java 代理仅配置为 Profiler 或配置为与 Diagnostics 服务器和 / 或 TransactionVision 处理器配合使用的 Java 代理。有关详细信息，请参阅 “运行 Java 代理安装模块”（第 139 页）。例如 （适用于 shell）：

```
setenv PATH /u/Java6_31/J6.0/bin:/bin
```

然后：

```
<probe_install_dir>/bin/setupModule.sh
```

5 在安装了代理并运行安装模块后，您必须插桩应用程序服务器使用的 JRE，然后配置应用程序服务器 JVM 参数以调用 Java 代理。请参阅第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”。

6 验证代理安装，如 “验证 Java 代理安装”（第 149 页）所述。

7 根据需要完成安装后配置。请参阅“关于其他配置和自定义插桩”（第 150 页）。

在多台 z/OS 计算机上安装 Java 代理

如果计划在多台 z/OS 计算机上安装 Java 代理，则可能需要在第一台计算机上创建一个供代理实现用的 pax 归档文件，然后使用此 pax 归档文件将代理安装到其他计算机上。有关详细信息，请联系系统管理员。

使用常规安装程序安装 Java 代理

Java 代理安装程序支持在组件已经认证的所有平台上安装代理。但是，代理可能会在尚未经过认证的其他平台上工作，这时，则可使用产品安装光盘中提供的常规安装程序，在这些未经认证的平台上安装代理。

要让代理在不受常规安装程序支持的平台上工作，请运行常规安装程序并手动将代理配置为 “Java 探测器”，以便代理与其他 Diagnostics 组件通信并监控应用程序的处理。

要在未经认证的平台上安装和配置 Java 代理，请执行以下操作：

1 在 HP Diagnostics 安装盘的 **Diagnostics_Installers** 文件夹中找到 **HPDiagTVJavaAgt_< 版本号 >_unix.tgz**。

2 如下例所示，使用 gzip 解压缩 **HPDiagTVJavaAgt_< 版本号 >_unix.tgz**：

```
gzip -d HPDiagTVJavaAgt_9.10_unix.tgz
```

此命令完成后，解压缩名为 **HPDiagTVJavaAgt_< 版本号 >_unix.tar** 的文件。

3 要解压缩 tar 文件，请运行以下 tar 命令：

```
tar -xfp HPDiagTVJavaAgt_9.10_unix.tar
```

此命令将创建解压缩后的目录 **JavaAgent**。

4 运行 Java 代理安装模块将 Java 代理仅配置为 Profiler 或配置为与 Diagnostics 服务器和 / 或 TransactionVision 处理器配合使用的 Java 代理。有关详细信息，请参阅 “运行 Java 代理安装模块”（第 139 页）。

```
<probe_install_dir>/bin/setupModule.sh
```

5 在安装了代理并运行安装模块后，您必须插桩应用程序服务器使用的 JRE，然后配置应用程序服务器 JVM 参数以调用 Java 代理。请参阅第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”。

6 验证代理安装，如 “验证 Java 代理安装”（第 149 页）所述。

7 有关详细信息，请参阅 “关于其他配置和自定义插桩”（第 150 页）。

静默安装 Java 代理

支持静默安装 Java 代理。“静默安装”是指无需用户交互的自动执行的安装。静默安装会在每个安装步骤接受来自响应文件的输入，以代替用户输入。

在多台计算机上执行静默安装之前，必须生成用于在安装过程期间提供输入的响应文件。此响应文件可用于需要在安装期间输入相同内容的所有静默安装。

重要信息：对于每个新版本的 Diagnostics，应该先重新记录响应文件，然后在多台计算机上执行静默安装。

响应文件的后缀为 **.rsp**。可以使用标准的文本编辑器编辑响应文件。

静默安装使用两个响应文件：一个用于 Java 代理安装，另一个用于 Java 代理设置模块。

要生成供代理安装的响应文件，请执行以下操作：

➤ 使用以下命令行选项执行常规安装：

```
<installer> -options-record <installResponseFileName>
```

其中 <installResponseFileName> 是完全限定文件。此命令可创建其中包含在安装期间提交的所有信息的响应文件。

要生成 Java 代理设置模块的响应文件，请执行以下操作：

➤ 运行带有以下命令行选项的 Java 代理设置模块。

在 Windows 上：

```
<probe_install_dir>\bin\setupModule.cmd  -createBackups  -console  -recordFile
   <JASMResponseFileName>
```

在 UNIX 上：

```
<probe_install_dir>/bin/setupModule.sh  -createBackups  -console  -recordFile
   <JASMResponseFileName>
```

其中 <JASMReponseFileName> 是完全限定文件。这两个命令都可创建包含安装期间提交的所有信息的响应文件。

要执行 Java 代理静默安装，请执行以下操作：

➤ 使用 Java 代理安装的响应文件执行静默安装。

首先设置环境变量，然后运行带有以下 **-silent** 命令行选项的安装程序：

```
set HP_JAVA_AGENT_SETUP=-DoNotRun
<installer>  -silent  -options <installResponseFileName>
```

在 UNIX 系统上，使用引号指定环境变量。

```
set HP_JAVA_AGENT_SETUP="-DoNotRun"
```

要使用 Java 代理设置模块执行静默配置，请执行以下操作：

➤ 使用 Java 代理设置模块的响应文件执行静默安装。

取消设置环境变量，然后运行带有以下 **-silent** 命令行选项的 Java 代理设置模块：

```
set HP_JAVA_AGENT_SETUP=
<setupModule>  - silent  -createBackups  -console  -installFile <JASMResponseFileName>
```

在 UNIX 系统上，使用空引号取消设置环境变量。

```
set HP_JAVA_AGENT_SETUP=""
```

要在执行静默安装时在响应文件名之后指定两个其他选项，请执行以下操作：

➤ 通过指定 **is:log < 日志文件路径 >** 选项，可以创建日志文件。

➤ 通过指定 **is:tempdir< 临时目录路径 >** 选项，可以将临时目录更改为用户指定的目录。

设置文件权限 （仅适用于 UNIX）

仅在 UNIX 上，安装 Java 代理后，使代理的 “组” 与应用程序服务器的 “组” 相同。然后，在探测器安装目录中为组中的文件分配以下权限：

- 对 < **探测器安装目录** > 目录和文件的读取访问权限。
- 对 < **探测器安装目录** >/**bin** 目录的执行访问权限。
- 对 < **探测器安装目录** >/**log** 目录的读取 / 写入访问权限。

根据组织的安全要求，您可能希望进一步限制对此目录的访问；例如：

```
chmod 775 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/log
```

确定 Java 代理的版本

在请求支持时，对于有疑问的 Diagnostics 组件，了解其版本十分有用。

您可以使用以下任意一种方法来确定 Java 代理的版本：

- 查找版本文件 < **探测器安装目录** >**version.txt**。此文件包含四位数的版本号和内部版本号。
- 在探测器日志文件（< **探测器安装目录** >/**log**/<**probe_id**>/**probe.log**）中获取版本号。
- 还可以在 Diagnostics UI 的 “系统运行状况” 视图中获取版本号（请参阅附录 D，“使用管理员的系统视图”）。

卸载 Java 代理

要卸载 Java 代理，请执行以下操作：

- 在 Windows 计算机上，选择 “开始” > “所有程序” > “HP Java Agent” > “Uninstaller”。

或者，运行位于 **< 探测器安装目录 >_uninst** 目录中的 **uninstaller.exe**。

- 在 Linux 或 Solaris 计算机上，运行 **< 探测器安装目录 >/_uninst** 目录中的 **uninstaller.bin**。
- 在其他 UNIX 计算机上，选择 1.4 或更高版本的 JVM，并运行 **java -jar < 探测器安装目录 >/_uninst/uninstall.jar** 以卸载 Java 代理。

同时，请记住删除启动时添加到应用程序服务器的 Java 代理参数。

6

准备应用程序服务器以使用 Java 代理进行监控

本章描述如何准备应用程序服务器，以便 HP Diagnostics Java 代理能够监控应用程序。

本章包括：

有关准备应用程序服务器以进行监控

安装 HP Diagnostics Java 代理之后，必须准备 （插桩）应用程序服务器以允许 Java 代理监控应用程序。此准备过程通常包含 “插桩由应用程序服务器使用的 JRE”，以及 “配置应用程序服务器” JVM 参数以调用 Java 代理。

Diagnostics JRE 插桩不会修改已安装的 JRE，而会将已插桩的类的副本放置在 Java 代理安装目录下。然后，通过合适的 JVM 参数，这些插桩的类将会加载到运行应用程序服务器的 JVM 中。插桩是通过使用 Diagnostics JRE Instrumenter 实用程序完成的。可使用多种选项自动调用或手动调用该程序。

存在两种级别的插桩：

- 基本插桩。

  将 Java 代理添加到应用程序服务器启动项之后， Java 代理将插桩和监控应用程序服务器。这是通过将 -javaagent 选项添加到应用程序服务器 JVM 参数来实现的。

- 推荐插桩。

  除基本插桩之外，还建议使用 Java 代理提供的 JRE Instrumenter 实用程序插桩应用程序服务器所使用的 JRE（Java 运行时环境）。通过这种额外插桩，Java 代理将提供一些高级功能，如正在申请专利的回收泄漏定位 (CLP) 功能。 CLP 将自动检测泄漏集，并提供对泄露发生位置的堆栈跟踪。这样，可帮助尽早发现问题，以便有充足时间缓解问题 （如会导致最终的内存不足错误 / 服务器崩溃的问题），并可通过避免分析堆转储这类繁琐的任务来节省开发人员的时间（请参阅 “配置收集泄漏定位”（第 337 页））。此外，该额外的插桩在特定应用程序服务器 （如 WebSphere 6.1）上还具有性能优势。

有关如何使用各种 JRE 插桩模式的常规说明，请参阅 “关于 JRE Instrumenter 以及各种要调用的选项”（第 217 页）。

对于使用 JRE 1.4 的旧版应用程序服务器（如 WebLogic 8.1 和 WebSphere 5.1/6.0），基本插桩不可用；必须对其使用推荐的插桩。

然后，请在以下列表中找到您的应用程序服务器，并按照关于插桩和配置操作的说明进行操作。

关于配置应用程序服务器的示例

本节提供如何配置各种常用应用程序服务器以进行监控的示例。有关可用于调用 JRE Instrumenter 的各种方法的描述信息，请参阅 “关于 JRE Instrumenter 以及各种要调用的选项”（第 217 页）一节。

重要信息：在更改任何应用程序服务器配置之前，请确保您了解启动脚本的结构、属性值的设置方法，以及环境变量的使用方法。另外，在进行更改之前，请始终对计划更新的所有文件进行备份。

“示例 1：配置 GlassFish”（第 164 页）

“示例 2：配置 JBoss”（第 167 页）

“示例 3：配置 Oracle”（第 171 页）

“示例 4：配置 SAP NetWeaver”（第 177 页）

“示例 5：配置 TIBCO ActiveMatrix/BusinessWorks”（第 181 页）

“示例 6：配置 Tomcat”（第 185 页）

“示例 7：配置 WebLogic”（第 190 页）

“示例 8：配置 WebSphere”（第 194 页）

“示例 9：配置 WebMethods”（第 210 页）

有关各平台所支持的应用程序服务器版本的最新信息，请参阅 HP Diagnostics Support Matrix （网址为 http://support.openview.hp.com/sc/support_matrices.jsp），或与 HP 客户支持联系。

注意：

- 本章所示的脚本示例可能包含换行符，以便于阅读。但实际脚本中并没有换行符，命令文本会根据需要在屏幕上自动换行。
- 如果在指定的路径中有空格，请使用引号。

示例 1：配置 GlassFish

配置 GlassFish 应用程序服务器涉及修改其配置文件，以添加 JVM 参数。以下说明描述常规 GlassFish 3.x 应用程序服务器实施。站点管理员应使用这些说明来指导您对特定环境进行适当更改。

对于 GlassFish 应用程序服务器，使用隐式模式为其配置 JRE 插桩 （请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页））。

要配置 GlassFish 应用程序服务器，请执行以下操作：

1 在 GlassFish 配置文件中查找 **org.osgi.framework.bootdelegation** 属性，并将 **com.mercury.opal.capture.proxy** 附加到其末尾 （还需要逗号作为分隔符）。

在 GlassFish 3.1.2 中，此属性位于 **<GlassFish 安装目录 >/glassfish/config/osgi.properties** 中。

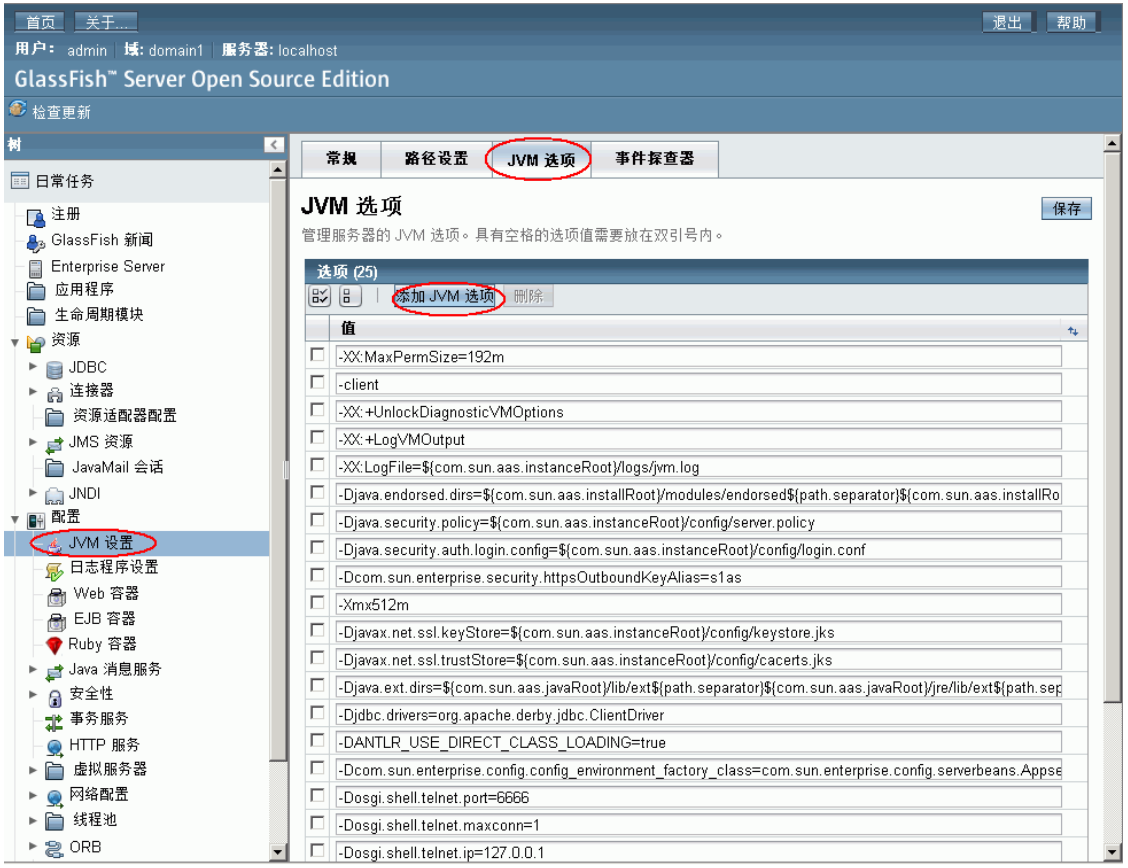
在较早版本的 GlassFish 中，此属性可能位于以下两个文件中：

<GlassFish 安装目录 >/osgi/equinox/configuration/config.ini

<GlassFish 安装目录 >/osgi/felix/conf/config.properties

2 通过使用以下步骤登录 GlassFish 管理控制台并转至 “JVM 选项” 页面：

对于 GlassFish 3.1.2，在左侧树中，转至 “配置” > “{config_name}” > “JVM 设置”，其中 {config_name} 为服务器配置名称 （如 “server-config”），然后选择 “JVM 选项” 选项卡。请参考以下屏幕截图。



如果使用的是较早版本的 GlassFish，请单击左侧树中的 “应用程序服务器”，然后选择顶部的 “JVM 设置” 选项卡。然后选择 “JVM 选项” 选项卡。

3 单击“添加 JVM 选项”按钮依次添加两个 JVM 参数。第一个参数 (-javaagent) 可使应用程序服务器 JVM 在启动时调用 Java 代理。首次调用时，第二个参数 (-Xbootclasspath) 可使应用程序服务器 JRE 得以插桩。在 -Xbootclasspath 参数中，输入一个名称以指定用于存储所插桩类的目录的名称。应使用您所选的名称取代以下示例中的 **MyServer**。

以下为针对 Windows 环境的示例：

```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre
```

以下为针对 UNIX 环境的示例：

```
-javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/MyServer/instr.jre
```

注意：对于群集设置，还需添加以下 JVM 参数：-Dprobe.id=<ProbeID>_%0 其中，< 探测器 ID> 为要分配给应用程序服务器群集的探测器名称。字符串“%0”将由唯一 ID 替换，以便区分不同探测器实例。

---

4 重新启动 GlassFish 应用程序服务器。

将调用 Java 代理，且 Java 代理会隐式运行 JRE Instrumenter 以插桩 JRE。

如果 GlassFish 应用程序没有启动，则可以检查和更改 **<GlassFish 安装目录 >/glassfish/domains/< 域名 >/config/domain.xml** 文件中的 JVM 参数以解决此问题，其中 < 域名 > 为域的名称 （如 **domain1**）。

如果由于以下错误而导致 GlassFish 应用程序服务器花长时间初始化且启动失败：

无法加载 Logmanager "com.sun.enterprise.server.logging.ServerLogManager"

则添加以下 JVM 选项
-Ddiag.agent.init.delay.ms=< 延迟毫秒数 >
其中 < 延迟毫秒数 > 是毫秒数（例如 6000）。可增加 < 延迟毫秒数 >，直到不再出现错误为止。

5 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/< 探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明未正确设置 JVM 参数。请在 GlassFish 服务器日志中查找错误消息。

6 或者，重新启动应用程序服务器，以便其使用已插桩的 JRE。

重要信息：如果将来要更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），则在重新启动应用程序服务器之前，必须删除 **<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/MyServer** 目录 （请使用您的目录名称代替 MyServer），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。有关详细信息，请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。

示例 2：配置 JBoss

JBoss 应用程序服务器由 shell 或命令脚本启动。因此，建议您修改启动脚本以配置 JBoss 应用程序服务器。由于站点管理员会经常自定义 JBoss 提供的文件，我们无法提供适用于每种情况的详细配置说明。因此，以下各节将提供对 JBoss 应用程序服务器执行常规实施的说明，并提供特定示例。站点管理员应使用这些说明来指导您在自定义环境中进行相关更改。

对于 JBoss 应用程序，使用显式模式为其配置 JRE 插桩 （请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页））。

要使用 “自动显式”模式，需要完成两个任务：

- 使用应用程序服务器所使用的相同的 JRE 修改用以运行 JRE Instrumenter 的应用程序服务器启动脚本。JRE Instrumenter 的输出将为您提供需在下一任务中使用的 JVM 参数。
- 配置 JRE Instrumenter 的输出中的应用程序服务器 JVM 参数。

要配置 JBoss 应用程序服务器，请执行以下操作：

1 查找用于为应用程序启动 JBoss 的启动脚本，例如：

- 在低于 7.0 的 JBoss 版本上：

  启动脚本文件通常位于与以下示例类似的路径中：

  <JBOSS_HOME>\bin\run.[bat|sh]

  其中 **<JBOSS 主 >** 是 JBoss 安装目录的路径，如 **C:\ jboss-4.2.3.GA or C:\jboss-6.0.0.Final**。

- 在 JBoss 7.0 或更高版本上：

  启动脚本文件通常位于与以下示例之一相类似的路径中：

  <JBOSS_HOME>\bin\domain.[bat|sh]

  <JBOSS_HOME>\bin\standalone.[bat|sh]

  其中 **<JBOSS_HOME>** 是 JBoss 安装目录的路径，如 **C:\jboss-as-7.1.0.Final**。

注意：如果 JBoss 应用程序服务器作为 Windows 服务启动，则在继续执行以下步骤之前，请确保可使用启动脚本启动此应用程序服务器，因为 Windows 服务会在后台调用启动脚本。

---

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 查找启动此应用程序服务器的 java 命令行 （或代码块）。

以下示例显示了 .bat 文件中的 java 命令行：

```
"%JAVA%" %JAVA_OPTS% ^  -Djava.endorsed.dirs="%JBOSS_ENDORSED_DIRS%"
^
-classpath "%JBOSS_CLASSPATH%" ^  org.jboss.Main %*
```

以下示例显示了 .sh 文件中的 java 命令行：

```
  if [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then
      # Execute the JVM in the foreground
      "$JAVA" $JAVA_OPTS \
        -Djava.endorsed.dirs="$JBOSS_ENDORSED_DIRS" \
        -classpath "$JBOSS_CLASSPATH" \
        org.jboss.Main "$@"
      JBOSS_STATUS=$?
    else
      # Execute the JVM in the background
      "$JAVA" $JAVA_OPTS \
        -Djava.endorsed.dirs="$JBOSS_ENDORSED_DIRS" \
        -classpath "$JBOSS_CLASSPATH" \
        org.jboss.Main "$@" &
      JBOSS_PID=$!
      . ……
```

4 直接在以上 java 命令行 （或代码块）添加两行。第一行调用 java 命令以运行 JRE Instrumenter ；第二行添加所需的 JVM 参数。如果不确定要使用哪些 JVM 参数，则可以稍后在步骤 6 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以下示例中的 **MyServer**。

以下示例显示了在使用 JRE 5 或更高版本的 JBoss 应用程序服务器的 .bat 文件中添加的两行：

```
"%JAVA%" -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

以下示例显示了在使用 JRE 5 或更高版本的 JBoss 应用程序服务器的 .sh 文件中添加的两行：

```
"$JAVA" -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/jreinstrumenter.jar -f MyServer

JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/
MyServer/instr.jre -javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

以下示例显示了在使用 JRE 1.4 版本的 JBoss 应用程序服务器的 .bat 文件中添加的两行：

```
"%JAVA%" -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA
gent\DiagnosticsAgent\classes\boot
```

注意：如果您的 java 命令行不使用 JAVA_OPTS 变量定义 JVM 参数，则需要将这些示例中所显示的变量名称 JAVA_OPTS 更改为正确的名称。

重要信息：

- 对于 JBoss 6 或更高版本，请将以下 JVM 参数添加到 JAVA-OPTS：
  -Djava.util.logging.manager=org.jboss.logmanager.LogManager
- 对于 JBoss 7 或更高版本，请将以下 JVM 参数添加到 JAVA-OPTS：
  -Djboss.modules.system.pkgs=org.jboss.byteman,com.mercury.opal

5 保存对启动脚本的更改，并使用经过修改的脚本重新启动应用程序服务器。

6 为了帮助捕获错误或拼写错误，请执行启动脚本，查找 JRE Instrumenter 的输出（搜索 Xbootclasspath）。如果已在步骤 4 中向启动脚本添加了第二行（设置 JVM 参数），请将该行与 JRE Instrumenter 输出中的 JVM 参数进行对比。如果它们不相同，则使用 JRE Instrumenter 所提供的正确 JVM 参数更新启动脚本，然后重新启动应用程序服务器。如果未在步骤 4 中向启动脚本添加第二行（设置 JVM 参数），则现在进行添加，并重新启动应用程序服务器。相关示例，请参阅步骤 4。

7 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩未成功，或者配置的 JVM 参数不正确。有关详细信息，请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

示例 3：配置 Oracle

本节提供关于配置 Oracle 9i 或 10g 应用程序服务器的说明。

关于配置 Oracle9i 应用程序服务器：

通过将 JRE Instrumenter 所提供的 JVM 参数添加到用于启动应用程序服务器的 XML 文件中，可以配置 Oracle9i 应用程序服务器。

要配置 Oracle9i 应用程序服务器，请执行以下步骤：

1 在服务器启动后，查找用于控制应用程序服务器配置的 XML 文件。此文件通常位于 **<Oracle 9iAS 安装目录 >/opmn/conf/opmn.xml**。

2 对 **opmn.xml** 文件进行备份，并使用编辑器打开要编辑的 **opmn.xml** 文件。

3 手动运行 JRE Instrumenter 以插桩 Oracle 应用程序服务器所使用的 JRE。（有关如何手动运行 JRE Instrumenter 的说明，请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。）

4 将已从 JRE Instrumenter 结果（如 **Xbootclasspath** 属性）中复制出来的 Java 参数添加到 *java-option value* 属性。

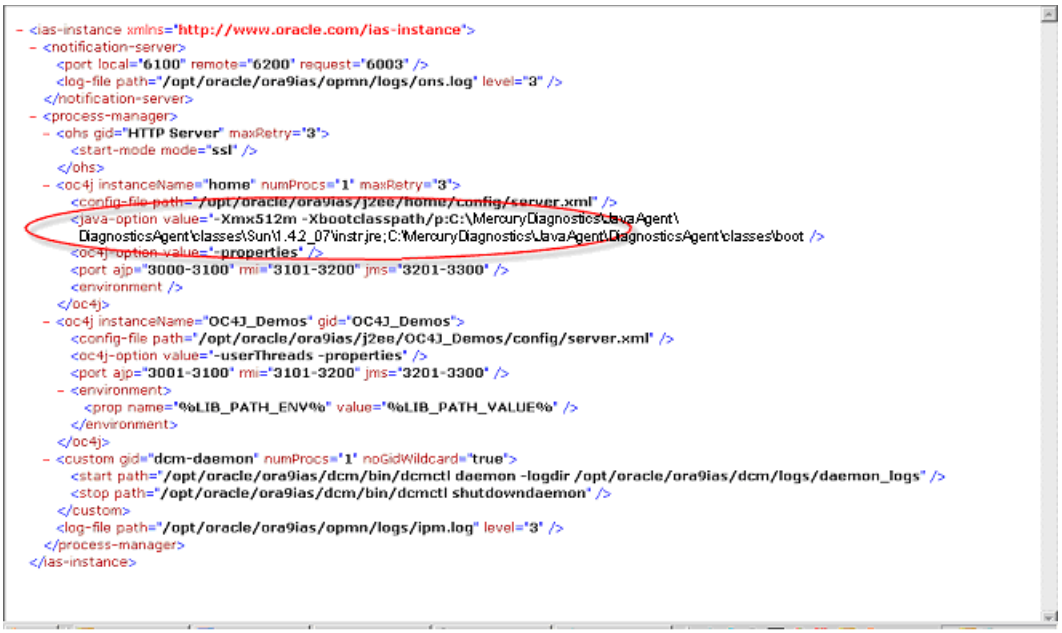
以下是 **Xbootclasspath** 参数的一个示例：

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\Sun\1.4.2_04\instr.jre;
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

注意： 修改 **-Xbootclasspath** 参数时，如果在指定的路径中有空格，请使用引号。

下图是在添加 **Xbootclasspath** 参数之前 Oracle 9iAS 启动脚本的示例：

下图是在添加 **Xbootclasspath** 参数之后 Oracle 9iAS 启动脚本的示例：



5 保存对 XML 文件的更改，并重新启动 Oracle 应用程序服务器。

6 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明您尚未运行 JRE Instrumenter，或者输入的 Java 参数 Xbootclasspath 不正确。

重要信息：如果将来要更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），则必须重新运行 JRE Instrumenter 以插桩新的 JRE 并相应更改 JVM 参数。否则，应用程序服务器可能不会启动。有关详细信息，请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。

要配置 Oracle 10g 应用程序服务器，请执行以下操作：

1 手动运行 JRE Instrumenter 以插桩 Oracle 应用程序服务器所使用的 JRE。（有关详细信息，请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。）JRE Instrumenter 将提供之后要使用的 JVM 参数（如 -Xbootclasspath 参数）。

2 打开 Oracle 的 “应用程序服务器控制台”，

3 单击 “主页 (或 MyOC4J)” 系统组件。

4 在 “OC4J: home” 页面上，选择 “管理”。

5 在 “管理”页面上选择 “服务器属性”。

6 在 “服务器属性”窗口中的 “命令行选项”下，将已从 JRE Instrumenter 结果复制出来的 JVM 属性添加到 “Java 选项”框中。

注意：必须在 /p 切换符之前添加 (^)，否则 Oracle 会将切换选项符 (/) 改为 (\)。

以下是插入了 (^) 的 Xbootclasspath 参数的一个示例。

```
-Xbootclasspath^/ p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\Sun\1.4.2_07\instr.jre;
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

7 应用更改，然后重新启动 Oracle 服务器。

8 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明您尚未运行 JRE Instrumenter，或者未正确输入的 Java 参数 （如 **Xbootclasspath**）。

重要信息：如果将来要更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），则必须重新运行 JRE Instrumenter 以插桩新的 JRE 并相应更改 JVM 参数。否则，应用程序服务器可能不会启动。有关详细信息，请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。

示例 4：配置 SAP NetWeaver

根据 SAP NetWeaver 应用程序服务器所使用的 JRE 版本，关于配置的说明信息会有所不同。以下两节提供了关于 JRE 版本 5 或更高版本的常规 NetWeaver 实施的说明，以及关于 JRE 版本 1.4 的常规 NetWeaver 实施的说明。站点管理员应使用这些说明来指导您对特定环境进行适当更改。

对于使用 JRE 版本 5 或更高版本的 SAP NetWeaver 应用程序服务器，请使用隐式模式配置 JRE 插桩 （请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页））。对于使用 JRE 版本 1.4 的 SAP NetWeaver 应用程序服务器，请手动运行 JRE Instrumenter （请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页））。

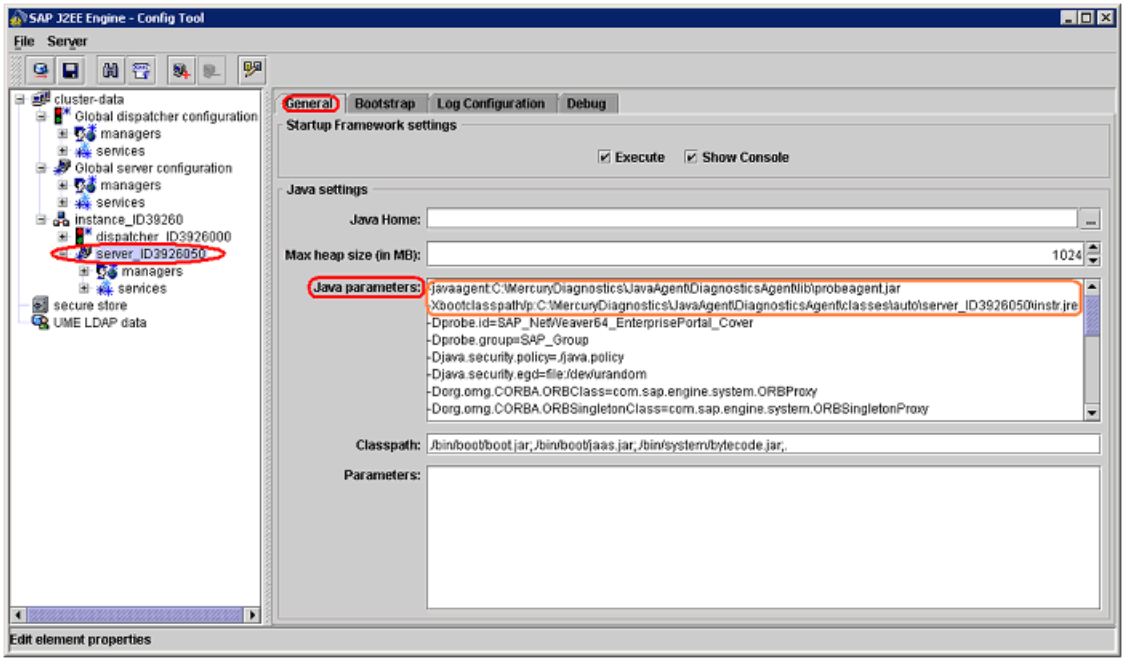
要配置使用 JRE 版本 5 或更高版本的 SAP NetWeaver 应用程序服务器，请执行以下操作：

1. 运行 NetWeaver J2EE 引擎配置工具。用于运行此工具的脚本名为 **configtool.bat**，位于 **usr\sap\CR2\JC00\j2ee\configtool** 目录中，其中 CR2 为 SAP 系统名称的示例。
2. 在配置工具 UI 的左侧树中，请选择要监控的服务器。例如，在以下屏幕截图中，选择 **cluster-data > instance_ID39260 > server_ID3926050**。然后在右侧选择 “General”选项卡，并向 “Java parameters”文本窗添加两个 JVM 参数。

第一个参数 (**-javaagent**) 可使应用程序服务器 JVM 在启动时调用 Java 代理。首次调用时，第二个参数 (**-Xbootclasspath**) 可使应用程序服务器 JRE 得以插桩。在 -Xbootclasspath 参数中，输入一个名称以指定用于存储插桩类的目录。在以下目录中，使用了名称 server_ID3926050。可以使用您所选的名称代替 server_ID3926050。

```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\server_ID3926050\instr.jre
```

以下是 JVM 参数示例的一个屏幕截图。



3 保存更改并退出配置工具。

4 编辑 **<Java 代理安装目录 >\etc\capture.properties** 文件，并为这些属性分配以下值：

event_buffer.size = 10000
event_buffer.flush.level = 1000

5 重新启动 SAP NetWeaver 应用程序服务器。

6 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/< 探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩可能已失败，或者输入的 JVM 参数不正确。在 NetWeaver bootstrap 日志中查找错误消息。

重要信息：如果将来要更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），则在重新启动应用程序服务器之前，必须删除 **<Java 代理安装目录 >/DiagnosticsAgent/classes/auto/server_ID3926050** 目录（使用您的目录名称代替 server_ID3926050），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。有关详细信息，请参阅“在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。

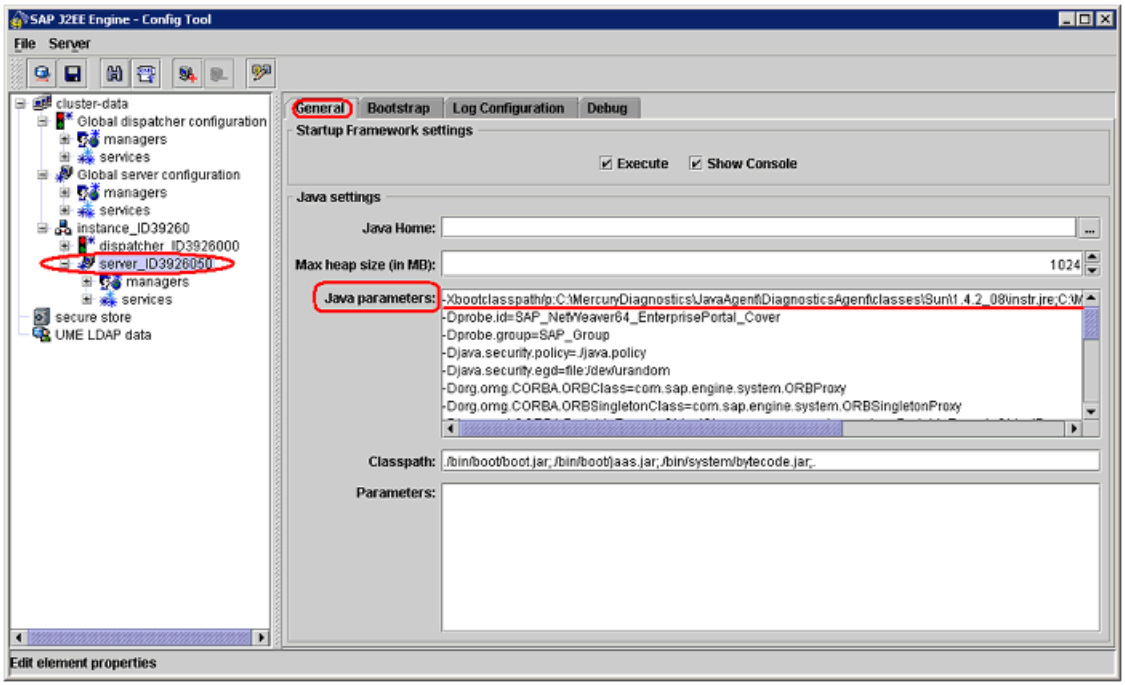
要配置使用 JRE 版本 1.4 的 SAP NetWeaver 应用程序服务器，请执行以下操作：

1 查找用于运行 NetWeaver 应用程序服务器的 JRE。

2 运行 JRE Instrumenter 以插桩此 JRE。JRE Instrumenter 将提供之后要使用的 JVM 参数 （如 **-Xbootclasspath** 参数） 。要了解如何运行 JRE Instrumenter 并将所生成的 JVM 参数复制到剪贴板以供下述步骤 4 使用，请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。

3 运行 NetWeaver J2EE 引擎配置工具。用于运行此工具的脚本名为 **configtool.bat**，位于 **usr\sap\CR2\JC00\j2ee\configtool** 目录中，其中 CR2 为 SAP 系统名称的示例。

4 在配置工具 UI 的左侧树中，选择要监控的服务器，例如 **cluster-data > instance_ID39260 > server_ID3926050**。在 “General” 选项卡中，可找到 “Java parameters” 文本窗口。将 JRE Instrumenter 所提供的 JVM 参数添加到文本窗口中。请参考以下屏幕截图。



5 保存更改并退出配置工具。

6 编辑 **<Java 代理安装目录 >\etc\capture.properties** 文件，并将以下值分配到这些属性

event_buffer.size = 10000
event_buffer.flush.level = 1000

7 重新启动 NetWeaver 应用程序服务器。

8 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/< 探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明未正确设置 JVM 参数。请在 NetWeaver bootstrap 日志中查找错误消息。

示例 5：配置 TIBCO ActiveMatrix/BusinessWorks

以下各节描述如何配置 TIBCO ActiveMatrix 和 BusinessWorks 以便对应用程序进行监控。

对于 TIBCO ActiveMatrix 和 BusinessWords 应用程序服务器，使用隐式模式为其配置 JRE 插桩 （请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页））。

重要信息：如果将来要更新应用程序服务器所使用的 JRE（如应用应用程序服务器修补程序），则在重新启动应用程序服务器之前，必须删除 **<Java 代理安装目录 >/DiagnosticsAgent/classes/auto/Tibco_Node1** 目录 （使用您的目录名称代替 Tibco_Node1），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。

要插桩 TIBCO 服务器，请按照以下适用于不同 TIBCO 版本的步骤中所示添加两个 JVM 参数。第一个参数 (-javaagent) 可使应用程序服务器 JVM 在启动时调用 Java 代理。首次调用时，第二个参数 (-Xbootclasspath) 可使应用程序服务器 JRE 得以插桩。在 -Xbootclasspath 参数中输入用于标识服务器的名称。在以下步骤中，请使用您所选的名称代替以粗体显示的示例名称。

要配置 TIBCO BusinessWorks，请执行以下步骤：

对于 TIBCO BusinessWorks，请根据部署应用程序的方式以及要监控的应用程序，将类似于以下示例的两个 JVM 参数附加到以下 TIBCO BusinessWorks 文件之一的 **java.extended.properties**。如果未显示 **java.extended.properties**，请添加该文件。以下示例显示了可用于添加此参数的方法之一：

```
java.extended.properties=-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/TIBCO_BW1/instr.jre
```

如果要监控之前部署的应用程序或仅监控 TIBCO BusinessWorks 中的特定应用程序，请使用 JVM 参数更新 **<BusinessWorks 安装目录 >\tra\domain\< 域名 >\application\< 应用程序名称 >\< 应用程序名称 >.tra**。要插桩的 JRE 是在 .tra 文件用户属性 **tibco.env.TIB_JAVA_HOME** 中指定的。

- 如果使用 TIBCO Administrator 部署了应用程序。使用 JVM 参数更新 **bwengine.tra**。
- 如果使用 TIBCO Designer 部署了应用程序。使用 JVM 参数更新 **bwengine.tra**。

要配置 TIBCO BusinessWorks 以进行 JMX 度量收集，请执行以下步骤：

要进行 TIBCO BusinessWorks JMX 度量收集，请启用对 Business Works 流程的 JMX 访问。通过将以下属性添加到在其中配置 Java 代理插桩的同一个 Business Works **.tra** 文件，可以完成此操作。

JmxEnabled=true

其他 JMX 度量由 TIBCO 使用的某些组件公开，如 Apache Tomcat 和 Pramati J2EE 服务器。默认情况下，不会收集其中某些度量（在 **metric.config** 文件中已注释掉这些度量）。可通过在 **metrics.config** 文件中取消注释这些度量来激活它们。不激活这些度量的原因是，它们主要表示性能调整配置参数，并且在应用程序生命周期中极少发生更改。

有关 JMX 度量收集的一般信息，请参阅“Java 代理 - JMX 度量捕获”（第 679 页）。

要配置 TIBCO ActiveMatrix Service Bus 2.0 和 2.3，请执行以下步骤：

1 对于 TIBCO ActiveMatrix Service Bus (AMSB) 2.0 和 2.3，请找到 AMSB **.tra** 文件并将 JVM 附加到 **java.extended.properties**。如果 java.extended.properties 不存在，请添加该文件。

以下是适用于 Windows 的一个示例，其中使用 **TIBCO_Node1** 作为名称：请注意使用斜杠 (/) 而非反斜杠 (\)。

```
java.extended.properties=-javaagent:jarC:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/TIBCO_Node1/instr.jre
```

2 此外，要在 AMSB 中查看出站 JMS Web 服务操作，需要使用正确版本更新 **etc\inst.properties** 中的 **details.conditional.properties**。目前支持 AMSB 版本 2.0 和 2.3。请确保仅启用这一个版本（方法是将一个版本设置为 “true” 并将其他版本设置为 “false”）。以下示例显示启用默认版本 AMSB 2.3：

```
details.conditional.properties= \
 mercury.enable.SOAPHandler=true, \
 mercury.enable.autoLoadSOAPHandler=true, \
 mercury.enable.resourcemonitor.jdbcConnection=false, \
 mercury.enable.resourcemonitor.jdbcStatement=false, \
 mercury.enable.resourcemonitor.jdbcResultSet=false, \
 mercury.enable.tibco.amsb2.0=false, \
 mercury.enable.tibco.amsb2.3=true
```

要配置 TIBCO ActiveMatrix Service Bus 3.1.2 生产环境，请执行以下操作：

1 对于生产环境中的 TIBCO ActiveMatrix Service Bus (AMSB) 3.1.2，找到以下 AMSB 文件：

<tibco amx 配置目录 >\data\tibcohost\< 企业名称服务器名称 >\tools\machinemodel\machine.xmi

2 为要监控的**每个节点**更新文件的 **runtimes** 部分。例如：

<runtimes xsi:type="machinemodel:OSGiRuntime" **name**="**Node1**"

在每个节点的 runtimes 部分中找到 **frameworkProperties** 键 **org.osgi.framework.bootdelegation**，并将 **com.mercury.*** 附加到此属性的值中。

例如：

<frameworkProperties key="org.osgi.framework.bootdelegation" value="com.ibm.*, ...,sun.*,**com.mercury.***"/>

3 然后为**每个节点**找到 .tra 文件。

<tibco amx 配置目录 >\data\tibcohost\< 企业名称服务器名称 >\nodes\< 节点名称 >\bin\tibams_< 节点名称 >.tra

在每个文件中，将两个 JVM 参数附加到 **java.extended.properties**。以下示例使用 **AMSB_Node1** 作为名称。

```
java.extended.properties=-javaagent:C:/ MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/ JavaAgent/DiagnosticsAgent/classes/auto/AMSB_Node1/instr.jre
```

要配置 TIBCO ActiveMatrix Service Bus 3.1.2 开发环境，请执行以下操作：

1 对于开发环境中的 TIBCO ActiveMatrix Service Bus (AMSB) 3.1.2，可以更新启动委派：

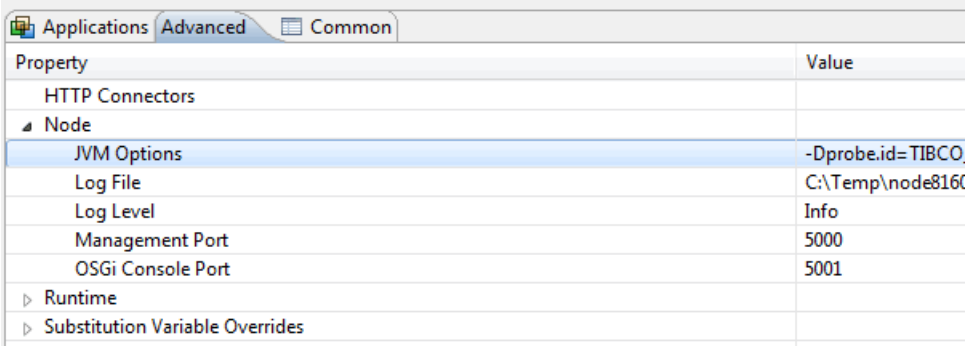
```
<tibco_amx_configuration_dir>\components\shared\1.0.0\plugins\com.tibco.metadata.h
pa.tibcohost.nodetype.integration_3.1.200.000\META-INF\com.tibco.amf.node.types\co
m.tibco.amf.hpa.tibcohost.node.integration.feature\3.1.200\provisioning.properties
```

并将 **com.mercury.*** 附加到 **org.osgi.framework.bootdelegation** 结尾。

例如：

```
# OSGi framework properties
org.osgi.framework.bootdelegation=\
...
 org.xml.*,\
 sun.*,\
 com.mercury.*
```

2 然后在 “Run Configuration” 的 “Advance” 选项卡中更新 JVM Options，以添加两个 JVM 参数。



以下示例使用 **AMSB_DevNode** 作为名称。

```
-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar
-Xbootclasspath/p:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/classes/auto/AMSB_DevNode/instr.jre
```

示例 6：配置 Tomcat

其他应用程序或服务器中经常嵌入 Apache Tomcat。因此，插桩它的方式可能会有所不同。以下各节提供有关如何在简单场景中配置 Tomcat 服务器的说明，但对特定情况仍具有指导意义。

如果 Tomcat 服务器由 shell 或 Windows 命令脚本启动，建议您修改启动脚本以对其进行插桩 （请参阅 “用启动脚本配置 Tomcat 服务器”（第 185 页））。

在 Windows 环境中，如果将 Tomcat 作为 Windows 服务安装且无脚本，建议您修改其 Java 选项以对其进行插桩。请参阅 “在不使用启动脚本的情况下配置 Tomcat 服务器”（第 189 页）。

用启动脚本配置 Tomcat 服务器

由于其他应用程序或站点管理员会经常自定义 Tomcat 提供的文件，我们无法提供适用于每种情况的详细配置说明。因此，以下各节将提供对 Tomcat 服务器执行常规实施的说明，并提供特定示例。站点管理员应使用这些说明来指导您在自定义环境中进行相关更改。

可以通过“自动显式”插桩模式，使用启动脚本配置 Tomcat 服务器（请参阅“在“自动显式”模式下使用 JRE Instrumenter”（第 219 页））。这包含两个任务：

- 使用应用程序服务器所使用的相同的 JRE 修改用以运行 JRE Instrumenter 的应用程序服务器启动脚本。JRE Instrumenter 的输出将为您提供需在下一任务中使用的 JVM 参数。
- 配置 JRE Instrumenter 的输出中的应用程序服务器 JVM 参数。

要使用启动脚本配置 Tomcat 服务器，请执行以下操作：

1 查找用于为应用程序启动 Tomcat 的启动脚本。

在某些场景中，启动脚本将以调用以下脚本为结尾，用以启动 Tomcat：

<Tomcat 安装目录 >/bin/catalina.[bat|sh]

其中 **<Tomcat 安装目录 >** 是 Tomcat 安装目录路径，如 **C:\apache-tomcat-7.0.8**。

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 找到启动 Tomcat 服务器的 **java** 命令行 （或代码块）。

以下是 **catalina.bat** 文件中的一个示例：

```
rem Execute Java with the applicable properties
if not "%JPDA%" == "" goto doJpda
if not "%SECURITY_POLICY_FILE%" == "" goto doSecurity
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% %DEBUG_OPTS%
-Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -classpath "%CLASSPATH%"
-Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%"
-Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS%
%ACTION%
……
```

以下是 catalina.sh 文件中的一个示例：

```
if [ "$1" = "-security" ] ; then
  if [ $have_tty -eq 1 ]; then
    echo "Using Security Manager"
  fi
  shift
  eval \"$_RUNJAVA\" \"$LOGGING_CONFIG\" $JAVA_OPTS $CATALINA_OPTS \
    -Djava.endorsed.dirs=\"$JAVA_ENDORSED_DIRS\" -classpath \"$CLASSPATH\" \
    -Djava.security.manager \
    -Djava.security.policy==\"$CATALINA_BASE/conf/catalina.policy\" \
    -Dcatalina.base=\"$CATALINA_BASE\" \
    -Dcatalina.home=\"$CATALINA_HOME\" \
    -Djava.io.tmpdir=\"$CATALINA_TMPDIR\" \
    org.apache.catalina.startup.Bootstrap "$@" start
else
  eval \"$_RUNJAVA\" \"$LOGGING_CONFIG\" $JAVA_OPTS $CATALINA_OPTS \
    -Djava.endorsed.dirs=\"$JAVA_ENDORSED_DIRS\" -classpath \"$CLASSPATH\" \
    -Dcatalina.base=\"$CATALINA_BASE\" \
    -Dcatalina.home=\"$CATALINA_HOME\" \
    -Djava.io.tmpdir=\"$CATALINA_TMPDIR\" \
    org.apache.catalina.startup.Bootstrap "$@" start
fi
```

4 直接在以上 java 命令行 （或代码块）添加两行。第一行调用 java 命令以运行 JRE Instrumenter ；第二行添加所需的 JVM 参数。如果不确定要使用哪些 JVM 参数，则可以稍后在步骤 6 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以下示例中的 **MyServer**。

以下示例显示了在使用 JRE 5 或更高版本的 Tomcat 服务器的 **catalina.bat** 文件中添加的两行：

```
%_EXECJAVA% -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

以下是使用 JRE 版本 5 或更高版本的 Tomcat 服务器的 **catalina.sh** 文件的一个示例：

```
"$_RUNJAVA" -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/ jreinstrumenter.jar -f MyServer

JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/ DiagnosticsAgent/classes/
MyServer/instr.jre -javaagent:/opt/MercuryDiagnostics/ JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

以下是使用 JRE 版本 1.4 的 Tomcat 服务器的 **catalina.bat** 文件的一个示例：

```
%_EXECJAVA% -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer

set JAVA_OPTS=%JAVA_OPTS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\Mercur
yDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

注意：如果您的 java 命令行不使用 JAVA_OPTS 变量定义 JVM 参数，则需要将这些示例中所显示的变量名称 JAVA_OPTS 更改为正确的名称。

---

5 保存更改并重新启动应用程序服务器。

6 在启动脚本的输出中，查找 JRE Instrumenter 的输出（搜索 Xbootclasspath）。如果已在步骤 4 中向启动脚本添加了第二行（设置 JVM 参数），请将该行与 JRE Instrumenter 输出中的 JVM 参数进行对比。如果它们不相同，则使用 JRE Instrumenter 所提供的正确 JVM 参数更新启动脚本，然后重新启动应用程序服务器。如果未在步骤 4 中向启动脚本添加第二行 （设置 JVM 参数），则现在进行添加，并重新启动应用程序服务器。相关示例，请参阅步骤 4。

7 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩未成功，或者配置的 JVM 参数不正确。有关详细信息，请参阅 “在 “自动显式” 模式下使用 JRE Instrumenter”（第 219 页）。

在不使用启动脚本的情况下配置 Tomcat 服务器

以下说明描述如何配置作为 Windows 服务运行的 Tomcat 服务器：

要在不使用启动脚本的情况下配置 Tomcat 服务器，请执行以下操作：

1 在 Windows 任务栏中，右键单击 Apache Tomcat 服务图标，然后选择 “配置”。或者，也可以从“开始”菜单中导航。例如，“程序”>“Apache Tomcat 7.0”>“配置 Tomcat”。

2 在 “Apache Tomcat 属性”对话框中，选择 “Java”选项卡。

3 在“Java 选项”框中，按如下所示添加两个 JVM 参数。第一个参数 (-javaagent) 可使应用程序服务器 JVM 在启动时调用 Java 代理。首次调用时，第二个参数 (-Xbootclasspath) 可使应用程序服务器 JRE 得以插桩。在 -Xbootclasspath 参数中，输入一个名称以指定用于存储所插桩类的目录的名称。应使用您所选的名称取代以下示例中的 **MyServer**。

```
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre
```

重要信息：每个 JVM 参数必须独立成行。

---

4 重新启动 Tomcat 服务。

将调用 Java 代理，且 Java 代理会隐式运行 JRE Instrumenter 以插桩 JRE。

5 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/< 探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明未正确设置 JVM 参数。在 **<Tomcat 安装目录 >/logs/catalina.< 日期 >.log** 文件中查找错误消息，其中 < 日期 > 为当日的日期。

6 或者，重新启动应用程序服务器，以便其使用已插桩的 JRE。

重要信息：如果将来要更新 Tomcat 服务器所使用的 JRE，则在重新启动 Tomcat 服务器之前，必须删除 **<Java 代理安装目录 >/DiagnosticsAgent/classes/auto/MyServer** 目录（使用您的目录名称代替 MyServer），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。有关所用插桩模式的常规信息，请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。

示例 7：配置 WebLogic

WebLogic 应用程序服务器由 shell 或命令脚本启动。因此，建议修改启动脚本以插桩 WebLogic 应用程序服务器。

由于站点管理员会经常自定义 WebLogic 提供的启动脚本，我们无法提供适用于所有情况的详细配置说明。因此，下节将提供对 WebLogic 应用程序服务器执行常规实施的常规说明，并提供特定示例。站点管理员可使用这些说明来指导您在自定义环境中进行相关更改。

使用 “自动显式”插桩模式配置 WebLogic 服务器 （请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页））。这包含两个任务：

- 使用应用程序服务器所使用的相同的 JRE 修改用以运行 JRE Instrumenter 的应用程序服务器启动脚本。JRE Instrumenter 的输出将为您提供需在下一任务中使用的 JVM 参数。
- 配置 JRE Instrumenter 的输出中的应用程序服务器 JVM 参数。

要配置 WebLogic 应用程序服务器，请执行以下操作：

1 查找用于为您的域启动 WebLogic 的启动脚本。

➤ 在 WebLogic 9.0 或更高版本上：

启动脚本文件通常位于与以下示例类似的路径中：

< 域主目录 >\bin\startWebLogic.[cmd|sh]

其中 **< 域主目录 >** 为域目录路径，如 **C:\bea\user_projects\domains\< 域名 >**；或 **C:\bea\wlserver_10.0\samples\domains\< 域名 >**，其中 **< 域名 >** 为域的名称。

例如，如果域名是 **MedRec**，则路径结构可能如下所示：

C:\bea\wlserver_10.0\samples\domains\medrec\bin\startWebLogic.cmd

➤ 在 WebLogic 8.1 上：

启动脚本文件通常位于与以下示例之一相类似的路径中：

<WLS 主目录 >\server\bin\startWLS.[cmd|sh]

其中 **<WLS 主目录 >** 为 WebLogic 安装目录的路径，如 **C:\bea\weblogic81**

<DOMAIN 主目录 >\start< 域名 >Server.[cmd|sh]

其中 **<DOMAIN 主目录 >** 为域目录的路径，如 **C:\bea\user_projects\domains\< 域名 >** 或 **C:\bea\weblogic81\samples\domains\< 域名 >**，其中 **< 域名 >** 为您的域名。

例如，如果域名是 **MedRec**，则路径结构可能如下所示：

C:\bea\weblogic81\samples\domains\medrec\startMedRecServer.cmd

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 找到启动应用程序服务器的 **Java** 命令行。以下是 .cmd 文件中的一个示例：

```
%JAVA_HOME%\bin\java %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-Dweblogic.Name=%SERVER_NAME%
-Djava.security.policy=%WL_HOME%\server\lib\weblogic.policy
%PROXY_SETTINGS% %SERVER_CLASS%
```

4 直接在以上 java 命令行 （或代码块）添加两行。第一行调用 java 命令以运行 JRE Instrumenter ； 第二行添加所需的 JVM 参数。如果不确定要使用哪些 JVM 参数，则可以稍后在步骤 6 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以下示例中的 **MedRec**。

以下示例显示了在 WebLogic 9.x 或更高版本中的 .cmd 文件中添加的两行：

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MedRec

set JAVA_OPTIONS=%JAVA_OPTIONS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MedRec\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

以下示例显示了在 WebLogic 9.x 或更高版本中的 .sh 文件中添加的两行：

```
${JAVA_HOME}/bin/java -jar /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/jreinstrumenter.jar -f
MedRec

JAVA_OPTIONS="$JAVA_OPTIONS -Xbootclasspath/p:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/
classes/MedRec/instr.jre -javaagent:/opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/probeagent.jar"
```

以下示例显示了在 WebLogic 8.1 中的 .cmd 文件中添加的两行：

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MedRec

set JAVA_OPTIONS=%JAVA_OPTIONS% -Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MedRec\instr.jre;C:\MercuryDiagnostics\JavaAg
ent\DiagnosticsAgent\classes\boot
```

注意：

- 如果您的 java 命令行不使用 JAVA_OPTIONS 变量定义 JVM 参数，则需要将这些示例中所显示的变量名称 JAVA_OPTIONS 更改为正确的名称。
- 在具有 JRockit JR 的 WebLogic 8.1 上，请将以下 JVM 参数添加到第二行的结尾：
  -Xmanagement:class=com.mercury.opal.capture.proxy.JRockitManagement

---

5 保存对启动脚本的更改，并使用经过修改的脚本重新启动应用程序服务器。

6 在启动脚本的输出中，查找 JRE Instrumenter 的输出（搜索 Xbootclasspath）。如果已在步骤 4 中向启动脚本添加了第二行（设置 JVM 参数），请将该行与 JRE Instrumenter 输出中的 JVM 参数进行对比。如果它们不相同，则使用 JRE Instrumenter 所提供的正确 JVM 参数更新启动脚本，然后重新启动应用程序服务器。如果未在步骤 4 中向启动脚本添加第二行 （设置 JVM 参数），则现在进行添加，并重新启动应用程序服务器。相关示例，请参阅步骤 4。

注意：如果无法找到 JRE Instrumenter 输出，或存在错误消息，则表明没有正确插桩 JRE。应检查启动脚本并解决任何出现的问题。

---

7 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\ DiagnosticsAgent\ log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩未成功，或者配置的 JVM 参数不正确。有关详细信息，请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

示例 8：配置 WebSphere

WebSphere 应用程序服务器是由 UNIX 环境中的 Shell 脚本或节点代理启动的。在 Windows 环境中，这些服务器可能会作为 Windows 服务安装，但同样可由 Windows 命令脚本启动。因此，建议您修改启动脚本以在 “自动显式”模式下运行 JRE Instrumenter 插桩由应用程序服务器使用的 JRE。

如果无法修改启动脚本，或 WebSphere 应用程序服务器正在 z/OS 上运行，则可以手动运行 JRE Instrumenter，或者如果 WebSphere 版本为 7.0 或更高版本，则可在自动隐式模式中使用 JRE Instrumenter。在后面的场景中，仅需要添加 JVM 参数。有关详细信息，请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。

另一方面，由于 WebSphere Application Server JVM 参数不由启动脚本控制，而是由配置文件控制，因此推荐您使用 “Integrated Solutions Console”（旧版本中也称为 “WebSphere Application Server Administrative Console”）添加在调用 Java 代理和使用已插桩 JRE 时需使用的 JVM 参数。

控制台的外观会因 WebSphere 的版本而异，因此，以下示例可能不完全符合您的 WebSphere 版本，但仍可提供所需信息。在控制台正确的位置中输入用于 Diagnostics 监控的所需参数。

提供了适用于 WebSphere 5.1/6.0 的步骤（如下所示），以及适用于 WebSphere 6.1 或更高版本的步骤（请参阅 “要配置 WebSphere 6.1 或更高版本，请执行以下步骤：”（第 203 页））。另请参阅 “要配置 WebSphere 6.1/7.0 服务器以收集 JMX 度量，请执行以下操作：”（第 209 页）。

要配置 WebSphere 5.1 或 6.0，请执行以下步骤：

1 找到用于启动 WebSphere 应用程序服务器的脚本，

例如， **<WAS 安装路径 >\bin\startServer.bat**

其中 **<WAS 安装路径 >** 是 WebSphere 安装目录的路径，例如 **C:\Program Files\IBM\WebSphere\AppServer**。

注意：在某些系统中，可能使用配置文件的 bin 目录中的 startServer.[bat|sh] 脚本来启动服务器。但在 **<WAS 安装目录 >/bin** 目录中，此脚本通常为调用 **startServer.[bat|sh]** 脚本的简单包装。

---

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 对于 WebSphere 5.1 或 6.0，请找到用于运行应用程序服务器启动程序的 java 命令行。以下是 **startServer.bat** 文件中的一个示例：

```
"%JAVA_HOME%\bin\java" -Dcmd.properties.file=%TMPJAVAPROPFILE%
%WAS_TRACE% %WAS_DEBUG% %CONSOLE_ENCODING% "%CLIENTSAS%"
"%CLIENTSSL%" %USER_INSTALL_PROP% "-Dwas.install.root=%WAS_HOME%"
com.ibm.ws.bootstrap.WSLauncher
com.ibm.ws.management.tools.WsServerLauncher "%CONFIG_ROOT%"
"%WAS_CELL%" "%WAS_NODE%" %* %WORKSPACE_ROOT_PROP%
```

在标识的 java 命令行之上添加一行，以调用 JRE Instrumenter。在此行中，需要指定用于存储插桩类的目录的名称。以下为 Windows 上的 WebSphere 6.0 的一个示例。应使用您所选的名称取代 **MyServer**。

```
"%JAVA_HOME%\bin\java" -jar
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer
```

4 保存更改，然后使用经过修改的脚本重新启动应用程序服务器。

5 在启动脚本的输出中，查找 JRE Instrumenter 的输出 （搜索 Xbootclasspath）。

以下是 Windows 上 WebSphere 6.0（或 5.1）的一个示例 JRE Instrumenter 输出 (-Xbootclasspath)（使用 **-f MyServer** 指定用于存储插桩类的目录。请参阅上述步骤 3）。应使用您所选的名称代替 MyServer。

```
-Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA
gent\DiagnosticsAgent\classes\boot
```

如果无法找到 JRE Instrumenter 输出，或存在错误消息，则表明没有正确插桩 JRE。应检查启动脚本，并解决问题。有关详细信息，请参阅“在“自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

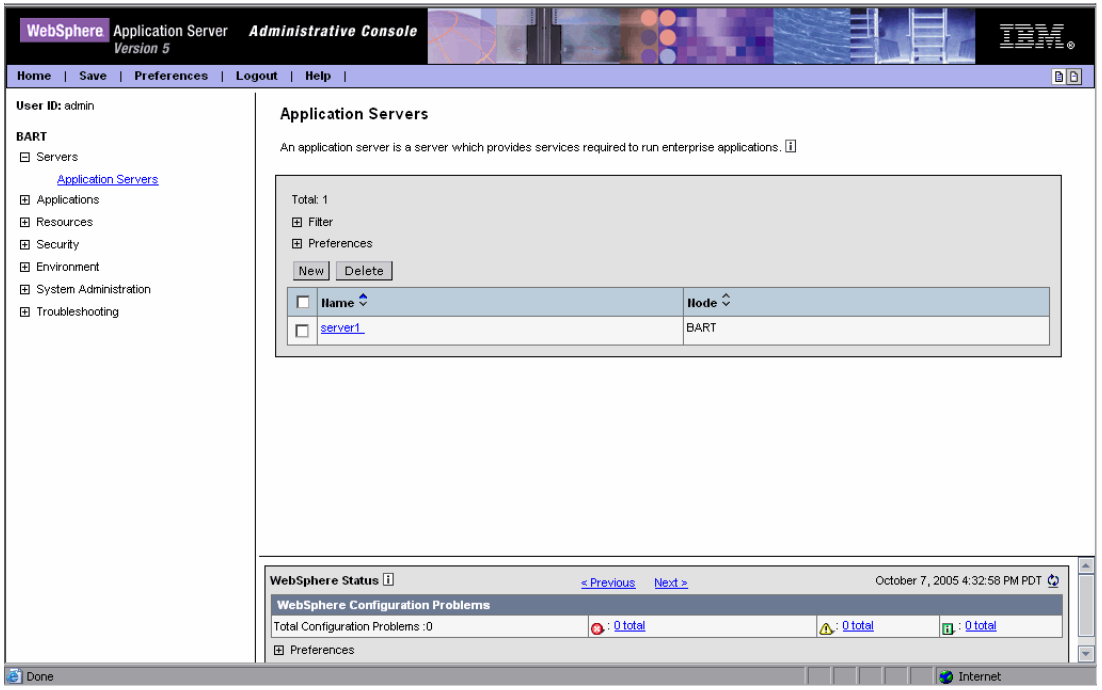
获得 JRE Instrumenter 输出之后，需将其添加到应用程序服务器 JVM 参数。

6 使用 Web 浏览器访问 WebSphere 应用程序服务器管理控制台，以便探测器监控应用程序服务器实例：

```
http://<App_Server_Host>:9090/admin
```

请用应用程序服务器主机的计算机名称替换 **< 应用程序服务器主机 >**，可能还需用正确的管理端口号 （如 9060、 9061 等）替换 9090。

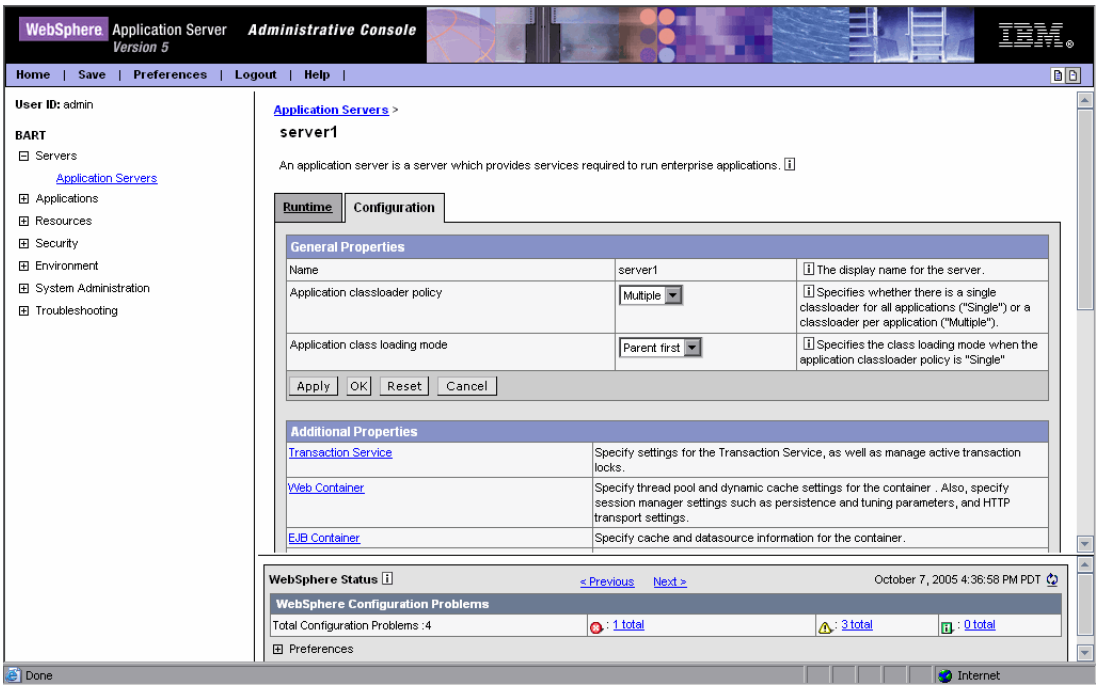
此时将打开 Websphere 应用程序服务器管理控制台。



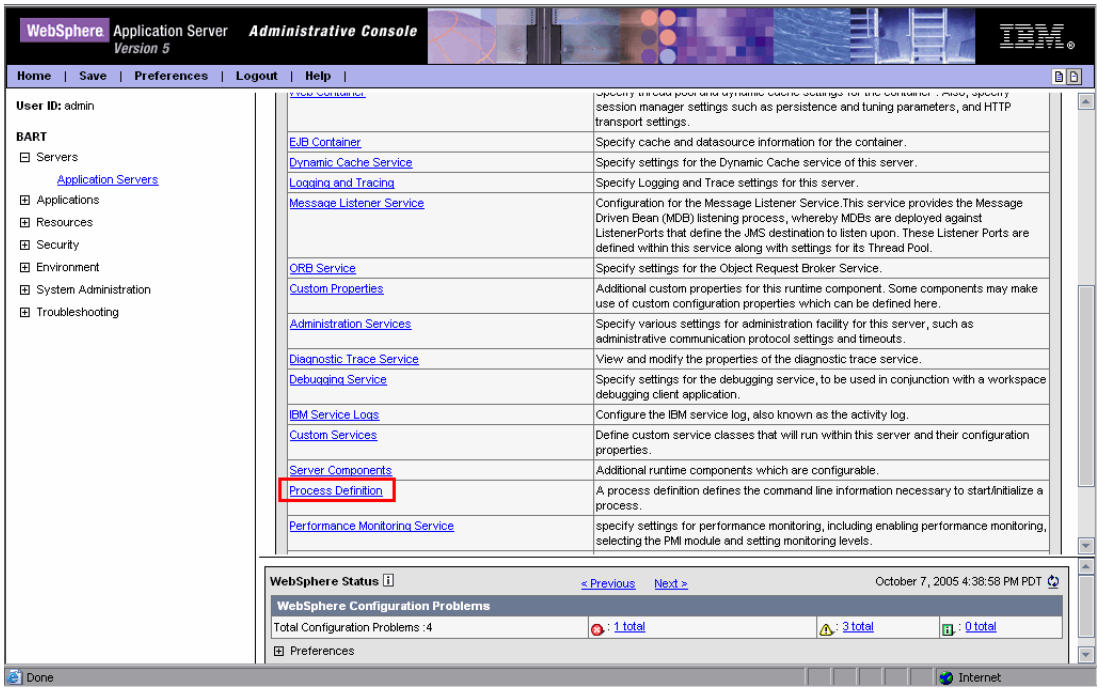
7 在左面板中，选择 “Servers” > “Application Servers”。

8 从右面板的应用程序服务器列表中，选择需要配置为受探测器监控的服务器的名称。

此时将显示所选应用程序服务器的 “Configuration” 选项卡。

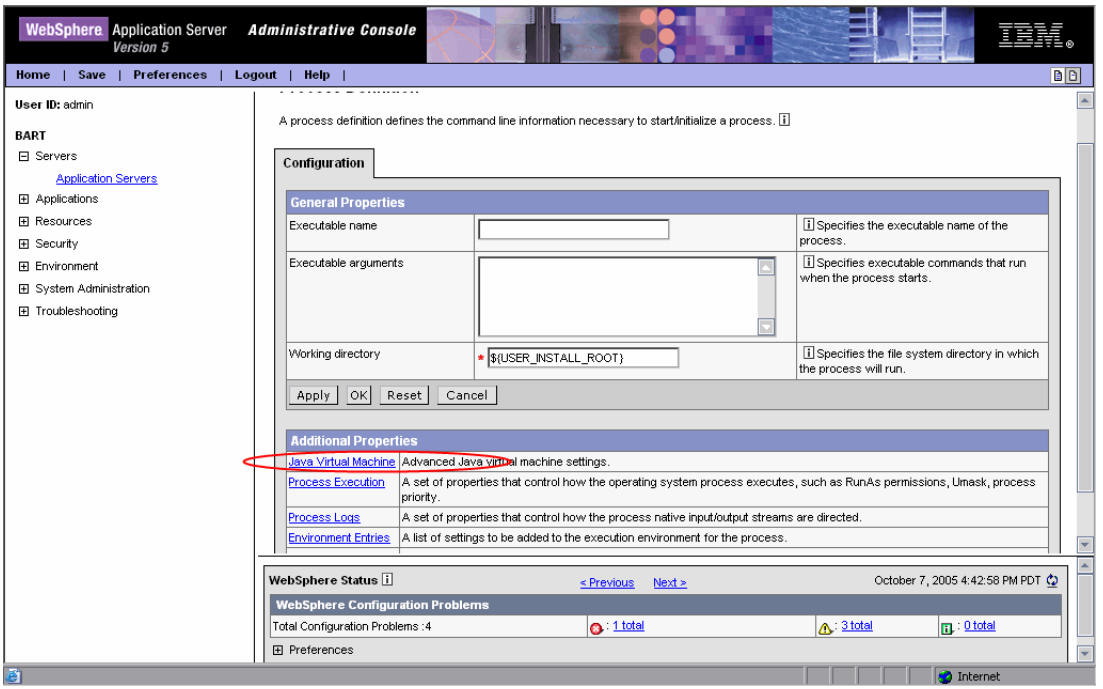


9 向下滚动“Configuration”选项卡，在“General Properties”栏中，查找“Process Definition”属性。



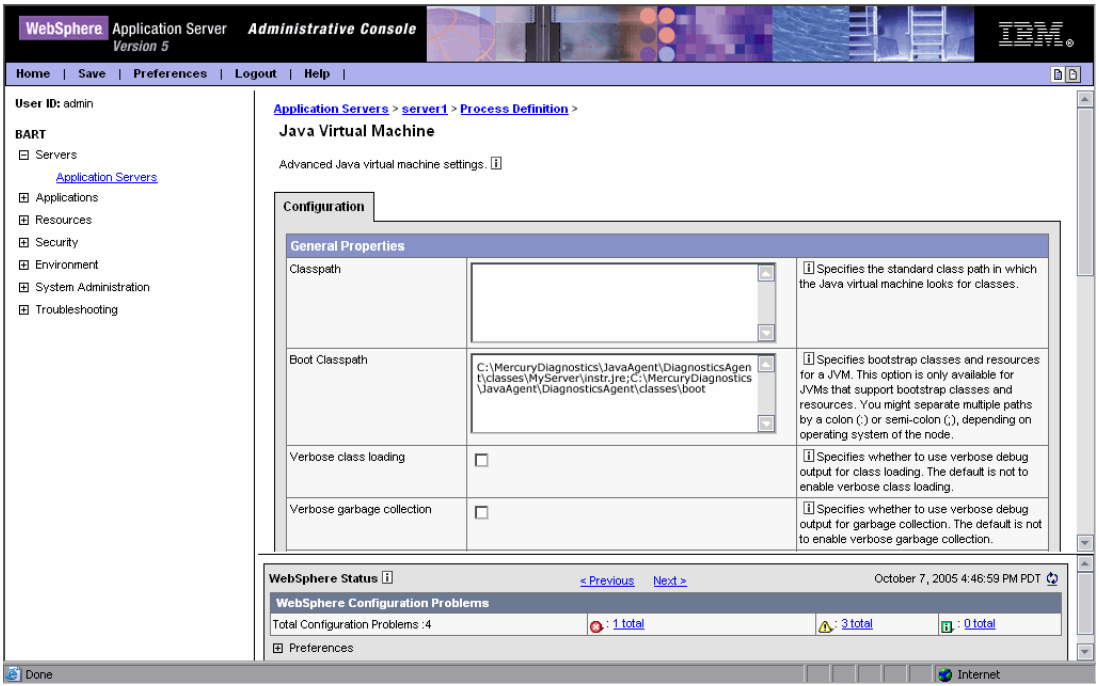
10 单击“Process Definition”。

11 向下滚动右面板，查找 “Java Virtual Machine”。



12 单击 “Java Virtual Machine”。

13 此时将显示 Java 虚拟机的 “Configuration” 选项卡。



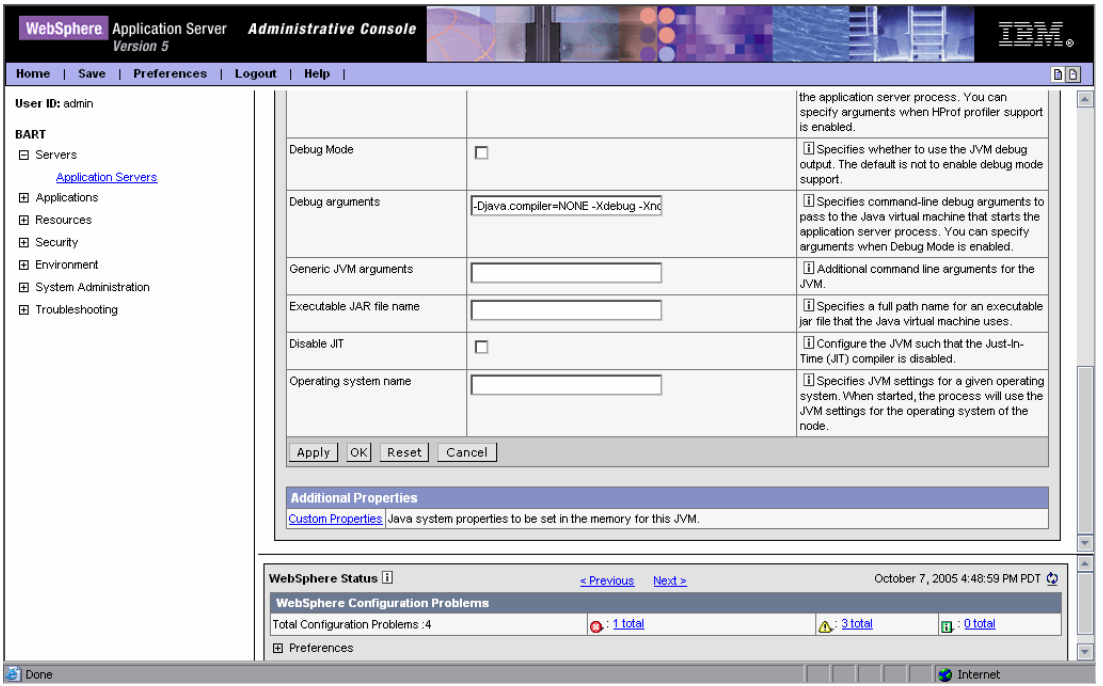
14 在 “Boot Classpath” 框中，输入由 JRE Instrumenter 提供的 JVM 参数的引导类路径（即 -Xbootclasspath/p: 参数之后的字符串）。JRE Instrumenter 的 JVM 参数输出已在之前的步骤 3 中生成。

以下是一个适用于 WebSphere 6.0 （或 5.1）的示例，其中使用 **-f MyServer** 指定用于存储插桩类的目录的名称：

```
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot
```

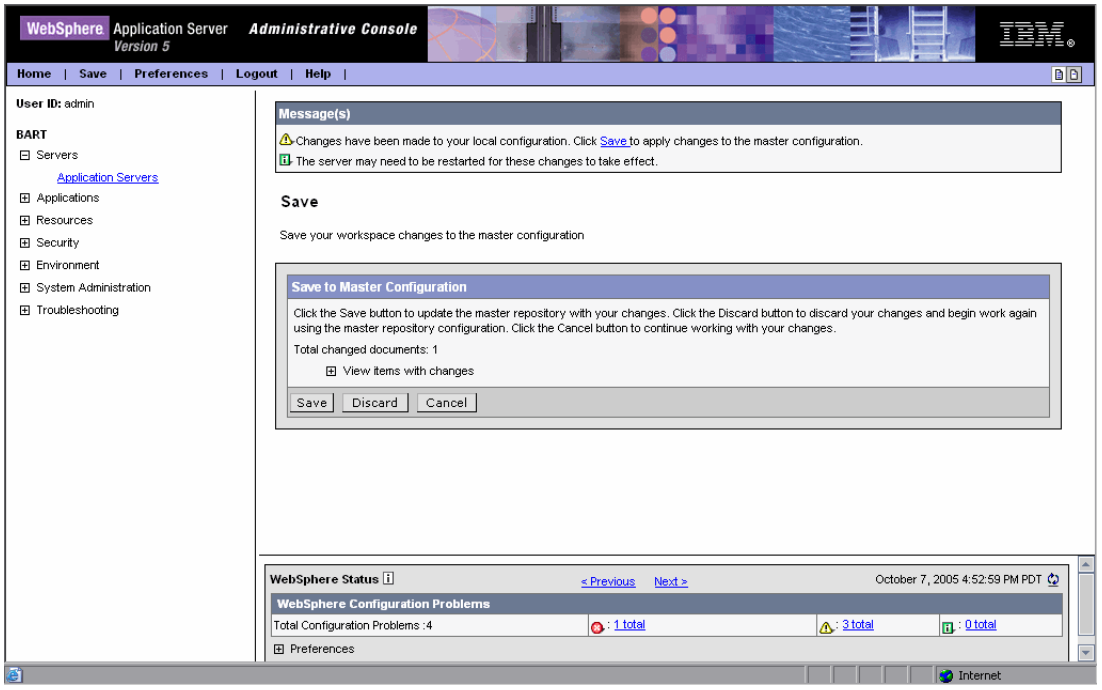
注意：在具有 32 位 WebSphere 6.0 JRE 的 64 位平台 （操作系统）上，如果在手动模式下使用了 JRE Instrumenter，则还需要向 “Generic JVM arguments” 框添加 **-Xj9**。

15 向下滚动到 “Configuration” 选项卡底部，直到看见命令按钮为止。



单击 “Apply”。

16 此时将出现一条消息，确认您的更改已应用。在“Save to Master Configuration”区域中，单击 “Save”。



17 单击“Save”，将更改应用到主配置。如果收到确认提示，请再次单击“Save”。

18 重新启动 WebSphere 应用程序服务器。

19 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 Java 代理未正确启动或您尚未运行 JRE Instrumenter，或者输入的 Java 参数 Xbootclasspath 不正确。有关详细信息，请参阅 “关于 JRE Instrumenter 以及各种要调用的选项”（第 217 页）。

要配置 WebSphere 6.1 或更高版本，请执行以下步骤：

1 找到用于启动 WebSphere 应用程序服务器的脚本，

例如， **<WAS 安装路径 >\bin\startServer.bat**

其中 **<WAS 安装路径 >** 是 WebSphere 安装目录的路径，例如 **C:\Program Files\IBM\WebSphere\AppServer**。

注意：在某些系统中，可能使用配置文件的 bin 目录中的 startServer.[bat|sh] 脚本来启动服务器。但在 **<WAS 安装目录 >/bin** 目录中，此脚本通常为调用 **startServer.[bat|sh]** 脚本的简单包装。

---

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 对于 WebSphere 6.1 或更高版本，请找到用于定义 JAVA_EXE 变量的代码块。

```
if exist "%JAVA_HOME%\bin\java.exe" (
   set JAVA_EXE="%JAVA_HOME%\bin\java"
) else (
   set JAVA_EXE="%JAVA_HOME%\jre\bin\java"
)
```

在以上代码块下方添加一行，以调用 JRE Instrumenter。在此行中，需要指定用于存储插桩类的目录的名称。以下为 Windows 上的 WebSphere 6.1 的一个示例。应使用您所选的名称取代 **MyServer**。

```
%JAVA_EXE% -jar
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f MyServer
```

4 保存更改，然后使用经过修改的脚本重新启动应用程序服务器。

5 在启动脚本的输出中，查找 JRE Instrumenter 的输出 （搜索 Xbootclasspath）。

WebSphere 6.1。以下是 Windows 上 WebSphere 6.1 的一个示例 JRE Instrumenter 输出 (-Xbootclasspath)（使用 **-f MyServer** 指定用于存储插桩类的目录。请参阅上述步骤 3）。应使用您所选的名称代替 MyServer。

```
-Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;C:\MercuryDiagnostics\JavaA
gent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

WebSphere 7.0 或 8.0。以下是 Windows 上 WebSphere 7.0（或 8.0）的一个示例 JRE Instrumenter 输出 (-Xbootclasspath)（使用 **-f MyServer** 指定用于存储插桩类的目录。请参阅上述步骤 3）。应使用您所选的名称代替 MyServer。

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

如果无法找到 JRE Instrumenter 输出，或存在错误消息，则表明没有正确插桩 JRE。应检查启动脚本，并解决问题。有关详细信息，请参阅“在“自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

获得 JRE Instrumenter 输出之后，需将其添加到应用程序服务器 JVM 参数。

6 打开 WebSphere 应用程序服务器管理控制台。例如：

http://< 应用程序服务器主机 >:9060/ibm/console

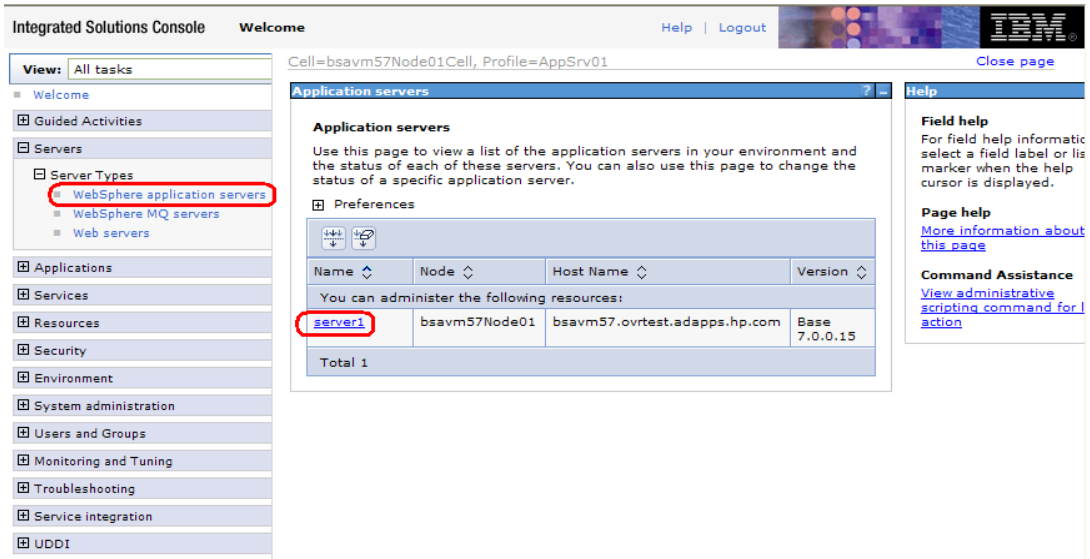
请使用应用程序服务器主机的计算机名称替换 **< 应用程序服务器主机 >**，并使用正确的管理端口号 （如 9060、 9061 等）替换 9060。

7 导航到 “Java 虚拟机”页面。例如：

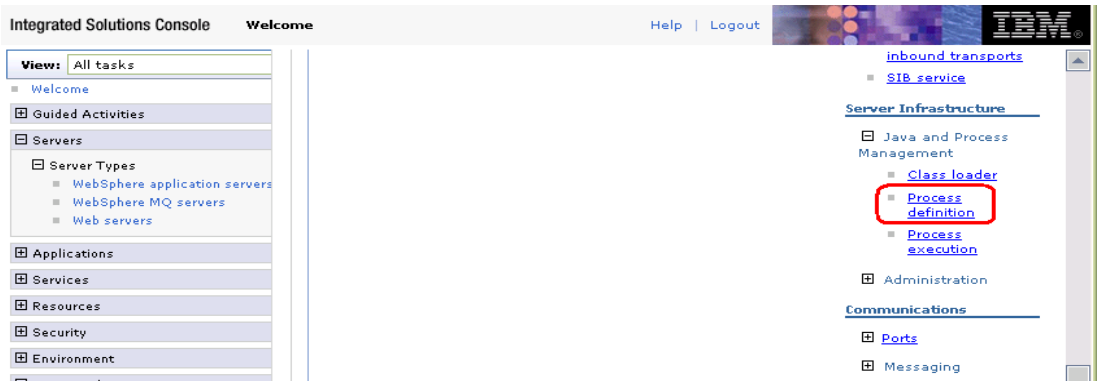
对于 WebSphere 6.1，导航到：“服务器” > “应用程序服务器”

对于 WebSphere 7.0，导航到：“Servers” > “Server Types” > “WebSphere Application servers”

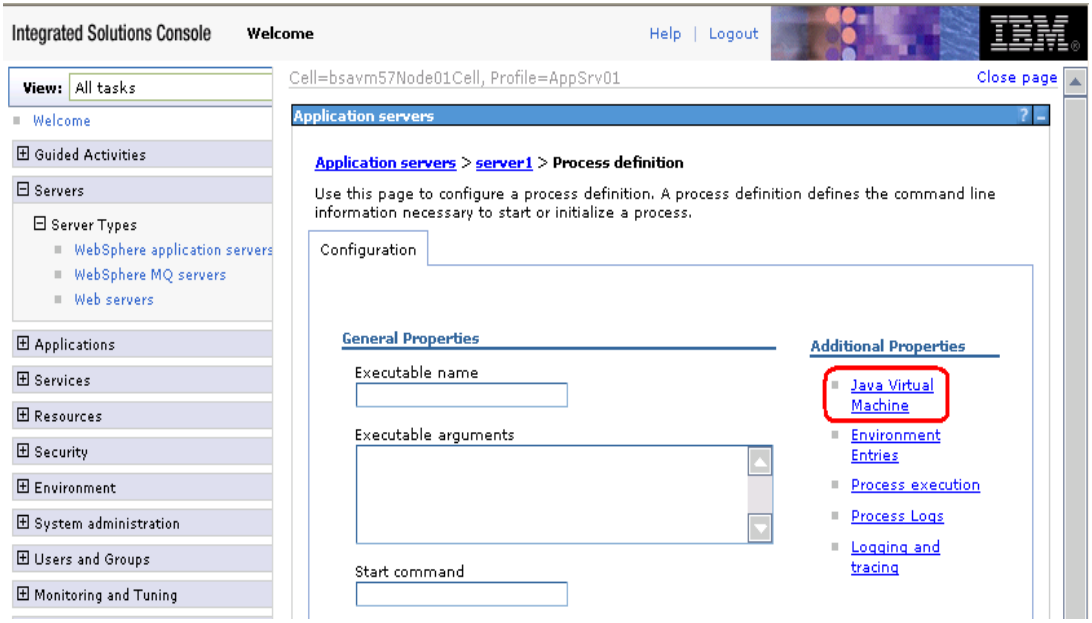
然后单击应用程序服务器实例名称 （如 **server1**）。



之后，在“Server Infrastructure”>“Java and Process Management”下，单击“Process Definition”>“Java Virtual Machine”。



然后，在“Additional Properties”下，单击“Java Virtual Machine”。



8 在“Java Virtual Machine”页面的“Generic JVM Arguments”框中，输入 JRE instrumenter 的 JVM 参数。JRE Instrumenter 的 JVM 参数输出已在之前的步骤 3 中生成。

以下是一个适用于 WebSphere 6.1 的示例，其中使用 **-f MyServer** 指定用于存储插桩类的目录的名称：

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre;
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

以下是没有修改启动脚本而是手动使用 JRE Instrumenter 插桩 JRE 的 WebSphere 6.1 的一个示例：

```
-Xbootclasspath/
p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\IBM\1.5.0\instr.jre;C:\MercuryDiagnostics\JavaA
gent\DiagnosticsAgent\classes\boot -Xshareclasses:none
```

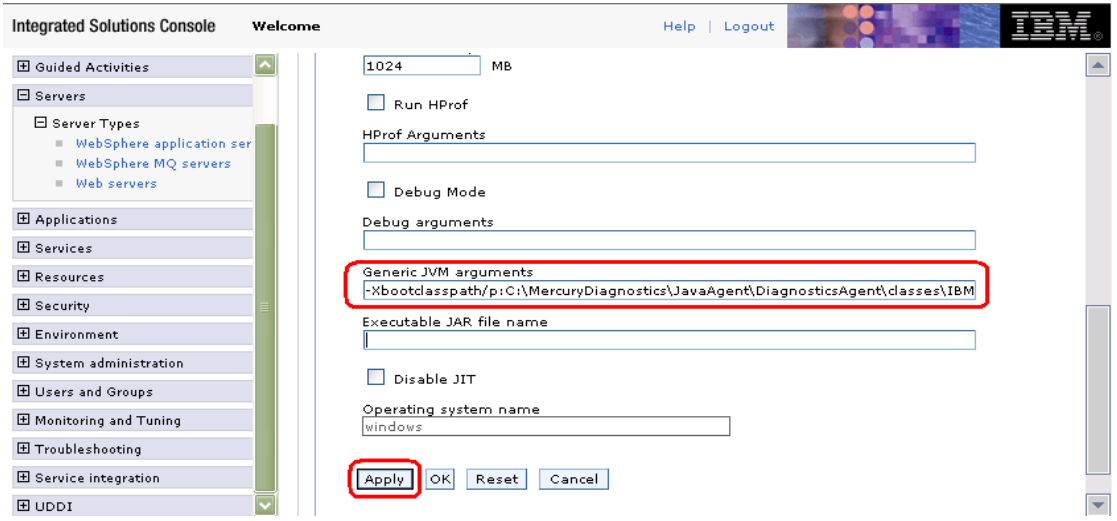
以下是一个适用于 WebSphere 7（或 8）的示例，在启动脚本中使用 **-f MyServer** 作为 JRE Instrumenter 的命令行选项：

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

以下是没有修改启动脚本或手动运行 JRE Instrumenter（在自动隐式模式中使用 JRE Instrumenter）的 WebSphere 7 （或 8）的一个示例：

```
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\server1\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar -Xshareclasses:none
```

9 应用并保存更改。



10 重新启动 WebSphere 应用程序服务器，

11 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/<探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明 Java 代理未正确启动或您尚未运行 JRE Instrumenter，或者输入的 Java 参数 Xbootclasspath 不正确。有关详细信息，请参阅 “关于 JRE Instrumenter 以及各种要调用的选项”（第 217 页）。

运行用于 WebSphere IDE 的 JRE Instrumenter

如果使用 WebSphere IDE，则必须手动运行 JRE Instrumenter 以确保为 WSAD IDE 使用正确的 Java 可执行文件。

WSAD IDE 有不同的 java.exe 可执行文件供您选择。您必须插桩用于运行 WebSphere 的可执行文件。

要插桩正确的 java.exe，请执行以下操作：

1. 确定要使用的 WebSphere 版本。
2. 确定合适的 java.exe 的位置。
3. 运行 JRE Instrumenter 并添加正确的 JVM。有关详细信息，请参阅“关于 JRE Instrumenter 以及各种要调用的选项”（第 217 页）。

配置 WebSphere 以收集 JMX 度量

您可能需要在 WebSphere 服务器上配置性能监控基础结构 (PMI) 服务，以便接收 JMX 度量。

重要信息：如果 Diagnostics 无法将应用程序服务器识别为 WebSphere 服务器，则必须启用 PMI，并将 Jar 文件添加到 server.policy 文件中。

要配置 WebSphere 6.1/7.0 服务器以收集 JMX 度量，请执行以下操作：

1. 打开 WebSphere 管理控制台。
2. 在控制台导航树中，选择 “服务器” > “应用程序服务器”。此时控制台会显示应用程序服务器列表。
3. 在 “应用程序服务器”列表中，单击要配置的应用程序服务器的名称，此时控制台会显示所选应用程序服务器的 “运行时”和 “配置”选项卡。
4. 单击 “配置”选项卡。
5. 在 “配置”选项卡中：

- 在 “性能”标题下，单击 “性能监控基础结构 (PMI)”。
- 在 “常规属性”标题下，选中 “启用性能监控基础结构 (PMI)”复选框。
- 在 “当前监控的统计信息集”标题下，选择 “已扩展”。

6 单击 “应用”或 “确定”。

7 如果在应用程序服务器上启用了 Java 2 Security，则打开服务器策略文件（**<WebSphere 6.x 安装目录 >/work/tools/ibm-6.0/websphere/appserver/profiles/default/properties/server.policy** 或 **<WebSphere 7.0 安装目录 >/AppServer/profiles/< 配置文件名称 >/properties/server.policy**），然后添加以下安全权限以启用 JMX 收集：

```
grant codeBase "file:/< 探测器安装目录 >/lib/probe-jmx.jar"
{ permission java.security.AllPermission; }

grant codeBase "file:/< 探测器安装目录 >/lib/probe-jmx-was6.jar" {
  permission java.security.AllPermission;
};
```

8 重新启动应用程序服务器。

示例 9：配置 WebMethods

此示例中讨论了两种类型的 WebMethods 服务器：

- WebMethods 集成服务器
- My webMethods 服务器

由于站点管理员会经常自定义 WebMethods 提供的启动脚本，我们无法提供适用于所有情况的详细配置说明。因此，以下部分通过 WebMethods 集成服务器和 My WebMethods 服务器的特定示例提供了常规说明。站点管理员应使用这些说明来指导您在自定义环境中进行相关更改。

要配置 WebMethods 集成服务器：

WebMethods 集成服务器通过 Shell 或命令脚本启动。因此，建议修改启动脚本以插桩服务器。

1 找到用于启动 WebMethods 集成服务器的启动脚本。基于服务器的启动方式，有两个选项：

...\IntegrationServer\bin\server.bat

...\profiles\IS\bin\runtime.bat

2 对启动脚本进行备份，并使用编辑器打开此启动脚本。

3 按照以下所述更新文件。

a 对于 **server.bat** 文件，找到用于启动服务器的部分：

```
if "1%1"=="1-service" (
        if exist LOCKFILE del LOCKFILE
        "%JAVA_EXEC%" -classpath %IS_PROXY_JAR%
com.wm.app.server.CustomServiceUpdater -isdir "%IS_DIR%" -wrapperdir
"%IS_DIR%\..\profiles\IS\configuration" -binpath "%PATH%" -jvmargs
"%SERVER_VM_OPT% %JAVA2_MEMSET% %JAVA_OPTS%" -progargs
%3#%4#%5#%6#%7#%8#%9
        goto :EOF
)

call "%PROFILES_DIR%\bin\start_runtime.bat" %1 %2 %3 %4 %5 %6 %7 %8 %9
```

并直接在此部分之上添加以下内容：

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

第一行调用 java 命令以运行 JRE Instrumenter ；第二行添加所需的 JVM 参数。如果不确定要为第二行使用哪些 JVM 参数，则可以稍后在步骤 5 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以上示例中的 **MyServer**。

b 对于 **runtime.bat** 文件，找到用于启动服务器的部分：

```
%JAVA_RUN% -Xbootclasspath/a:"%OSGI_CLASSPATH%" %JAVA_OPTS%
%JAVA_SYSPROPS% -cp "%OSGI_FRAMEWORK_JAR%"
org.eclipse.equinox.launcher.Main -configuration %OSGI_CONFIGURATION_AREA%
%CMD_ARGS%
goto end_start_cmd
```

直接在此部分之上添加两行，如此示例所示：

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

第一行调用 java 命令以运行 JRE Instrumenter ；第二行添加所需的 JVM 参数。如果不确定要为第二行使用哪些 JVM 参数，则可以稍后在步骤 5 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以上示例中的 **MyServer**。

4 保存对启动脚本的更改，并使用经过修改的脚本重新启动应用程序服务器。

5 在启动脚本的输出中，查找 JRE Instrumenter 的输出（搜索 Xbootclasspath）。如果已在步骤 3 中向启动脚本添加了第二行（设置 JVM 参数），请将该行与 JRE Instrumenter 输出中的 JVM 参数进行对比。如果它们不相同，则使用 JRE Instrumenter 所提供的正确 JVM 参数更新启动脚本，然后重新启动应用程序服务器。如果未在步骤 3 中向启动脚本添加第二行 （设置 JVM 参数），则现在进行添加，并重新启动应用程序服务器。

注意：如果无法找到 JRE Instrumenter 输出，或存在错误消息，则表明没有正确插桩 JRE。应检查启动脚本并解决任何问题。

6 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩未成功，或者配置的 JVM 参数不正确。有关详细信息，请参阅 “在“自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

要配置 My WebMethods 服务器启动脚本，请执行以下操作：

My WebMethods 服务器由脚本或包装配置启动。因此，可按照此示例中所述修改启动脚本，或按照下一步骤中的描述编辑包装配置以插桩服务器。

1 找到用于启动 WebMethods 服务器的启动脚本。脚本文件为：...\MWS\server\bin\mws.bat。

2 备份此文件，并使用编辑器打开它。

3 按照以下描述更新此文件。

对于 **mws.bat** 文件，找到 RUN_CMD 的定义，如以下示例中突出显示的部分所示：

```
set JAVA_OPTIONS=%JAVA_OPTIONS% -Dserver.name=%SERVER_NAME%
-Djava.awt.headless=true
set PARAMS=
set MAIN_CLASS=com.webmethods.portal.system.PortalSystem
set RUN_CMD=%JAVA% -cp %CLASSPATH% %JAVA_ARGS% %JAVA_OPTIONS%
%ACTION_PARAMS% -Dmain.class=%MAIN_CLASS%
7 %8 %9
```

在此部分之上添加两行，如以下示例所示：

```
%JAVA_HOME%\bin\java -jar C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\jreinstrumenter.jar -f
MyServer

set JAVA_OPTIONS=%JAVA_OPTIONS%
-Xbootclasspath/p:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\MyServer\instr.jre
-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

第一行调用 java 命令以运行 JRE Instrumenter；第二行添加所需的 JVM 参数。如果不确定要为第二行使用哪些 JVM 参数，则可以稍后在步骤 5 中添加。在这两行中，请输入用于存储自动插桩的 JRE 类的目录的名称。应使用您所选的名称取代以上示例中的 **MyServer**。

4 保存对启动脚本的更改，并使用经过修改的脚本重新启动应用程序服务器。

5 在启动脚本的输出中，查找 JRE Instrumenter 的输出（搜索 Xbootclasspath）。如果已在步骤 3 中向启动脚本添加了第二行（设置 JVM 参数），请将该行与 JRE Instrumenter 输出中的 JVM 参数进行对比。如果它们不相同，则使用 JRE Instrumenter 所提供的正确 JVM 参数更新启动脚本，然后重新启动应用程序服务器。如果未在步骤 3 中向启动脚本添加第二行 （设置 JVM 参数），则现在进行添加，并重新启动应用程序服务器。

注意：如果无法找到 JRE Instrumenter 输出，或存在错误消息，则表明没有正确插桩 JRE。应检查启动脚本并解决任何问题。

---

6 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >\DiagnosticsAgent\log\< 探测器 ID>\probe.log** 文件中的条目。如果文件中没有条目，则表明 JRE 插桩未成功，或者配置的 JVM 参数不正确。有关详细信息，请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

要配置 My WebMethods 服务器配置包装，请执行以下操作：

My WebMethods 服务器由脚本或包装配置启动。因此，可修改启动脚本或编辑包装配置文件 （如本示例所示），以插桩服务器。

1 找到用于启动 My WebMethods 服务器的配置包装。配置文件为：
...MWS\server\< 服务器名称 >\config\wrapper.conf

2 备份此文件，并使用编辑器打开它。

3 按照以下描述更新此文件。

对于 **wrapper.conf** 文件，添加以下内容 （根据您的配置文件，更改数字 270 和 280）：

```
wrapper.java.additional.270=-Xbootclasspath/p:
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\classes\auto\MyServer\instr.jre

wrapper.java.additional.280=-javaagent:C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\lib\probeagent.jar
```

第一行调用 java 命令以运行 JRE Instrumenter ；第二行添加所需的 JVM 参数。首次调用时，第二个参数 (-Xbootclasspath) 可使应用程序服务器 JRE 得以插桩。在 -Xbootclasspath 参数中，输入一个名称以指定用于存储所插桩类的目录的名称。应使用您所选的名称取代以上示例中的 **MyServer**。

4 保存对配置包装的更改，并使用经过修改的包装重新启动应用程序服务器。

将调用 Java 代理，且 Java 代理会隐式运行 JRE Instrumenter 以插桩 JRE。

5 要验证探测器的配置是否正确，请检查 **<Java 代理安装目录 >/DiagnosticsAgent/log/< 探测器 ID>/probe.log** 文件中的条目。如果文件中没有条目，则表明未正确设置 JVM 参数。

6 或者，重新启动应用程序服务器，以便其使用已插桩的 JRE。

重要信息：如果将来要在用配置包装启动时更新 My WebMethods 服务器所使用的 JRE，则在重新启动 My WebMethods 服务器之前，必须删除 **<Java 代理安装目录 >/DiagnosticsAgent/classes/auto/MyServer** 目录（使用您的目录名称代替 MyServer），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。有关所用插桩模式的常规信息，请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。

关于 JRE Instrumenter 以及各种要调用的选项

JRE Instrumenter 是一种插桩 JRE 的实用程序，可帮助 Java 代理提供一些高级功能，如正在申请专利的回收泄漏定位 (CLP) 功能。JRE Instrumenter 不会以任何方式修改已安装的 JRE，而是将已测得的类的副本置于 **<Java 代理安装目录 >/DiagnosticsAgent/classes** 目录下。您可以使用 JRE Instrumenter 插桩多个已安装到系统上的 JRE。

JRE Instrumenter 插桩应用程序服务器 JVM 以及在其上运行的应用程序所使用的一些标准 Java 类。同时还提供必须在应用程序服务器启动时使用的 JVM 参数，以便应用程序服务器能够使用插桩的类。

通过使用不同命令行选项，可以以三种不同方式调用和使用 JRE Instrumenter，每一种方式都有其优点和限制。根据应用程序服务器的特性，可使用以下方法之一 （有关示例，请参阅 “关于配置应用程序服务器的示例”（第 163 页））。

- **自动显式模式。**如果应用程序服务器由脚本或可由脚本启动，建议使用此模式。要使用此模式，需向应用程序服务器启动脚本添加行，从而显式地以非交互方式运行 JRE Instrumenter 来插桩 JRE。此脚本将使用新插桩的 JRE（以及其他参数）继续启动应用程序服务器 JVM。请参阅 “在 “自动显式”模式下使用 JRE Instrumenter”（第 219 页）。

- **自动隐式模式。**通过这种模式，您无需显式运行 JRE Instrumenter，只需修改应用程序服务器 JVM 参数以调用 Java 代理，并根据需要要求 Java 代理运行 JRE Instrumenter。首次使用 Java 代理时，它将隐式运行 JRE Instrumenter 以插桩 JRE。但是，此插桩的 JRE 不会在首次使用，应用程序服务器将使用非插桩的 JRE。在下次启动应用程序服务器时，才会使用插桩的 JRE。因此，要想使用 Java 代理的完整监控功能，您需要在启用 Java 代理之后重新启动应用程序服务器两次。请参阅 “在自动隐式模式下使用 JRE Instrumenter”（第 222 页）。
- **手动模式。**使用这种模式时，您需要在 Java 代理安装结束时或稍后交互式地手动运行 JRE Instrumenter，以插桩 JRE。然后，根据 JRE Instrumenter 所提供的参数修改应用程序服务器 JVM 参数。这是 HP Diagnostics 早期版本中 JRE Instrumenter 的运行方式。请参阅 “在手动模式下使用 JRE Instrumenter”（第 224 页）。

如果更新了 JRE（如应用应用程序服务器修补程序），或更新了 Java 代理，则可能需要重新插桩 JRE。在每种模式中都将讨论这个问题。

下表总结了四种执行插桩的不同方法的要求：

		推荐插桩（使用 JRE Instrumenter）		
	基本插桩	**在自动显式模式中**	**在自动隐式模式中**	**在手动模式中**
所需的最低 JRE 版本	1.5	1.4	1.5	1.4
要求应用程序服务器由脚本启动	否	是	否	否

		推荐插桩 （使用 JRE Instrumenter）		
	基本插桩	在自动显式模式中	在自动隐式模式中	在手动模式中
要求了解安装 JRE 的位置	否	否	否	是
要求手动运行 JRE Instrumenter	否	否	否	是
要求了解可配置 JVM 参数的位置	是 *	是 *	是 *	是 *
要求启用 Java 代理之后重新启动应用程序服务器	是，一次	是，一次或两次	是，两次	是，一次
要求升级 / 修补 JRE 之后进行维护	否	否	是	是

* 如果找不到可定义 JRE 调用参数的位置，仍然可以选择使用环境变量 （如 **JAVA_OPTIONS**）完成此操作。

在 “自动显式”模式下使用 JRE Instrumenter

应用程序服务器 （如 WebLogic 和 JBoss 应用程序服务器）由脚本启动时，建议在自动显式模式下使用 JRE Instrumenter。如果 WebSphere 应用程序服务器由脚本或可由脚本启动，同样建议在自动显式模式中使用 JRE Instrumenter - 这种情况适用于除 z/OS 以外的大多数平台。如果 Tomcat 没有安装为 Windows 服务，也建议在自动显式模式中使用 JRE Instrumenter。

要使用 “自动显式”模式，需要完成两个任务：

- 使用应用程序服务器所使用的相同的 JRE 修改用以运行 JRE Instrumenter 的应用程序服务器启动脚本。JRE Instrumenter 的输出将为您提供需在下一任务中使用的 JVM 参数。
- 配置 JRE Instrumenter 的输出中的应用程序服务器 JVM 参数。

注意：在更改任何配置之前，请确保您了解启动脚本的结构、属性值的设置方法，以及环境变量的使用方法。另外，在进行更改之前，请始终对计划修改的所有文件进行备份。

在修改应用程序服务器启动脚本过程中，需首先标识在其中调用 JRE 以启动应用程序服务器 JVM 的行。然后在此行的正上方添加类似如下所示的行，以使用应用程序服务器所用的相同 JRE 调用 JRE Instrumenter：

```
<java_command> -jar <JavaAgent_install_dir>/DiagnosticsAgent/lib/jreinstrumenter.jar
-f <pathname>
```

其中 **<java_command>** 必须与用于启动应用程序服务器 JVM 的 java 命令**完全**相同，因为此 JRE 即由 JRE Instrumenter 插桩的 JRE。通过复制启动应用程序服务器 JVM 的行的开始部分，通常可获得 java 命令。

以下表格显示一些常用应用程序服务器的原始启动脚本所使用的 java 命令。（请注意，此表格仅用于提示；您的应用程序服务器启动脚本可能使用其他 java 命令。）

应用程序服务器	Shell 脚本 (.sh)	Windows 命令脚本 （.bat 或 .cmd）
JBoss	"$JAVA"	"%JAVA%"
Tomcat	${_RUNJAVA}	%_RUNJAVA%.
WebLogic	${JAVA_HOME}/bin/java	%JAVA_HOME%\bin\java
WebSphere	${JAVA_EXE}	%JAVA_EXE%

<Java 代理安装目录 > 指示安装 Java 代理所在的目录。

< 路径名 > 必须为对应的路径名。JRE Instrumenter 会将已插桩的类置于 **<Java 代理安装目录 >/DiagnosticsAgent/classes/< 路径名 >/instr.jre** 目录中。如果要与 Diagnostics 一起运行多个应用程序服务器，则应为每个应用程序服务器指定唯一的 < 路径名 >（如探测器名称），以便 JRE Instrumenter 的多个实例不会相互干扰。有关详细信息，另请参阅“配置应用程序服务器上多个 Java 进程的监控”（第 231 页）。

如上所述将行添加到启动脚本之后，每次使用此启动脚本启动应用程序服务器时，都将调用 JRE Instrumenter 以插桩当前使用的 JRE。此外，还将打印下一个任务中应使用的 JVM 参数。通常可从运行启动脚本的输出中找到 JRE Instrumenter 的输出。

以下是 JRE Instrumenter 插桩典型 JRE 版本 5.0 或更高版本的输出示例：

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/<pathname>/instr.jre
-javaagent:<JavaAgent_install_dir>/DiagnosticsAgent/lib/probeagent.jar
```

以下是 JRE Instrumenter 插桩典型 JRE 版本 1.4.x 的输出示例：

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/<pathname>/instr.jre;
<JavaAgent_install_dir>/DiagnosticsAgent/classes/boot
```

使用自动显式 JRE 插桩的第二个任务是按照 JRE Instrumenter 的输出来修改应用程序服务器 JVM 参数。在很多情况下，只需在启动脚本中修改 java 命令行选项，使其包含由 JRE Instrumenter 提供的 JVM 参数。但是，在某些情况下（如对于 WebSphere 应用程序服务器），可能需要修改配置文件或使用管理控制台以添加这些 JVM 参数。

注意：要获取 JRE Instrumenter 的输出，需按照第一个任务中的描述修改启动脚本并重新启动应用程序服务器。然后，在更改应用程序服务器 JVM 参数后，需再次重新启动应用程序服务器 （这导致用户需重新启动应用程序服务器两次）。但是，对于大多数 JRE 来说，由 JRE Instrumenter 提供的实际 JVM 参数将与上述示例中提供的参数相同，或包含这些参数。因此，可以安全地添加这些“默认”JVM 参数，即使在运行经修改的脚本之前也是如此。这是在关于特定应用程序服务器的说明中使用的方法。有关如何使用自动显式模式进行配置的详细信息，请参阅应用程序服务器 （JBoss、 WebLogic、 WebSphere、 Tomcat）的对应示例。

此外，也可以将 JRE Instrumenter 的输出重定向 （或发送）到 java 命令行选项，或者从其他源获取 JVM 参数以避免重新启动两次。

在自动隐式模式下使用 JRE Instrumenter

如果应用程序服务器无法由脚本启动，如 GlassFish、 NetWeaver、作为一个 Windows 服务安装的 Tomcat （无脚本）、安装在 z/OS 上的 WebSphere、 TIBCO ActiveMatrix 和 BusinessWorks，建议在自动隐式模式下使用 JRE Instrumenter。

要使用此模式，无需显式调用 JRE Instrumenter ； Java 代理将隐式调用它。您只需配置应用程序服务器 JVM 参数以调用 Java 代理，然后当 Java 代理发现 JVM 引导类路径包含指向与以下模式匹配的位置的路径时， Java 代理将进入自动插桩模式以创建插桩的类，并生成包含已插桩类副本的指定目录。

```
<JavaAgent_install_dir>/DiagnosticsAgent /classes/auto/<name>/instr.jre
```

例如，如果添加以下 JVM 参数：

```
-Xbootclasspath/p:<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/ServerOne/instr.jre
-javaagent: <JavaAgent_install_dir>/DiagnosticsAgent /lib/probeagent.jar
```

在首次执行应用程序服务器时， **<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/ServerOne/instr.jre** 目录甚至可能不存在。Java 代理将创建并使用已插桩的类填充指定的目录。 Java 代理将使用在其上运行它的正确的 （未插桩） JRE。

应用程序服务器的首次执行不会使用已插桩的 JRE，但所有后续执行都将使用首次运行中所准备的已插桩的类。

重要信息：如果更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），或更新 Java 代理，则在重新启动应用程序服务器之前，必须删除 **<JavaAgent_install_dir>/DiagnosticsAgent/classes/auto/ServerOne** 目录（使用您的目录名称代替 ServerOne），以便插桩新的 JRE。否则，应用程序服务器可能不会启动。此外，如要 Java 代理重新插桩 JRE，还可手动删除此目录。

在手动模式下使用 JRE Instrumenter

可手动运行 JRE Instrumenter 并将提供的 JVM 参数复制到应用程序服务器启动设置中。对于 Oracle 应用程序服务器，建议在手动模式下使用 JRE Instrumenter。

JRE Instrumenter 可执行以下功能：

- 标识可插桩的 JRE。
- 在指定的目录中搜索其他 JRE。
- 插桩您指定的 JRE，并提供必须添加到 JRE 启动脚本的参数，以指向所插桩的类的位置。
- 在 Windows 或 UNIX 环境中使用图形界面或控制台模式运行 Instrumenter 时， Instrumenter 会将已插桩的类放在 **<Java 代理安装目录 >/DiagnosticsAgent/classes/<JRE 供应商 >/<JRE 版本 >** 目录下的文件夹中。

重要信息：如果更新应用程序服务器所使用的 JRE （如应用应用程序服务器修补程序），或者更新 Java 代理，则必须重新运行 JRE Instrumenter 以插桩新的 JRE 并相应更改 JVM 参数。否则，应用程序服务器可能不会启动。

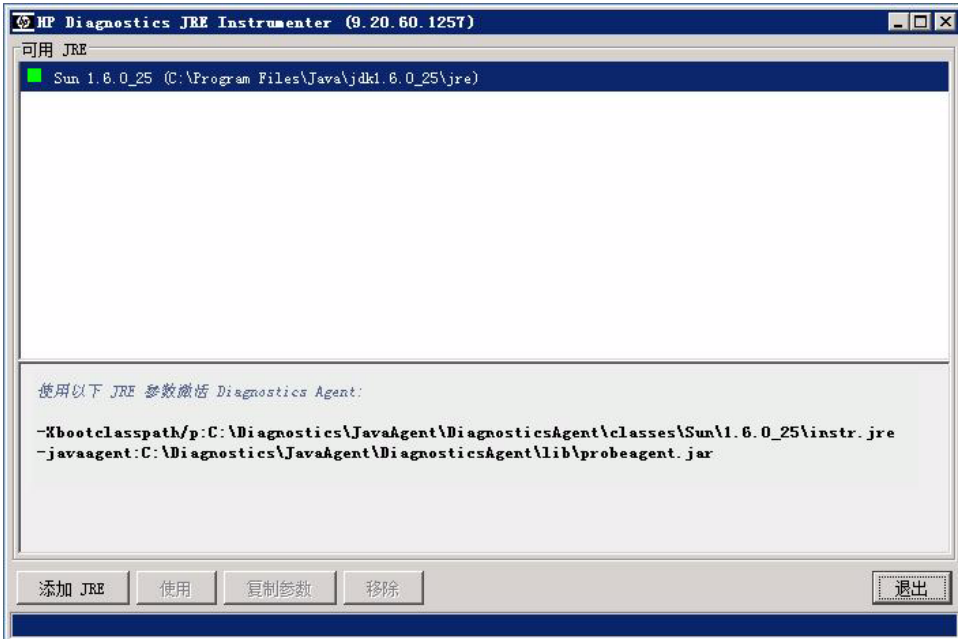
在 UI 模式下运行 JRE Instrumenter

在不使用任何选项的情况下运行 JRE Instrumenter 时， Instrumenter 将显示其图形用户界面的对话框。

要在 Windows 系统上启动 JRE Instrumenter 实用程序，请运行 **< 探测器安装目录 >\bin\jreinstrumenter.cmd** 目录。

要在 Windows 系统上启动 JRE Instrumenter 实用程序，请运行 **< 探测器安装目录 >\bin\jreinstrumenter.cmd** 目录。

Instrumenter 将列出它搜寻出的可插桩的 JVM。这些 JVM 的名称前面均带有一个绿色方块。

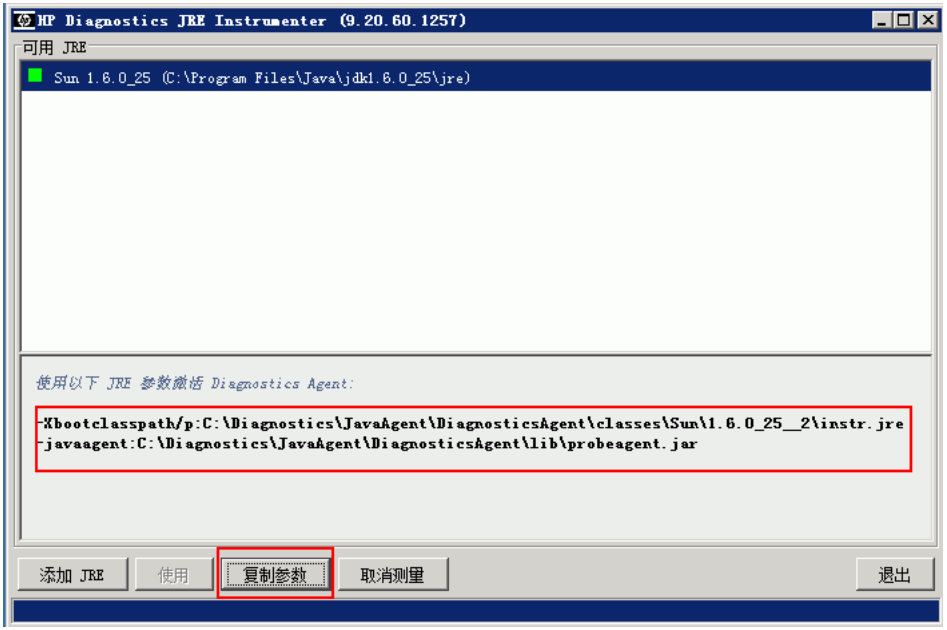


如果对话框中未列出“JRE 目录”，则单击“添加 JRE”按钮，浏览到 JRE。导航至要在其中开始搜索 JVM 的目录位置，然后选择要在其中开始搜索的文件，并单击“从此处搜索”。Instrumenter 将进行搜索，然后在“可用 JRE”列表中列出所找到的 JVM。

选择要插桩的 JRE，然后单击“使用”。

JRE Instrumenter 会插桩所选 JVM 的部分类，并将已插桩的类置于 **< 探测器安装目录 >/classes** 目录下的文件夹中。同时还会在“可用的 JRE”列表下方的框中显示 JVM 参数，当应用程序服务器启动时必须使用此参数。

JRE Instrumenter 在插桩 JRE 时还会创建 JVM 参数，必须将此参数包含在应用程序服务器的启动脚本中，以引导应用程序使用已插桩的类。从 “可用的 JVM” 列表中选择已插桩的 JVM 时， JVM 参数便显示在列表下方的框中。



单击 “复制参数”，将对应的参数放到剪贴板上。将 JVM 参数复制到剪贴板后，便可在配置应用程序服务器时使用 JVM 参数以激活由 Java 代理提供的监控。

重要信息：之后在配置应用程序服务器时，需使用剪贴板内容，因此请注意不要覆盖这些内容。

单击 “退出”关闭 JRE Instrumenter 窗口，并继续配置应用程序服务器 JVM 参数。

有关如何将 JVM 参数插入应用程序服务器启动脚本的常规信息，请参阅“将 JVM 参数包含在应用程序服务器启动脚本中”（第 228 页）。有关如何将 JVM 参数插入各种应用程序服务器（如 JBoss、WebLogic 和 Tomcat）启动脚本的特定示例，请参阅 “关于配置应用程序服务器的示例”（第 163 页）。

在控制台模式下运行 JRE Instrumenter

打开 **< 探测器安装目录 >\bin**，找到 JRE Instrumenter 可执行文件。运行以下命令：

```
./jreinstrumenter.sh -console
```

在 Instrumenter 运行时，将显示可用处理选项的列表。下表提供各处理选项的文档：

Instrumenter 函数	**描述**
jreinstrumenter -l	显示 JRE Instrumenter 已知的 JVM 的列表。将显示 JVM 供应商、JRE 版本和 JRE 的所在位置。
jreinstrumenter -i <jre 目录 >	选择要插桩的特定目录中的 JRE。将其中的 <jre 目录 > 替换为在其中找到您从“可用的 JVM”列表选择的 JRE 的文件夹的路径。 此命令将指示 JRE Instrumenter 为所选 JVM 插桩类，并将已插桩的类放在 **< 探测器安装目录 > /classes/ <JVM 供应商 >/<JRE 版本 >** 目录下的文件夹中。

Instrumenter 函数	描述
jreinstrumenter -a < 目录 >	在特定目录中搜索 JVM，并将找到的所有 JVM 添加到 JRE Instrumenter 已知的 JVM 列表中。将其中的 < 目录 > 替换为希望 Instrumenter 开始搜索的位置的路径。 Instrumenter 将搜索指定位置的目录，包括所有目录及其子目录。搜索完成后将显示更新后的“可用的 JVM”列表。

从 JRE Instrumenter 的输出复制 JVM 参数后，可将其粘贴到在应用程序启动时能够找到这些内容的位置，以便激活由 Java 代理执行的监控。

退出 JRE Instrumenter，然后继续配置应用程序服务器 JVM 参数。

有关如何将 JVM 参数插入应用程序服务器启动脚本的常规信息，请参阅“将 JVM 参数包含在应用程序服务器启动脚本中”（第 228 页）。有关如何将 JVM 参数插入各种应用程序服务器（如 JBoss、WebLogic 和 Tomcat）启动脚本的特定示例，请参阅 “关于配置应用程序服务器的示例”（第 163 页）。

将 JVM 参数包含在应用程序服务器启动脚本中

JRE Instrumenter 在插桩 JVM 时还会创建 JVM 参数，必须将此参数包含在应用程序服务器的启动脚本中，以引导应用程序使用已插桩的类。Instrumenter 插桩完 JVM 后会显示 JVM 参数。

将 JVM 参数复制到剪贴板，并将其粘贴到应用程序服务器启动时可获取的位置。下面提供了常规说明。

有关如何为应用程序服务器（如 WebLogic、 WebSphere、 JBoss 和其他应用程序服务器）插入 JVM 参数的特定示例，请参阅 “关于配置应用程序服务器的示例”（第 163 页）。

要更新应用程序服务器配置，请执行以下操作：

1. 找到应用程序服务器启动脚本或已设置 JVM 参数的文件。
2. 对应用程序服务器的启动脚本进行任何更改之前，对此脚本进行备份。
3. 使用编辑器或应用程序服务器控制台打开启动脚本。
4. 将 JRE Instrumenter 中的 Java 参数添加到启动应用程序服务器的 Java 命令行中，例如：

```
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.4.2_04\instr.jre;
    <probe_install_dir>\classes\boot
```

在此实例中， **<probe_install_dir>** 是 Java 代理的安装目录路径。

这会将探测器连接到应用程序。

以下是在添加 Java 参数之前启动脚本中 WebLogic Java 命令行的示例：

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

以下是在添加 Java 参数之后启动脚本中 WebLogic Java 命令行的示例：

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m
-Xbootclasspath/p:<probe_install_dir>\classes\Sun\1.5.0_17\instr.jre;
-javaagent:<probe_install_dir>\lib\probeagent.jar
-classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer
-Dbea.home= “C:\\bea” -Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy="C:\bea\wlserver6.1/lib/weblogic.policy" weblogic.Server
```

5 保存对启动脚本所做的更改。

6 在测试模式下重新启动应用程序服务器，

7 要验证探测器的配置是否正确，请检查 <探测器安装目录>\log\<探测器 ID>\probe.log 文件中的条目。如果文件中没有条目，则表明您尚未插桩应用程序服务器所使用的 JRE，或者尚未配置应用程序服务器 JVM 参数以调用 Java 代理（有关应用程序服务器，请参阅本章中的说明）。

其他配置选项

以下各章为您提供了其他配置选项：

- “配置应用程序服务器上多个 Java 进程的监控”（第 231 页）
- “在应用程序服务器中调整 Java 代理的堆大小”（第 235 页）
- “配置 SOAP 消息处理程序”（第 235 页）
- “配置新 J2EE 服务器的搜寻以填充 CI”（第 239 页）
- “关于基于 OSGi Framework 的应用程序的特殊注意事项”（第 240 页）
- “关于 Azul 用户的特殊注意事项”（第 241 页）

配置应用程序服务器上多个 Java 进程的监控

如果应用程序服务器使用多个 Java 进程，或者要收集多个 Java 进程的性能数据，则必须执行其他一些代理配置步骤。有两个选项供您选择：可以在主机上为每个 JVM 配置一个单独的 Java 代理；也可以配置一个由所有 JVM 共享的 Java 代理安装。

本节包括：

➤ “配置由多个 JVM 共享的单个 Java 代理安装”（第 231 页）

➤ “为每个 JVM 配置单独的 Java 代理安装”（第 234 页）

配置由多个 JVM 共享的单个 Java 代理安装

要允许多个 JVM 共享一个 Java 代理安装，必须为每个 JVM 配置一个单独的探测器实例。此配置可以实现下列目标：

➤ 在 Diagnostics 服务器和探测器之间建立通信

➤ Diagnostics 服务器识别探测器

➤ JVM 所使用的 JRE 的插桩

要配置由多个 JVM 共享的单个 Java 代理安装，请执行以下步骤：

当使用单个 Java 代理安装监控多个 JVM 时，必须相应配置应用程序服务器 JVM 参数以调用 Java 代理。每个 JVM 可以使用不同的 JRE 插桩模式（有关 JRE 插桩模式的详细信息，请参阅第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”）。

1 如果没有插桩每个 JRE 版本，请立即插桩。请参阅第 6 章“准备应用程序服务器以使用 Java 代理进行监控”。

2 指定探测器可从其中自动选择端口的端口范围。 Java 代理使用微型 Web 服务器进行通信。将为探测器所监控的每个 JVM 分配一个单独的端口。默认情况下，端口号的范围（最小值 / 最大值）设置为 **35000–35100**。如果探测器监控的 JVM 超过 100 个，则必须扩大端口号的范围。

注意：如果防火墙将探测器与其他 Diagnostics 组件分隔开，请配置防火墙，以允许使用所指定范围内的端口进行通信。有关详细信息，请参阅附录 “配置 Diagnostics 以在防火墙环境中工作。”

如果要配置防火墙以允许探测器使用非默认端口范围进行通信，请更新在以下内容中讨论的端口范围值。

---

a 在文件夹 < **探测器安装目录** >/**etc** 中找到 **webserver.properties** 文件。

b 设置下列属性，以调整可用于探测器通信的端口范围。

- 端口号范围中的最小端口将使用以下属性：

  jetty.port=35000

- 端口号范围中的最大端口将使用以下属性：

  jetty.max.port=35100

3 使用以下方法之一指定一个唯一的探测器名称。

必须在同一行中输入各个命令行属性，不能包含任何换行符。在 Java 命令行中定义的探测器 ID 会覆盖在 **probe.properties** 文件中使用 **id** 属性定义的探测器名称。

a 使用 Java 命令行或启动脚本，为每个 JVM 指定一个自定义的探测器标识符。

```
-Dprobe.id=<Unique_Probe_Name>
```

以下示例显示了添加 **probe.id** 参数前的 WebLogic 启动脚本：

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath "%CLASSPATH%"
-Dweblogic.Domain=petstore -Dweblogic.Name=petstoreServer -Dbea.home="C:\\bea"
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy== “C:\bea\wlserver6.1/lib/weblogic.policy” weblogic.Server
```

以下示例显示了添加 **probe.id** 参数后的 WebLogic 启动脚本：

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m"
-Xbootclasspath/
p:C:\MercuryDiagnostics\JAVAProbe\classes\Sun\1.4.1_03;C:\MercuryDiagnostics\JA
VAProbe\classes\boot"
-classpath "%CLASSPATH%"
-Dprobe.id=<Unique_Probe_Name> -Dweblogic.Domain=petstore
-Dweblogic.Name=petstoreServer
-Dbea.home= “C:\\bea” -Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Dcloudscape.system.home=./samples/eval/cloudscape/data
-Djava.security.policy== “C:\bea\wlserver6.1/lib/weblogic.policy” weblogic.Server
```

b 如果仅指定了一个 Java 参数，但使用相同脚本启动了多个探测器，请使用 %0 字符串为每个探测器生成一个自定义的探测器标识符；例如，在群集环境中，一个启动脚本可用于启动探测到的多个应用程序服务器实例。

```
-Dprobe.id=<probeName>%0
```

在 Windows 中，请使用 %%0，使用第一个 % 转义第二个 %。

%0 将被动态地替换为一个数字，从而为每个探测器创建一个唯一的探测器名称；例如， <probeName>0、 <probeName>1 等。

4 指定每个探测器要使用的点文件。默认情况下，点文件名为 auto_detect.points。您可以指定在下列情况下使用自定义点文件：必须使用多个自定义插桩计划；同一台计算机上有多个 JRE 版本使用一个代理安装；或者是一个或多个 JRE 需要在层中包含特定的方法和类，以支持自定义插桩。

```
-Dprobe.points.file.name="<Custom_AutoDetect_Points_File>"
```

为每个 JVM 配置单独的 Java 代理安装

当一个主机上有多个 JVM 时，可以为每个 JVM 实例配置一个单独的 Java 代理安装。可以多次安装代理，并通过在每个代理的安装目录的 **probe.properties** 文件中设置代理的 **id** 属性来定义探测器的实例。

要为每个 JVM 配置单独安装的代理，请执行以下步骤：

1 如果没有插桩 JRE，请立即插桩 （参阅第 6 章 “准备应用程序服务器以使用 Java 代理进行监控”）。

2 在 < **探测器安装目录** >/**etc** 目录中找到 **probe.properties** 文件。

以下是一个示例：

```
C:\\MercuryDiagnostics\JAVAProbe\etc\probe.properties
```

3 为 **id** 属性分配一个在服务器和 Diagnostics 服务器中唯一的名称，如下所示：

```
id=<uniqueProbeName>
```

启动探测器时，会在 **< 探测器安装目录 >/log** 目录中创建一个用于储存该探测器日志消息的日志文件。

在应用程序服务器中调整 Java 代理的堆大小

堆大小可影响 Java 代理和应用程序服务器的性能。堆大小的默认值为 64 MB，但应用程序服务器通常会增大该值。将 Java 代理添加到应用程序服务器时，可能需要增大堆大小以提供 Java 代理所使用的内存。有关详细信息，请参阅 “Diagnostics Java 代理主机的要求”（第 34 页）。

可以使用以下 JVM 参数在应用程序服务器 JVM 配置中设置堆大小：

```
-Xmx<size>
```

通过更新 **-Xmx<size>** 选项中指定的值，可以增加堆大小。有关设置此参数的帮助，请参阅 JVM 文档。

配置 SOAP 消息处理程序

Java 探测器需要 Diagnostics SOAP 消息处理程序以支持以下功能：

- 收集 SOAP 错误的负载。
- 根据 SOAP 标头、正文或包络确定 SOA 用户 ID。

对于大多数应用程序服务器，写入插桩点和代码段是为了自动为正在监控的 Web 服务配置 Diagnostics 处理程序。

重要信息：对于某些应用程序服务器，已在 Diagnostics 中提供特殊插桩以自动加载 Diagnostics SOAP 消息处理程序。

但是，仍需对 WebSphere 5.1 JAX-RPC 和 Oracle 10g JAX-RPC 进行一些手动配置。请参阅 “加载 Diagnostics SOAP 消息处理程序”（第 237 页）。

此外，Diagnostics SOAP 消息处理程序并非对所有应用程序服务器均可用，它也不是可从 SOAP 负载中捕获 SOAP 错误或用户 ID 的自定义插桩。因此，无法在所有版本的应用程序服务器上使用此功能。有关 Diagnostics SOAP 消息处理程序支持的最新信息，请参阅 Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

本节包括以下主题：

禁用 SOAP 消息处理程序

默认情况下， SOAP 消息处理程序为启用状态。可以按如下方式禁用处理程序：

在 **< 探测器安装目录 >/etc/inst.properties** 文件中编辑 **details.conditional.properties** 属性，使其包含 **mercury.enable.autoLoadSOAPHandler = false**。

如果 SOAP 消息处理程序处于禁用状态，则必须手动配置在链中安装处理程序的位置。

加载 Diagnostics SOAP 消息处理程序

在大多数应用程序服务器上，SOAP 消息处理程序是自动加载的，但是需要在这些应用程序服务器上执行手动配置：

WebSphere 5.1 JAX-RPC

要在 WebSphere 5.1 JAX-RPC 上配置 SOAP 消息处理程序，请执行以下步骤：

注意：WebSphere 6.1 JAX-WS Web 服务不支持 Diagnostics 处理程序，您必须用 Diagnostics SOAP 处理程序类重新编译应用程序。

---

1 查找应用程序的 Web 服务部署描述符 (**webservices.xml**)。目录路径类似于以下路径：

< 安装根目录 >\config\cell\< 服务器 >\applications\ <Web 服务 EAR>\ deployments\<Web 服务名称 >\ <Web 服务 JAR|WAR 名称 >\WEB-INF

以下是一个示例：

```
C:\Program Files\WebSphere\AppServer\config\
cells\MyServer1\application\WebServicesSamples.ear\
deployments\WebServicesSamplea\AddressBookJ2WB.war\ WEB-INF
```

2 编辑 webservices.xml，并为每个 <port-component> 添加 Diagnostics 处理程序：

```
<port-component>
 ......
 <handler>
  <handler-name>Diagnostics RPC Handler</handler-name>
   <handler-class>
    com.mercury.opal.javaprobe.handler.soap.ProbeRPCHandler
   </handler-class>
 </handler>
 ......
</port-component>
```

3 将 Diagnostics 处理程序 jar 文件 （< 探测器安装目录 >\lib\probeSOAPHandler.jar）复制到 WebSphere 的 **lib** 目录下。

以下是一个示例：

```
cp C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\ lib\probeSOAPHandler.jar
C:\Program Files\WebSphere\AppServer\lib
```

上述步骤是通过在 Windows 上运行 IBM WebSphere 5.1.0 应用程序服务器得到的。

Oracle 10g JAX-RPC

要在 Oracle 10g JAX-RPC 上配置 SOAP 消息处理程序，请执行以下步骤。

1 查找应用程序的 Web 服务部署描述符 (**webservices.xml**)。目录路径类似于以下路径：

<OC4J 安装根目录 >\j2ee\home\applications\< 应用程序名称 >\ < 部署名称 >\WEB-INF\webservices.xml

2 编辑 webservices.xml，并为每个 <port-component> 添加 Diagnostics 处理程序：

```
<port-component>
 ......
 <handler>
  <handler-name>Diagnostics RPC Handler</handler-name>
   <handler-class>
    com.mercury.opal.javaprobe.handler.soap.ProbeRPCHandler
   </handler-class>
 </handler>
 ......
</port-component>
```

3 将 Diagnostics 处理程序 jar 文件 （< 探测器安装目录 >\lib\probeSOAPHandler.jar）复制到 <OC4J_install_root>\j2ee\home\applib 目录下。

以上步骤是通过在 Windows 上运行 Oracle Containers for J2EE (OC4J) 10g Release 3 (10.1.3.3) 得到的。

配置新 J2EE 服务器的搜寻以填充 CI

代理提供数据来填充 Business Service Management 中 J2EE 应用程序服务器和 J2EE 应用程序域的 CI。

探测器会自动为诸如 JBoss 和 WebLogic 等知名 J2EE 服务器填充 CI。

您也可以配置应用程序服务器搜寻，为其他 J2EE 服务器填充 CI。应用程序服务器名称可直接指定或配置为由 JMX 搜寻或按点 / 代码段搜寻。

按照如下说明，在探测器 **etc/metrics.config** 文件中配置应用程序服务器搜寻。

类 AppServerDiscoveryCollector 位于 **<探测器安装目录>/lib/probe-jmx.jar** 文件中，您可以写入自己的采集器类，以执行应用程序服务器搜寻和度量收集。

以下是针对常规应用程序服务器的应用程序服务器搜寻配置。请注意，采集器名称区分大小写，且应当与 **metrics.config** 文件中的所有其他采集器名称不同。

```
<user-defined-collector-name>.class.name =
com.mercury.diagnostics.capture.metrics.jmx.AppServerDiscoveryCollector
<user-defined-collector-name>.class.path = probe-jmx.jar
<user-defined-collector-name>.app_server.configure.discovery = true
<user-defined-collector-name>.app_server.type = <user-defined-type>
<user-defined-collector-name>.app_server.server_name =
<user-defined-server-name>
<user-defined-collector-name>.app_server.domain_name =
<user-defined-domain-name>
```

然后，应当在 app-server/javaprobe 启动脚本或 Java 命令行中添加以下 Java 系统属性定义。

```
-Dapp_server.discovery.collector=<user-defined-collector-name>
```

每隔 15 分钟，探测器就会刷新采集器 （包括 AppServerDiscoveryCollector），并基于任何新配置进行搜寻。

对于了解如何使用 JMX 来搜寻新应用程序服务器名称和 J2EE 域名的高级用户而言，可以在探测器 **etc/metrics.config** 文件中添加以下配置。

```
<user-defined-jmx-collector-name>.class.name =
com.mercury.diagnostics.capture.metrics.jmx.JMXCollector
<user-defined-jmx-collector-name>.class.path = probe-jmx.jar
<user-defined-jmx-collector-name>.depends.on.class =
javax.management.MBeanServer
<user-defined-jmx-collector-name>.app_server.configure.discovery = true
<user-defined-jmx-collector-name>.app_server.type = <user-defined-type>
<user-defined-jmx-collector-name>.app_server.server_name =
<user-defined-server-name>
<user-defined-jmx-collector-name>.app_server.server_name.discovery.by.jmx =
<jmx-ObjectName>.<jmx-AttributeName>
<user-defined-jmx-collector-name>.app_server.domain_name =
<user-defined-domain-name>
<user-defined-jmx-collector-name>.app_server.domain_name.discovery.by.jmx =
<jmx-ObjectName-1>.<jmx-AttributeName-1>@<jmx-ObjectName-2>.<jmx-AttributeNa
me-2>
```

关于基于 OSGi Framework 的应用程序的特殊注意事项

如果应用程序基于 OSGi Framework，则可能需要设置某些其他的属性。如果不是默认值，则将 **osgi.java.profile.bootdelegation** 属性设置为默认值“ignore”。然后，在 **osgi.java.profile** 文件中将 com.mercury.* 附加到 **org.osgi.framework.bootdelegation** 属性的结尾。例如：

```
org.osgi.framework.bootdelegation= <existing packages>,com.mercury.*
```

关于 Azul 用户的特殊注意事项

Azul 为企业 Java 用户提供了两种高扩展性和高性能解决方案：Vega 和 Zing。Vega 是一种连接用户本地网络的特殊硬件设备。Zing 相当于 Vega 的虚拟设备，以 VMware 或 KVM 的访客映像形式提供。Azul 设备的主要优点在于其创新的无停顿垃圾收集器，可持续运行并处理多达数亿字节的堆。虽然仅在实验室中测试了 Zing，但两种设备都受 Diagnostics 同等的支持。

Azul 提供的 Java SDK 或 JRE 安装在传统的系统上，如 Linux 或 Solaris，但是调用它后，它会将任何 Java 代码的执行委托给 Azul 设备。因此，虽然 Java 应用程序似乎运行在调用它的系统上，但实际上是运行在其他系统上。这个过程是无缝的，因此应用程序与其环境进行交互就好像运行在本地系统上一样。如果应用程序调用 JNI，则此调用会通过网络并在原始主机上执行。

这种执行模式给 Diagnostics 用户带来了大量问题。探测器进行的 JNI 调用成本很高，但最重要的是，这些调用没有为用户提供他们所希望的结果。

- CPU 时间戳没有正确运行。调用测量的是原始服务器上所使用的 CPU 时间，因此时间戳无效。
- 因为调用测量的是前端进程，因此进程度量也无效。
- 在大多数情况下，所有系统度量都无效。调用测量的是原始系统，与在设备上运行的应用程序无关。
- 垃圾收集度量混淆。由于 Azul 使用持续的垃圾收集器，因此会视垃圾收集百分比超过 100% 为正常状态。
- 堆细分和堆内存没有运行。
- VMware 特殊计时器没有运行 （即使在 VMware 上使用虚拟设备）

为 Azul 虚拟机配置 Diagnostics

调用 Azul java 命令需要添加可正确标识用于运行此应用程序的设备的参数。这对 JREinstrumenter 来说十分困难 （除非在自动隐式模式下运行），因为 JREinstrumenter 需要运行要插桩的 JRE 以确定其版本和供应商，但它无法添加所需参数。

此解决方案将编辑 Azul JRE 安装中找到的 **azul.properties** 文件，并定义所需参数。在 JREinstrumenter 运行时需要此设置，可在使用 Diagnostics 运行此应用程序时将其删除。

要消除可能的混淆以及无意义的开销，建议在使用 Diagnostics Agent 时使用以下设置：

- 在 **metrics.config** 中，取消注释 “system” 和 “ProcessMetrics” 采集器的所有度量，以及 “Java Platform” 采集器的垃圾收集度量。
- 在 **capture.properties** 中，将 use.cpu.timestamps 设置为 false。

7

准备应用程序服务器以使用 Java 代理进行客户端监控

本节说明如何准备应用程序服务器，以使用 Java 代理进行客户端监控。

本章包括：

关于客户端监控

客户端监控功能会度量由用户浏览器显示的 Web 页面性能，并会将这些度量与后端服务器请求关联。

将度量三种重要的度量：后端时间、前端时间和总时间。

后端时间是指，从发送 Web 页面请求开始到收到第一个响应字节为止的时间量。

前端时间是指，从收到第一个响应字节开始到加载页面完毕为止的时间量。

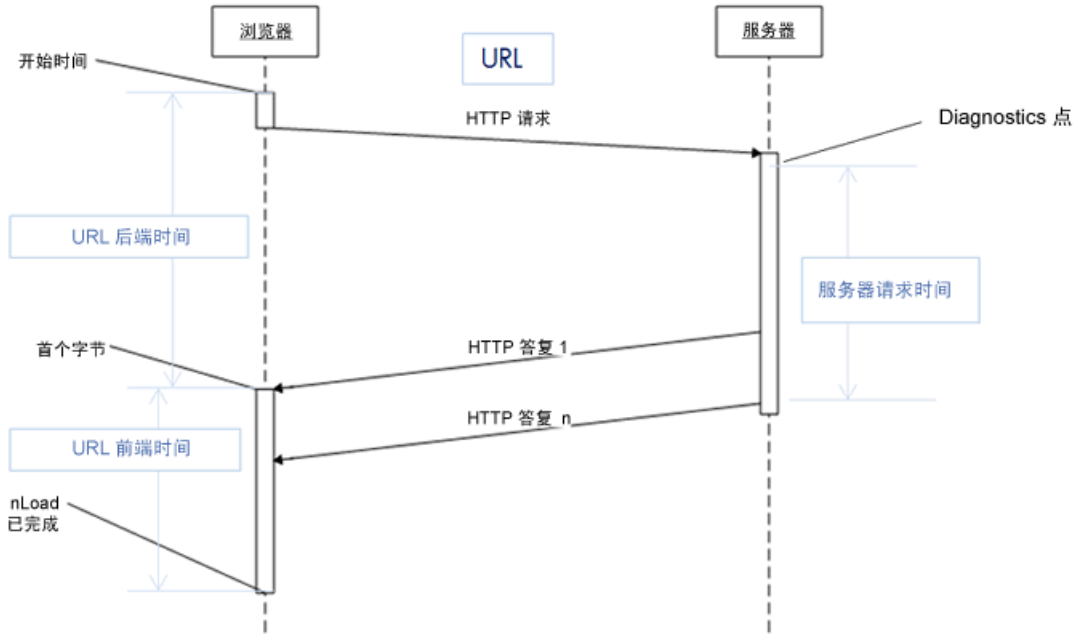
总时间是指前端时间和后端时间的总和。

客户端监控过程会聚合这些度量数据，并通过 URL、位置和“浏览器 - 操作系统”组合显示这些度量。

通过监控 Web 页面性能，应用程序的所有者可快速发现性能问题，并按照层（前端或后端）、位置和浏览器等特性区分这些问题。

如果问题位于后端，客户端监控功能会将 URL 关联到相关的服务器请求及其调用配置文件。

客户端监控功能的示例显示如下：



启用客户端监控功能

启用客户端监控功能时，需要在应用程序服务器上部署 .war 文件。在某些情况下，还需要配置 Web 服务器。“客户端监控”视图在 Diagnostics Enterprise UI 中提供。

要启用客户端监控，请执行以下操作：

在启用客户端监控功能时，系统将自动修改通过 JBoss、Tomcat、WebSphere 和 WebLogic 处理的大部分 JSP 页面，以使其在 < 标头 > 标记旁包含其他 Java 脚本。通过在浏览器内打开页面并选择 “浏览来源”，可查看已插桩的页面。

其他应用程序服务器要求为客户端监控执行手动页面插桩。请参阅 “手动插桩 HTML/JSP 页面以进行客户端监控：”（第 247 页）。

客户端监控 （包括 “自动 JSP 插桩”）在部署此 .war 文件前，将保持禁用状态。

1 部署 ClientMonitoring.war 文件。

使用应用程序服务器的管理控制台将 **< 探测器安装目录 >\contrib\ ClientMonitoring.war** 部署为应用程序。

在部署此 .war 文件前，客户端监控功能将保持禁用状态。

2 如果已将一个 Web 服务器配置为应用程序的前端，则还需要向此 Web 服务器的配置中添加以下上下文根：
/ClientMonitoring/*

提示：如果浏览器可访问以下链接，则可确认已正确配置 Web 服务器：（将返回空白页面） http://hostname:port/ClientMonitoring/B/。

例如：将客户端监控支持添加到 BSM （基于 JBoss 和 Apache Web 服务器）：

-- 复制 **< 探测器安装目录 >\contrib\ClientMonitoring.war** 到 BSM 网关服务器：**C:\HPBSM\AppServer\deploy**。

-- 将以下行添加至 BSM 网关服务器的 Apache Web 服务器配置：
C:\HPBSM\WebServer\conf\uriworkermap.properties：

/ClientMonitoring/*=localAjp

-- 重新启动 BSM 网关服务器和 BSM Apache Web 服务器。

配置和禁用客户端监控功能

如果需要，则可通过更新 **<探测器安装目录>\etc\dynamic.properties** 内的某些属性对客户端监控实行动态控制。

client.monitoring.enable 属性提供了动态启用和禁用客户端监控功能的主开关。设置为 False 时，收到的所有客户端监控数据事件将被丢弃，JSP 页面自动插桩将禁用，而且 **client.monitoring.sampling.percent** 将设置为 0.0（以禁用手动插桩的 JSP 页面的客户端监控 Java 脚本代码）。

可以通过调整 **dynamic.propertes** 中的 **client.monitoring.sampling.percent** 属性，来减轻服务器上的客户端监控负载。

此外，还可以通过将 **client.monitoring.strict.referrer** 设置为 **true** 来对引用页进行严格检查。这将确保仅使用通过客户端监控插桩的 Web 页面所生成的事件。默认值为 false。但是建议将该值设置为 true，如果此设置对您的环境有效的话。

手动插桩 HTML/JSP 页面以进行客户端监控

将以下代码添加到 HTML/JSP 页面：

```
<script>
if (window.t_firstbyte === undefined) {
    var t_firstbyte = Number(new Date());
}
</script>
<script type='text/javascript' src='/ClientMonitoring/boomerang-min.js'>
</script>
<script type='text/javascript' src='/ClientMonitoring/hp_diag-min.js'>
</script>
<script>
BOOMR.init({
    beacon_url: "/ClientMonitoring/B",
    user_ip: '10.0.0.1',
    RT: {
        cookie: "X-HP-CM-RT",
        cookie_exp: 600,
        expandFrames: true,
        hashURLs: true
    },
    HP: {
        cookie: "X-HP-CM-GUID"
    }
});
</script>
```

如果希望手动插桩 HTML/JSP 页面，则可通过在 inst.properties 内将以下属性设置为 False 永久禁用自动插桩。这些更改需要重新启动应用程序服务器。

< 探测器安装目录 >\etc\inst.properties:

```
details.conditional.properties= \
mercury.enable.clientmonitoring.JspWriterImpl.autoinstrumentation=false,\
mercury.enable.clientmonitoring.CoyoteWriter.autoinstrumentation=false,\
mercury.enable.clientmonitoring.BodyContentImpl.autoinstrumentation=false,\
```

8

安装 .NET 代理

本节描述如何安装 .NET 代理，并说明 .NET 代理的设置和配置。

本章包括：

.NET 代理安装概述

.NET 代理软件安装在要监控的应用程序所在的计算机上。借助 .NET 代理，您可以插桩应用程序域以执行监控。

有关 .NET 代理的要求，请参阅第 1 章，“准备安装 HP Diagnostics”。

.NET 代理 （版本 9.x）需要 .NET Framework 2.0 或更高版本。在运行 .NET 代理安装之前，必须在计算机上安装 .NET Framework。

重要信息：如果需要支持 .NET Framework 1.1，则需要使用较早版本的 .NET 代理 (8.x)。

WCF 要求和限制：如果要监控 .NET Windows Communication Foundation (WCF) 服务，需要安装 .NET Framework 3.0 SP1 或更高版本。支持使用以下传输的 WCF 绑定：

- Http
- TCP

如果应用程序使用不受支持的传输，则 .NET 探测器仅会为每个 WCF 方法创建常规服务器请求。这不是一种 Web 服务，也没有任何跨 VM 关联。

HP Diagnostics/TransactionVision .NET 代理安装程序用于安装 .NET 代理，以便收集 Diagnostics、 TransactionVision 或这两者的数据。

.NET 代理安装程序将在安装代理的系统上自动检测 ASP.NET 应用程序。请参阅“搜寻和标准插桩”（第 281 页）

安装程序将代理配置为对检测到的每个 ASP.NET 应用程序捕获基本工作负荷和事件。可以通过 **probe_config.xml** 文件来控制代理配置。请参阅 “对搜寻到的 ASP.NET 应用程序的自动插桩和配置”（第 282 页）。

.NET 代理使用**点文件**提供标准插桩以支持启动监控应用程序。这些点文件用于控制代理为应用程序捕获的工作负荷。请参阅第 10 章，“.NET 应用程序的自定义插桩”。请参阅 “启用和禁用应用程序的标准插桩”（第 289 页）。

将安装并启用以下点文件，以为监控 ASP.NET 应用程序提供插桩：

- ASP.NET.points
- ADO.points
- WCF.points
- 以下点文件可用于为使用其他 Microsoft 技术的应用程序提供插桩：
- Remoting.points （用于 .NET 远程环境）
- msmq.points （用于 MSMQ 环境）
- LWMD.points （用于分析应用程序内由收集使用的内存）

对于检测到的在 IIS 下安装的每个 ASP.NET 应用程序域，都会为其创建单独的插桩点文件（**<applicationDomin>.points** 文件）。**probe_config.xml** 文件包含每个检测到的 ASP.NET 应用程序的 appdomain 引用。且每个 appdomain 部分均包含一个插桩点文件引用。.NET 代理使用此运行时插桩从指定的应用程序中捕获方法延迟信息。

HP 软件即服务 (SaaS)。可将 HP Diagnostics 部署到 HP 软件即服务 (SaaS) 环境中。在 SaaS 部署中， Diagnostics .NET 代理安装在您公司的 IT 环境中，Diagnostics Commander 服务器和 Mediator 服务器由 HP 安装在 SaaS 系统中（在 HP Premise 上）。在设置 .NET 代理的过程中，可选择以下选项来配置此代理：具有 SaaS 托管的调解器 （安装在 HP Premise 上）的 Diagnostics 模式。

请参阅访问 .NET 代理安装程序开始安装。

访问 .NET 代理安装程序

有多种不同方法均可启动 .NET 代理安装程序。可从 Diagnostics 安装磁盘或 BSM 安装磁盘，或从 Business Service Management 内的 “安装” 页面安装 .NET 代理。也可从 SSO 门户安装此软件。如果要为 .NET 安装试用版 HP Diagnostics，则可从 HP 软件网站下载中心启动安装程序。

要从 Diagnostics 安装位置访问安装程序，请执行以下操作：

- Diagnostics 安装 DVD (Autorun.exe) 中将显示安装菜单页面。从菜单中选择 **Diagnostics Agent for .NET 32-bit** 启动 32 位 Windows 版本 .NET 代理的安装。选择 “Diagnostics Agent for .NET 64-bit” 启动 64 位版本的 .NET 代理安装。
- 通过在安装位置找到可执行文件 **HPDiagTV.NETAgt_< 版本号 >_win32.msi**（32 位）或 **HPDiagTV.NETAgt_< 版本号 >_win64.msi**（64 位）并将文件复制到新的安装位置，然后双击文件运行安装程序，也可直接运行对应的安装程序。

按照 “安装 .NET 代理”（第 254 页）中的说明继续操作。

要从 HP 软件下载中心下载安装程序，请执行以下操作：

1 使用您的 “HP Passport”登录帐户可访问 SSO 门户，网址为 http://support.openview.hp.com/selfsolve。

2 查找 Diagnostics（或 TransactionVision）下载资源，并选择下载 Diagnostics .NET 代理软件的对应链接。请注意，还可以通过下载中心获得 Diagnostics .NET Profiler 试用 / 评估版软件。

3 按照网站上的下载说明进行操作。

按照 “安装 .NET 代理”（第 254 页）中的说明继续操作。

要从 Business Service Management Diagnostics “下载”页面下载安装程序，请执行以下操作：

1 在“Business Service Management”的主菜单中选择“管理”>“Diagnostics”，然后单击 “下载”选项卡。或在主菜单中选择 “管理”>“平台”，然后单击 “设置和维护”选项卡。

2 在 “下载”页面上单击相应的链接，下载用于 32 或 64 位 Windows 的 .NET 代理安装程序。

注意：如果将 .NET 代理安装程序放置到 Business Service Management 访问所需的目录中，则可以在 Business Service Management 中使用这些安装程序。可以在 Diagnostic 服务器安装期间启用安装程序，也可以手动将 .NET 代理安装程序从安装光盘复制到所需位置。

按照 “安装 .NET 代理”（第 254 页）中的说明继续操作。

要从 HP Software 试用软件下载网站启动 HP Diagnostics Profiler for .NET 试用软件的安装程序，请执行以下操作：

1 转至 HP 软件网站的下载中心。

2 在 “快速搜索”区域的 “产品”列表中，单击 “Diagnostics”并单击 “搜索”。

3 在 “试用版软件”下选择相应链接。

4 按照网站上的下载说明进行操作。

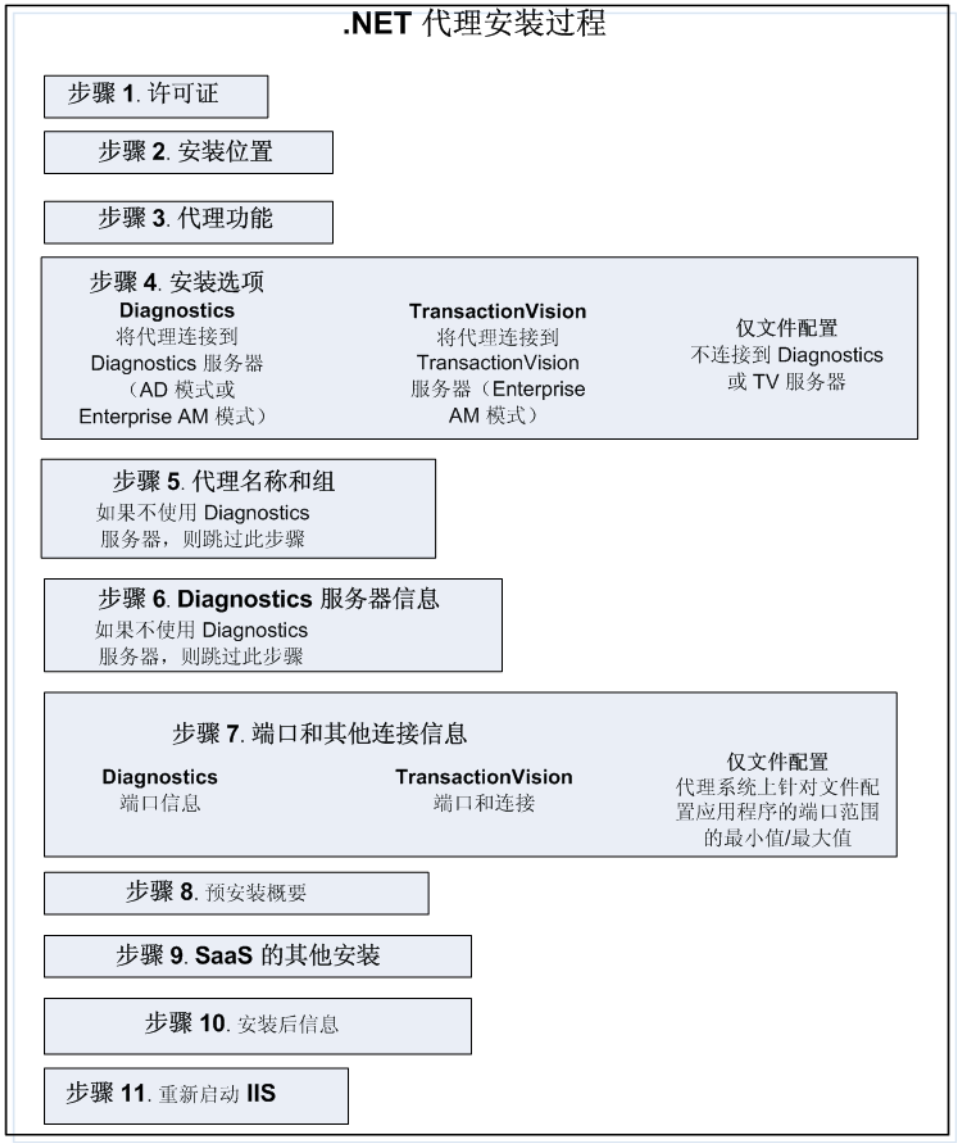
按照 “安装 .NET 代理”（第 254 页）中的说明继续操作。

安装 .NET 代理

本章提供有关*首次*安装 .NET 代理的详细说明。

重要信息：如果主机上存在预安装的 .NET 代理，则必须按照说明升级此代理系统，而不遵循这些安装说明，请参阅 “升级和修补程序安装说明”（第 851 页）。

下图概述了 .NET 代理的安装步骤；有关每个步骤的详细信息，请参阅本节其余内容。



.NET 代理安装过程包括以下步骤，请选择步骤 1. 最终用户许可证协议开始安装：

步骤 1. 最终用户许可证协议

接受最终用户许可证协议。

阅读协议并选择 “I accept the terms of the License Agreement”。

单击 “Next”继续下一步。

步骤 2. 指定安装位置

提供代理的安装位置。

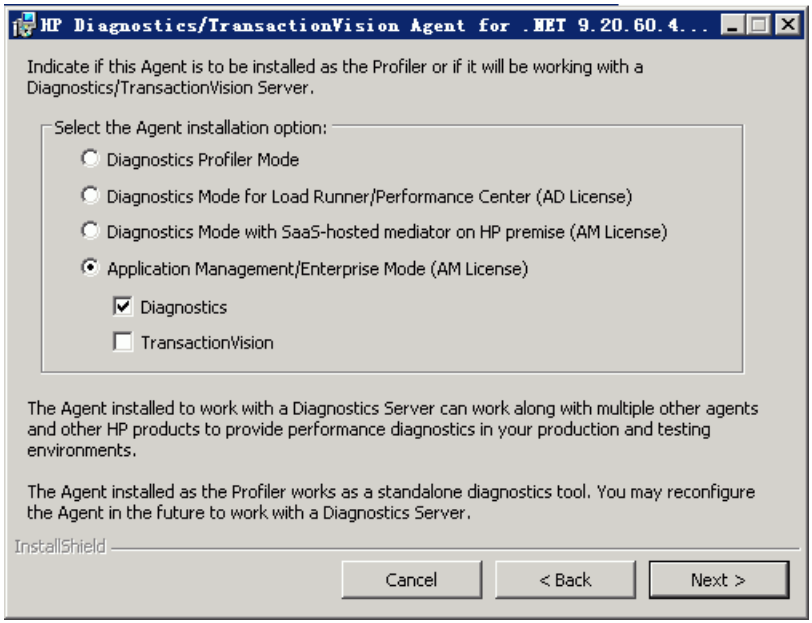
默认情况下，代理安装在 **C:\MercuryDiagnostics\.NET Probe** 中。此位置将变为 < 探测器安装目录 >。

接受默认目录，或者通过键入其他路径或单击 “Browse”导航到安装目录，以选择其他位置。

单击 “Next”继续下一步。

步骤 3. 选择安装选项

指明是将 .NET 代理作为独立 Profiler 安装，而不建立与服务器的任何连接 （例如，安装 Diagnostics .NET Profiler 试用版软件），还是安装代理以运行 LoadRunner/Performance 或与 Diagnostics 和 / 或 TransactionVision Server 配合使用。



选择适用于代理使用环境的方式。

➤ **Diagnostics Profiler Mode。**选择此选项将代理作为 Diagnostics .NET Profiler 安装，而不与 Diagnostics 服务器建立任何连接。在购买 HP Diagnostics 产品之前安装 Diagnostics .NET Profiler 试用版软件时，通常选择此选项。

如果选择 “Diagnostics Profiler Mode” 选项， **probe_config.xml < 模式 >** 元素的值在安装 .NET 代理时将设置为 **pro** 模式（请参阅 “<modes> 元素”（第 548 页））。

➤ **Diagnostics Mode for LoadRunner/Performance Center (AD License)。**选择此选项可安装代理以在负载测试（或预生产）环境中与 Diagnostics 服务器配合使用；在负载测试环境中，仅在 LoadRunner 或 Performance Center 运行中使用探测器。

在 AD 模式下运行探测器的好处在于，在 LoadRunner 或 Performance Center 运行时，仅针对 HP Diagnostics AD 许可证容量计算 AD 模式下的探测器。例如，如果在 LoadRunner/Performance Center AD 模式下安装了 20 个探测器，但一次运行中只有 5 个探测器，则只会针对 AD 许可证容量计算 5 个探测器。有关 AD 许可证容量的详细信息，请参阅 “基于当前连接的探测器许可证信息”（第 83 页）。

在 AD 模式下， .NET 代理仅在 LoadRunner 或 Performance Center 运行时捕获数据，并将结果存储在用于该运行的特定 Diagnostics 数据库中，例如 Default Client:21。当代理处于 AD 模式下时，除非探测器参与了 LoadRunner/Performance Center 运行，否则代理将不会使用资源或将任何数据发送到服务器。

如果选择此 AD 许可证选项， **probe_config.xml < 模式 >** 元素的值在安装 .NET 代理时将设置为 **ad** 模式 （请参阅 “<modes> 元素”（第 548 页））。

➤ **Diagnostics Mode with SaaS-hosted mediator on HP premise (AM License)。**选择此选项可安装代理以在 SaaS 环境中工作，在此环境中， .NET 代理将连接到 HP Premise 上的 HP SaaS 服务器。HP SaaS 管理员将为您提供有关将 .NET 代理连接到托管 HP SaaS 的 Diagnostics 调节器服务器的信息。

➤ **Application Management/Enterprise Mode (AM License)**。选择此选项可将代理安装为在企业（或生产）环境中与 Diagnostics 服务器和 / 或 TransactionVision Server 配合使用。

然后指定以下哪些服务器需要配置代理：

- ➤ Diagnostics 服务器 （本地安装）
- ➤ TransactionVision 服务器
- ➤ 本地安装的 Diagnostics 服务器和 TransactionVision 服务器

如果选择 TransactionVision，则请参阅 Business Service Management 文档库中的《HP TransactionVision Deployment Guide》，获取有关特定于 TransactionVision 的设置选项的详细信息。

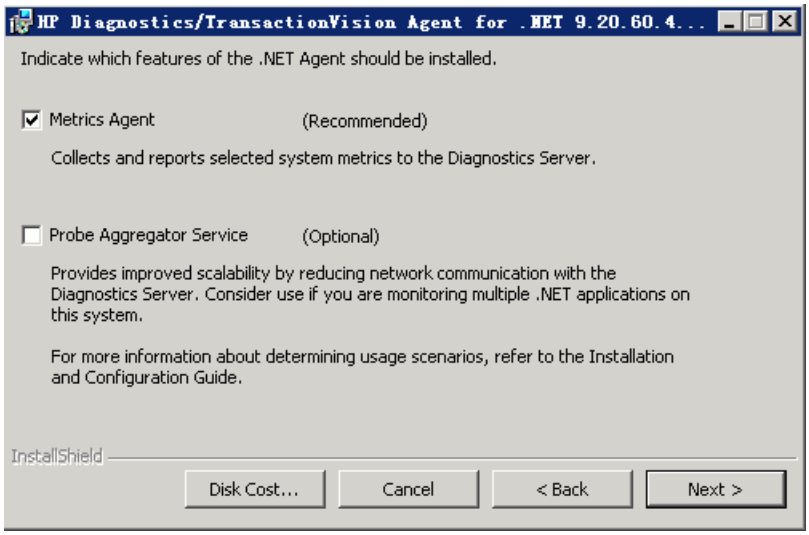
选择此选项后，如果在安装 .NET 代理时选择了 Diagnostics 服务器，**probe_config.xml < 模式 >** 元素的值将设置为**企业**模式；如果选择了 TransactionVision Server，则该值将设置为 **TV** 模式 （请参见 “<modes> 元素”（第 548 页））。

对于那些已设置企业模式的代理，将针对 HP Diagnostics AM 许可证容量计算代理。

单击 “Next” 继续下一步。

步骤 4. 选择要安装的代理功能

选择要安装的 .NET 代理功能。



Metrics Agent。建议您安装默认情况下已选中的度量代理。有关详细信息，请参阅第 17 章，“.NET 系统度量代理 - 系统度量捕获”。但是，如果不需要捕获主机上的系统度量，则可以取消选中 “Metrics Agent” 框。

Probe Aggregator Service。（可选）可以选择安装 Probe Aggregator 服务。如果正在配置代理使其在 HP SaaS 环境中工作，则会勾选 “Probe Aggregator ”，因为 SaaS 需要此选项，且无法更改。

在将性能数据发送到 Diagnostics Mediator 服务器之前，Probe Aggregator 服务将每 5 秒聚合一次 .NET 代理数据。这样可以通过减少与服务器的网络通信来提升可扩展性，但是 Aggregator 本身也将导致增加探测器系统开销。有关权衡性能以决定是否要安装 Probe Aggregator 的详细信息，请参阅“Probe Aggregator 服务”（第 285 页）。

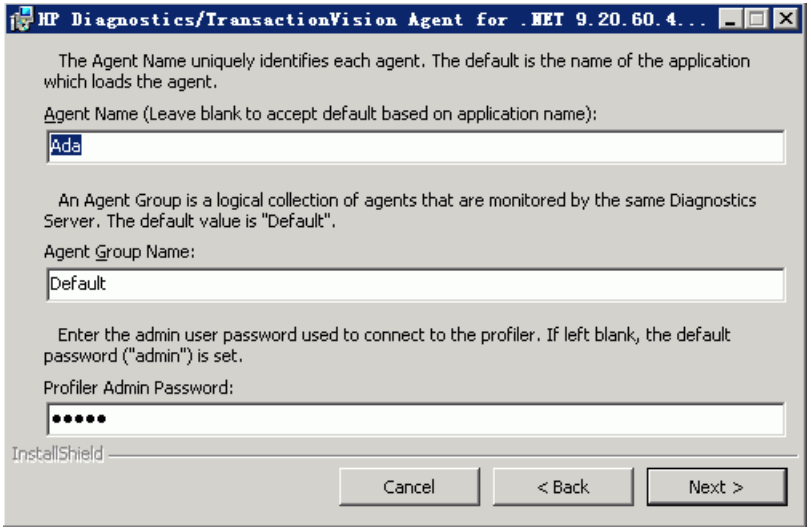
Disk Cost。要检查主机驱动器上的可用磁盘空间量，请单击“Disk Cost”按钮。此功能可确保有足够空间进行代理安装。

单击 “Next”继续下一步。

步骤 5. 代理名称和组

如果代理不向 Diagnostics 服务器进行报告，则可跳过此步骤。

输入代理名称和代理组名称。



➤ **Agent Name。**用于在 HP Diagnostics 中标识代理的名称。如果将此字段保留为空，则 .NET 代理将根据受监控应用程序的应用程序域名自动生成代理名称。代理名称将指定为探测器实体名称。

注意：建议将 “Agent Name”保留为空，以允许代理自动生成代理名称。如果要输入您自己的代理名称，请仔细阅读以下信息。

输入代理名称时的注意事项：

- 代理名称中的有效字符包括：字母、数字、短划线、下划线和句点。
- 可指定有助于识别受监控的应用程序和插桩类型的代理名称。

  例如，已安装用于监控 PetWorld 应用程序的 .NET 代理的名称可以是：

  PetWorld_Dotnet_Agent

- 指定代理名称后，主机上的所有代理将强制使用同一个代理名称。

  将代理名称字段保留为空时，代理自动生成的默认代理名称相当于指定 $(APPDOMAIN).NET。

  要覆盖该默认名称，请在运行时使用以下替换宏来增强名称：

  - $(MACHINENAME)：计算机主机名
  - $(APPDOMAIN)：应用程序域名
  - $(PID)：应用程序进程 ID
  - $(WEBSITENAME)：应用程序所在的 IIS 网站。
  - $(COMMANDLINE:n)，其中 n 为命令行参数号。

    例如：

    <id probeid=”ILTEST_$(COMMANDLINE:3)_rest” probegroup=”Default”/>

    及命令行 iltest “heart and lung” -abc server 可产生 ILTEST_server_rest 的 probeid。

请注意， n=0 表示可执行 / 命令名称。

注意：

- 对于 IIS 托管之外的应用程序，代理名称将会转换为默认名称 $(APPDOMAIN).NET。控制台应用程序就是这样的一个示例。
- 对于新安装的 IIS 应用程序，您可能需要从 Windows “开始”菜单的 “HP Diagnostics .NET 代理”程序组中运行 “重新扫描 ASP.NET 应用程序”。

- **Agent Group Name：**为现有组或将要创建的新组输入名称。代理组名称的默认值是 Default。代理组名称区分大小写。在 Diagnostics 中，代理组名称被用作探测器组名称。

  探测器组是向相同的 Diagnostics 服务器进行报告的探测器的逻辑分组。将跟踪探测器组的性能度量，并在多个 Diagnostics 视图上显示性能度量。

  例如，可以将某个特定企业应用程序的所有探测器分配到一个探测器组，这样既可以监控组级别的性能，又可以监控各个探测器实体的性能。

- **Profiler Admin Password。**输入用于连接 .NET Diagnostics Profiler 的管理用户密码。如果将其留为空白，则会设置默认密码 (admin)。

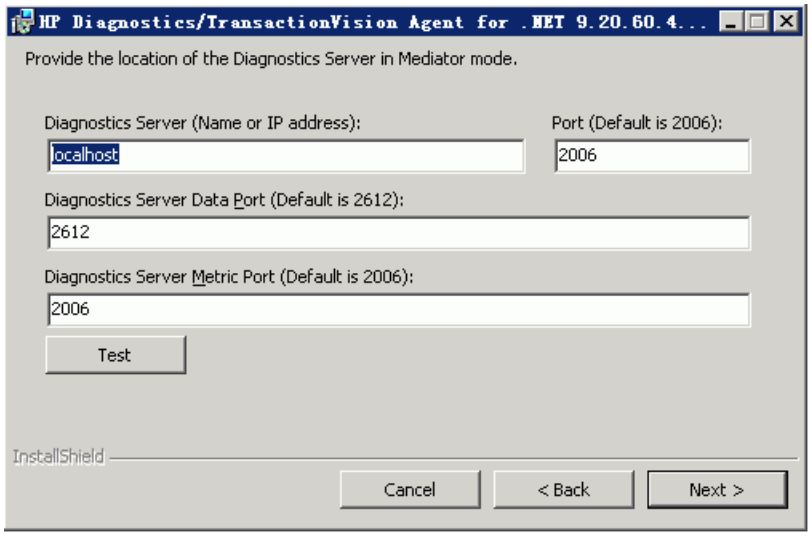
  单击 “Next”继续下一步。

步骤 6. Diagnostics 服务器信息

如果代理不向 Diagnostics 服务器报告或者安装代理以使其在 HP SaaS 环境中工作，可跳过此步骤。 HP SaaS 管理员将提供有关配置代理和托管 SaaS 的 Diagnostics 服务器之间通信的详细信息。

提供用于支持 .NET 代理与 Diagnostics Mediator 服务器之间的通信的信息。

如果选择安装 Probe Aggregator 服务，将会显示 Probe Aggregator 数据端口和度量端口，而不是 Diagnostics 服务器数据端口和度量端口。



- 在“Diagnostics Server (Name or IP address)”框中键入 Diagnostics Mediator 服务器主机的主机名或 IP 地址。
- 应指定完全限定主机名而不是简单的主机名。在包含 UNIX 的混合操作系统环境中，必须指定完全限定主机名，才能保证网络路由正常工作。
- 在 “Diagnostics Server Data Port”框中键入 Diagnostics 服务器用于侦听代理通信的端口号。默认端口号是 2612。如果在 Diagnostics 服务器安装后更改了端口，请在此处指定该端口号以代替默认端口号。
- 如果选择安装 Probe Aggregator Service，则会显示 “Probe Aggregator Data Port”框，而不是 “Diagnostics Server data port”框。安装探测器聚合时，可键入 Diagnostics Mediator 服务器用于侦听代理通信的端口号。默认端口号是 2626。如果在 Diagnostics 服务器安装后更改了端口，请指定该端口号以代替默认端口号。

- 在 “Diagnostics Server Metric Port” 框中键入 Diagnostics 服务器用于侦听系统度量代理通信的端口号。默认端口号是 **2006**。如果在安装 Diagnostics 服务器后更改了端口，请在此处指定该端口号而不是默认端口号。
- 如果选择安装 Probe Aggregator 服务，则会显示 “Probe Aggregator Metric Port” 框，而不是 “Diagnostics Server metric port” 框。安装探测器聚合时，可键入 Diagnostics Mediator 服务器用于侦听代理通信的端口号。默认端口号是 45000。如果在 Diagnostics 服务器安装后更改了端口，请指定该端口号以代替默认端口号。
- 要执行连接性检查以确保 Diagnostics 服务器正在运行并可通过安装主机进行访问，请单击 “Test”。
- 通过连接性检查，可以立即确定所提供的关于 Diagnostics Mediator 服务器的信息是否存在错误，以及 Diagnostics 服务器主机和代理主机之间的连接是否存在问题。如果无法连接到 Diagnostics Mediator 服务器主机，则会显示一条错误消息。
- 单击 “Next” 继续下一步。

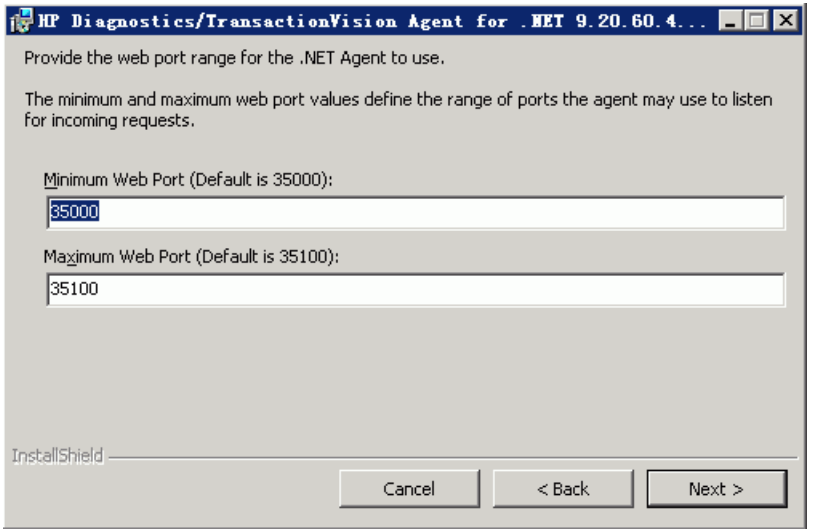
步骤 7. 端口和连接信息

根据您所选择的安装选项，将看到不同的端口和连接配置对话框。从以下选项进行选择，然后继续配置：

- Diagnostics 服务器的端口连接信息
- TransactionVision Server 的端口和连接信息
- 不与 Diagnostics 或 TransactionVision Server 连接的 Profiler 模式

如果要将代理安装为与 Diagnostics 服务器配合使用，则会看到以下对话框。

提供 Web 端口范围以供 .NET 代理使用。



- **Minimum Web Port。**键入要分配给代理的代理主机端口范围内的最小端口号。
- **Maximum Web Port。**键入要分配给代理的代理主机端口范围内的最大端口号。

注意：默认范围为 35000 到 35100 （包含 35000 和 35100）。

Web 端口范围的上限和下限由 “Minimum Web Port” 和 “Maximum Web Port” 字段确定。 Web 端口范围包含了代理可使用的端口。

代理启动后，它会尝试查找此范围中未使用的端口，从最小端口号开始查找，直至最大端口号。如果其他代理或应用程序之前请求了该范围中的某些端口，则这些端口可能会处于已使用状态。

该端口范围的最小大小等于将要在代理主机上并发运行的最大代理数目。

设置 Web 端口范围时的注意事项：

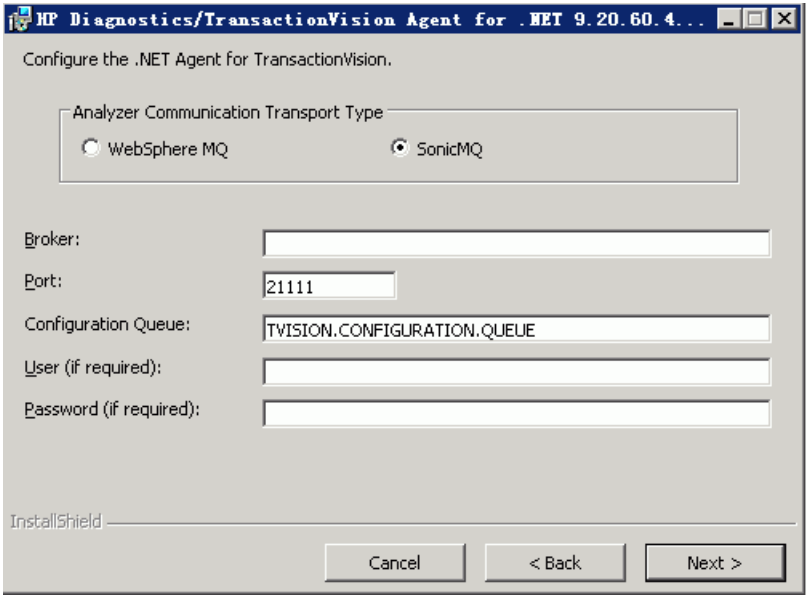
- 如果要将代理与 ASP.NET 应用程序配合使用，则应将端口数加倍，以供 ASP.NET appdomain 循环使用。
- 如果代理和与其通信的组件之间存在防火墙，应为范围内的端口开放防火墙。调整该范围以使其足够大。

单击 “Next” 继续下一步。如果还选择了让代理在 TransactionVision 环境内运行的选项，请参阅以下章节，了解其他配置。

如果要将代理安装为在 TransactionVision 环境内工作，则将看到以下对话框。

如果选择了将代理安装为与 TransactionVision Server 配合使用，则将在安装过程中看到其他屏幕。有关在 TransactionVision 环境内使用代理的信息，请参阅《TransactionVision Deployment Guide》。

出现 “Configure the .NET Agent for TransactionVision” 对话框。



选择消息中间件提供程序。选项有：WebSphere MQ 和 SonicMQ。

SonicMQ 与 .NET 代理包含在一起。如果指定此选项，则 Sonic MQ .NET 客户端 （Sonic.Client.dll - Progress SonicMQ .NET 客户端，版本 7.6.0.112）将作为代理安装的一部分进行安装。

也可以使用第三方 WebSphere MQ 安装。在这种情况下，必须在监控的主机上安装 MQ 系列 .NET 客户端（amqmdnet.dll - WebSphere MQ Classes for .NET，版本 1.0.0.3）。

默认情况下，将选择 SonicMQ。

➤ 对于 SonicMQ，请输入以下内容：

Broker。其上运行 Sonic 代理的主机的名称。通常是分析器主机名。

Port。代理通信时所使用的端口。默认为 21111。

Configuration Queue。配置队列的名称。默认为 TVISION.CONFIGURATION.QUEUE。

User。用户 ID （SonicMQ 安装可能需要此 ID 进行连接）。默认情况下无需用户名。

Password。密码 （SonicMQ 安装可能需要此密码进行连接）。因通过 **PassGen** 实用程序创建，形式模糊。默认情况下无需密码。有关 **PassGen** 的详细信息，请参阅 《Using Transaction Management》中的 “Command-Line Utilities”。

➤ 对于 WebSphere MQ，请输入以下内容：

Host。 WebSphere MQ 队列管理器所在的主机。

Port。 WebSphere MQ 队列管理器的端口号。

Configuration Queue。配置队列的名称。

User。用户 ID （WebSphere 安装可能需要此 ID 进行连接）。

Password。密码 （WebSphere MQ 安装可能需要此密码进行连接）。因通过 PassGen 实用程序创建，形式模糊。有关 **PassGen** 的详细信息，请参阅 《Using Transaction Management》中的 “Command-Line Utilities”。

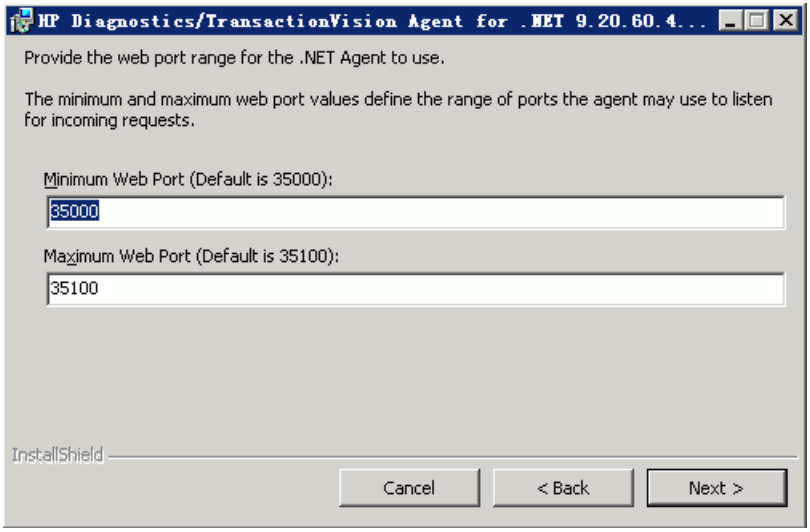
Websphere MQ channel。 WebSphere MQ 队列管理器的通道名称。

Websphere MQ Q Manager。 WebSphere 的 CCSID。

单击 “Next”继续下一步。

如果在 Profiler 模式下安装代理，则会看到以下对话框：

提供 Web 端口范围以供 .NET 代理使用。



- **Minimum Web Port。**键入要分配给代理的代理主机端口范围内的最小端口号。
- **Maximum Web Port。**键入要分配给代理的代理主机端口范围内的最大端口号。

注意：默认范围为 35000 到 35100 （包含 35000 和 35100）。

Web 端口范围的上限和下限由 “Minimum Web Port” 和 “Maximum Web Port” 字段确定。 Web 端口范围包含了代理可使用的端口。

代理启动后，它会尝试查找此范围中未使用的端口，从最小端口号开始查找，直至最大端口号。如果其他代理或应用程序之前请求了该范围中的某些端口，则这些端口可能会处于已使用状态。

该端口范围的最小大小等于将要在代理主机上并发运行的最大代理数目。

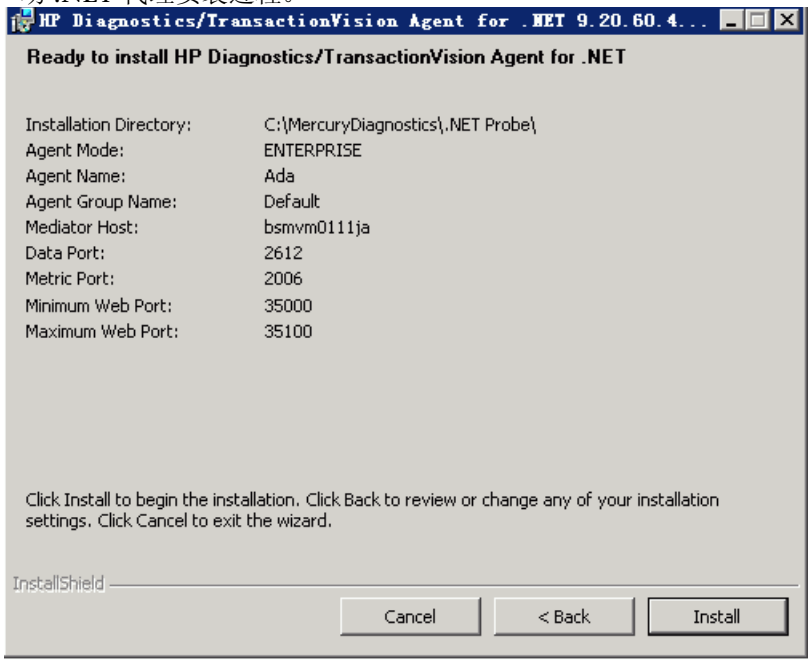
设置 Web 端口范围时的注意事项：

- 如果要将代理与 ASP.NET 应用程序配合使用，则建议您将端口数加倍，以供 ASP.NET appdomain 循环使用。
- 如果代理和与其通信的组件之间存在防火墙，则必须为范围内的端口开放防火墙。因此，您可能要调整该范围，以使其足够大。

单击 “Next” 继续下一步。

步骤 8. 预安装概要

此时将打开预安装概要屏幕。单击 “Back” 可进行任何更改。单击 “Install” 启动 .NET 代理安装过程。



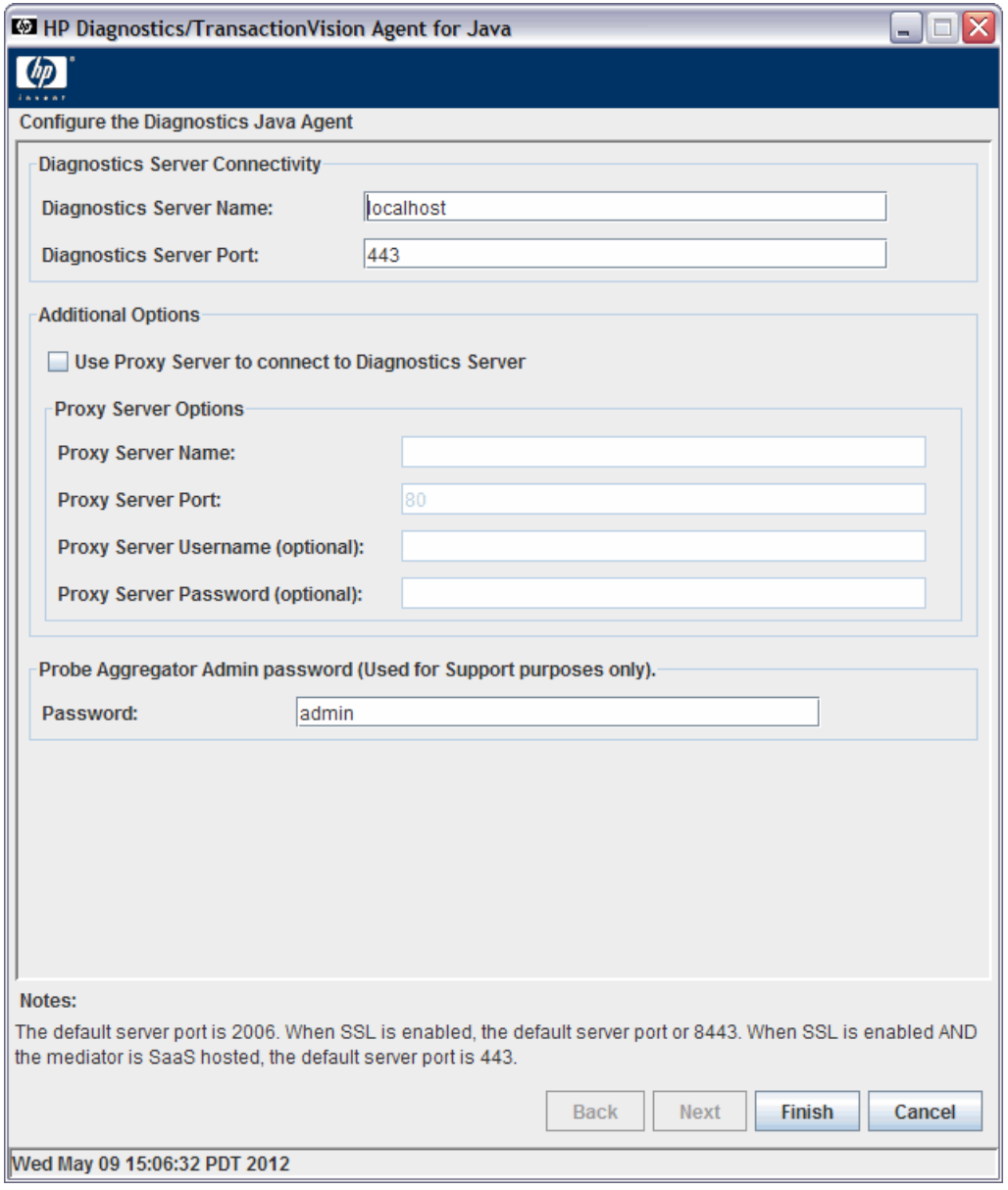
注意：仅安装代理作为 Profiler 使用时，不会测试度量端口的连接性。

如果安装代理以使其在 HP SaaS 环境中工作，请继续执行步骤 9，或者跳过下一步骤并继续执行步骤 10。

步骤 9. 使代理在 HP SaaS 环境中工作的其他设置

如果安装代理以使其在 HP SaaS 环境中工作，则 SaaS 设置模块会自动启动，或者可通过选择“开始”>“所有程序”>“HP Diagnostics .NET Probe”>“SaaS Setup”随时运行 SaaS 设置模块。

在 SaaS 设置模块中，会显示以下对话。如果不针对 HP SaaS 环境设置代理，则不会看到此对话。


HP Diagnostics/TransactionVision Agent for Java
Configure the Diagnostics Java Agent
Diagnostics Server Connectivity
Diagnostics Server Name:
localhost
Diagnostics Server Port:
443
Additional Options
Use Proxy Server to connect to Diagnostics Server
Proxy Server Options
Proxy Server Name:
Proxy Server Port:
80
Proxy Server Username (optional):
Proxy Server Password (optional):
Probe Aggregator Admin password (Used for Support purposes only).
Password:
admin
Notes:
The default server port is 2006. When SSL is enabled, the default server port or 8443. When SSL is enabled AND the mediator is SaaS hosted, the default server port is 443.
Back
Next
Finish
Cancel
Wed May 09 15:06:32 PDT 2012

- **Diagnostics Server Connectivity。**在 HP SaaS 环境中，Diagnostics 服务器是由系统上的 HP Premise 由 HP 设置的。SaaS 环境 的默认端口是 **443**。HP SaaS 管理员将为您提供有关要使用的 Diagnostics 服务器主机名和端口的信息。
- 如果代理服务器用于与 Diagnostics Mediator 服务器通信，请选中 “Use Proxy Server to connect to Diagnostics Server”复选框并输入相应的选项。在 HP SaaS 环境中，如果您的公司需要代理以与外部服务器通信，则您可以选中此选项。

  Proxy Server Options：

  - **Proxy Server Name。**代理服务器的主机名。
  - **Proxy Server Port。**代理服务器的端口。
  - **Proxy Server Username (optional)。**用于对代理服务器进行身份验证的用户。
  - **Proxy Server Password (optional)。**用于对代理服务器进行身份验证的密码。
- **Probe Aggregator Admin password。**该密码会自动设置为与 .NET Profiler 管理密码 （在步骤 5 中输入）相同的密码，因此在首次为 SaaS 设置代理时，此字段不可见。如果随后希望更改 Probe Aggregator 管理密码，可再次运行 SaaS 设置模块，并将显示此字段。

  继续执行下一步骤以完成安装。

步骤 10. 安装后信息

在最终安装屏幕上，可选择 “Show the Windows Installer Log”复选框来查看日志文件并检查错误。

单击 “Finish”退出安装程序。

如果在 SaaS 环境中使用代理，则会在托管 HP SaaS 的服务器上生成证书，并会在安装和设置的过程中下载该证书。您需要将此证书复制到 .NET 代理的主机上，并导入此证书。请参阅 “对于 SaaS 环境 - 导入证书”（第 276 页）。

有关安装后任务的信息，请参见 “安装后任务”（第 275 页）。

准备就绪后，必须重新启动 IIS，请参见以下步骤。

步骤 11. 重新启动 IIS

完成代理的安装和设置后，必须重新启动 IIS 或 Web 发布服务，才能将 .NET 代理与 ASP.NET 应用程序配合使用。

要从命令行或从 “开始” > “运行” 菜单重新启动 IIS，请键入 iisreset，并按 “Enter”。

对于 Diagnostics，此命令将重新启动 Web 发布服务，但不会立即启动 .NET 代理。当下次请求应用程序中的 Web 页面时，将启动代理并插桩应用程序，并且代理将使用 Diagnostics 命令服务器注册。

对于 TransactionVision，此命令将重新启动 Web 发布服务，但不会立即启动 .NET 代理。下次请求应用程序中的 Web 页面时，将启动代理，检测应用程序，并且代理将读取分析器的配置队列消息。

注意：只有在一定情况下 ASP.NET 才会自动重新启动应用程序，这些情况包括当检测到应用程序已重新部署时，或当应用程序超过了所配置的资源阈值时。

ASP.NET 重新启动受 .NET 代理监控的应用程序时，将停用代理并启动新代理。与此同时，可能会存在一个重叠期，此期间中多个代理会同时使用 Diagnostics 命令服务器注册并且会连接到 Diagnostics Mediator 服务器。此情况可能会导致 LoadRunner/Performance Center 和 Business Service Management 在应用程序重新启动期间报错。

继续下一部分，可了解安装后任务的更多信息。

有关验证安装的信息，请参阅 “验证 .NET 代理安装”（第 276 页）。

安装后任务

有关对 .NET 代理的其他配置的信息，请参阅以下主题：

- 有关 .NET 代理自动如何自动发现应用程序并配置标准插桩以支持监控的信息，请参阅 “搜寻和标准插桩”（第 281 页）。
- 有关为 Diagnostics 配置 .NET 代理的信息以及更多高级主题的链接，请参阅“关于针对 Diagnostics 的 .NET 代理配置”（第 278 页）。
- 要了解为 TransactionVision 配置 .NET 代理的相关信息并查看 TransactionVision 可利用 .NET 代理跟踪的事件类型，请参阅“关于针对 TransactionVision 的 .NET 代理配置”（第 279 页）。
- 有关详细信息，请参阅 “启用和禁用应用程序的标准插桩”（第 289 页）。
- 有关使用代理配置环境的信息，请参阅 “针对 HTTP 代理配置 Diagnostics 服务器和代理”（第 627 页）；有关防火墙的信息，请参阅 “配置 Diagnostics 以在防火墙环境中工作”（第 631 页）；有关启用 HTTPS，请参阅 “在组件之间启用 HTTPS”（第 797 页）。

验证 .NET 代理安装

在最终安装屏幕上，可选择 “Show the Windows Installer Log” 复选框来查看日志文件并检查错误。

在探测器启动之前，.NET 代理不会注册 Diagnostics 服务器。插桩的应用程序运行后，探测器将启动并注册服务器。对于 ASP.NET 应用程序，所插桩的应用程序首次请求页面时会出现这种情况。

在启动 .NET 探测器实例之后，可以启动 Diagnostics Enterprise UI 以验证探测器是否正在工作。转至 **http://<Diagnostics_commander_server>:2006/**。此时，可使用默认的用户 / 密码即 “admin/admin”，或给定的登录名（如果已为您设置此登录名）。

此外，还可检查系统运行状况视图，查找有关 .NET 代理部署及托管服务器的计算机的信息。

要访问系统视图，请执行以下操作：

1. 以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。
2. 在查询页面的列表中查找 Mercury 系统用户，并选择此链接打开 Diagnostics。
3. 登录 Diagnostics 并在 “应用程序” 窗口中选择 “整个企业”，然后选择任何链接打开 Diagnostics 视图。
4. 在 “视图” 窗格中，将显示 “系统视图” 视图组。打开视图组，并选择 “系统运行状况” 视图或 “系统容量” 视图。

对于 SaaS 环境 - 导入证书

如果在 SaaS 环境中使用代理，则会在托管 HP SaaS 的服务器上生成证书，并会将其下载到 .NET 代理主机系统。您将需要按照以下所述导入证书。有关在非 SaaS 环境中为 .NET 代理启用 HTTP 通信，请参阅附录 C，“在组件之间启用 HTTPS”。

要从托管 HP SaaS 的服务器导入证书，请执行以下步骤：

1. 来自托管 HP SaaS 的服务器的证书将下载到 .NET 代理主机系统，如下所示：

   C:\MercuryDiagnostics\.NET Probe\ProbeAggregator\etc\jasm_server0.cer
   C:\MercuryDiagnostics\.NET Probe\ProbeAggregator\etc\jasm_server1.cer

2. 在 Windows 任务栏的 .NET 代理主机系统上，选择 “开始” > “运行”。
3. 键入 mmc 并单击 “确定”，以运行 Microsoft 管理控制台。
4. 在 Microsoft 管理控制台菜单上，选择 “文件” > “添加 / 删除管理单元”，以显示 “添加 / 删除管理单元” 对话框。
5. 在 “添加 / 删除管理单元” 对话框中单击 “添加”。
6. 从 “可用的独立管理单元” 列表中选择 “证书”，然后单击 “添加”。
7. 在 “证书管理单元” 对话框中选择 “计算机帐户”，然后单击 “下一步”。
8. 在 “选择计算机” 对话框中选择 “本地计算机 （运行此控制台的计算机）”。
9. 从列表选择 “全球 Web 发布服务”，然后单击 “完成”。
10. 单击 “添加独立管理单元” 中的 “关闭”。
11. 在 “添加 / 删除管理单元” 对话框中单击 “确定”。
12. 在 Microsoft 管理控制台的 “控制台根目录” 对话框的左窗格中，展开 “证书 (本地计算机)” 的列表。
13. 在 “证书 (本地计算机)” 下，展开 “受信任的根证书颁发机构”。
14. 在 “受信任的根证书颁发机构” 下右键单击 “证书”，并选择 “所有任务” > “导入” 以启动 “证书导入向导”。
15. 单击 “下一步” 跳过 “证书导入向导” 的 “欢迎” 对话框。
16. 单击 “浏览” 以导航到下载证书所在的目录 C:\MercuryDiagnostics\.NET Probe\ProbeAggregator\etc\。选择每个证书并逐个导入。

a 单击“下一步”导入文件。

b 单击“下一步”接受“受信任的根证书颁发机构”的默认证书存储位置。

17 单击“完成”，以完成“证书导入向导”。

18 在“证书导入向导”的确认对话框中单击“确定”。在“受信任的根证书颁发机构”下选择“证书”，以查找您刚添加的证书。

19 重新启动 IIS。

关于针对 Diagnostics 的 .NET 代理配置

可以自定义 .NET 代理配置并添加自定义插桩，以适应环境和要诊断的性能问题。

安装程序会配置 ASP.NET 应用程序和 .NET 代理，使它们配合使用，以捕获应用程序的基本工作负荷。但是，一个或多个 ASP.NET 应用程序的安装方式可使它们无法被安装程序检测出来。或者，您可能要增强标准插桩，以捕获应用程序中的自定义类的性能度量。

可以在 Diagnostics 中使用 **probe_config.xml** 文件设置其他配置。有关此文件的详细信息，请参阅第 13 章，“了解 .NET 代理配置文件”。有关高级 .NET 代理配置的说明，请参阅第 14 章，“高级 .NET 代理配置”。

此外，还可以在 Diagnostics 中创建自定义插桩点，以处理应用程序环境中的特殊情况。有关自定义插桩的常规信息，请参阅第 10 章，“.NET 应用程序的自定义插桩”。

关于针对 TransactionVision 的 .NET 代理配置

与 TransactionVision 配合使用时，.NET 代理将从 .NET 应用程序捕获事件，并将事件发送至 TransactionVision 分析器。有关 TransactionVision 的详细信息，请参阅 *Business Service Management 文档库*。

针对 TransactionVision 的 .NET 代理配置

.NET 代理的默认配置允许您跟踪受监控的应用程序内的某些 .NET 事件。可以自定义 .NET 代理配置以控制要跟踪和发送至 TransactionVision 分析器的 .NET 事件。

要覆盖默认配置，请访问 **< 代理安装目录 >/etc/ probe_config.xml** 文件。有关可为 Diagnostics 和 TransactionVision 配置的元素的详细信息，请参阅 “了解 .NET 代理配置文件”（第 507 页）。

probe_config.xml 文件内的 < 模式 > 元素可在 Diagnostics 和 TransactionVision 安装时设置 （请参阅 “<modes> 元素”（第 548 页））。

选择将 .NET 代理安装为在 TransactionVision 环境内运行，则 **probe_config.xml** 文件内的 < 模式 > 元素将设置为 tv。如果这是唯一选定的模式，则代理将在 “仅 TV” 模式中运行，即禁用 Profiler 和 Diagnostics 探测器，且仅生成 TV 事件。选择将 .NET 代理安装为在其他环境内运行 （如配合 Diagnostics 使用）时，将启用 TV 事件和 Diagnostics 数据收集。

要指定特定于 TransactionVision 和特定于 TransactionVision 传输的配置，**probe_config.xml** 文件内的以下元素仅适用于 TransactionVision：

- <tv> 元素 （请参阅 “<tv> 元素”（第 575 页）了解详细信息）
- <timeskew> 元素 （请参阅 “<timeskew> 元素”（第 570 页）了解详细信息）
- <transport> 元素 （请参阅 “<transport> 元素”（第 572 页）了解详细信息）

➤ <gentvhttpeventforwcf> 元素 （请参阅 “<gentvhttpeventforwcf> 元素” （第 526 页）了解详细信息）

如果 .NET 代理使用 SonicMQ 传输与 TransactionVision 分析器通信，则可启用 SSL。有关详细信息，请参阅 《HP Business Service Management Hardening Guide》 PDF 文档。

默认情况下， .NET 事件并不关联。要启用关联，请参考 HP TransactionVision 文档。

TransactionVision 可利用 .NET 代理跟踪的事件类型

与 TransactionVision 配合使用时， .NET 代理将跟踪以下类型的 .NET 事件：

1 Web 服务

   a ASP.NET (*.asmx) - 客户端和服务器

   要生成事件，请使用 **ASP.NET.points** 文件。

   b WCF (*.svc) - 客户端和服务器

   要生成事件，请使用 **wcf.points** 文件。

   c REST 样式服务 - 服务器

   要生成事件，请按照 “为监控配置 WCF REST 服务”（第 404 页）中所述使用 **wcf.points** 文件并设置应用程序的插桩。

2 使用 .NET 执行的数据库调用

   要生成事件，请使用 **ADO.points** 文件。

3 .NET Remoting - 客户端和服务器

   要生成 .NET remoting 事件，请按照 “如何为 .NET Remoting 配置插桩” （第 411 页）中所述使用 **Remoting.points** 文件并为应用程序设置插桩。

4 MSMQ - 发生和接收 （异步）

   要生成事件，请使用 **Msmq.points** 文件。

5 HTTP

a 客户端出站 - 包括对 REST 服务的调用

要生成事件，请使用 **ASP.NET.points** 文件。

b ASP.NET 入站 / 服务器 （POST、 GET、 PUT） (*.aspx)

要为 HTTP 生成事件，请使用 **ASP.NET.points** 文件。

6 用户定义的事件

使用详细信息参数 **tv:user_event** （请参阅 “可选的点条目”（第 392 页））

要关闭事件生成，请从范围中删除相关点文件。

搜寻和标准插桩

.NET 代理安装程序将自动搜寻您可能要插桩的 ASP.NET 应用程序。安装 .NET 代理 后，即可请求代理重新扫描 IIS 配置以捕获任何新增或更改项。

在安装时搜寻 ASP.NET 应用程序

安装代理时，.NET 代理安装程序会在计算机上检测 ASP.NET 应用程序。.NET 代理安装程序将通过检查 IIS 配置并查找可能会引用 ASP.NET 应用程序的虚拟目录条目，来搜寻应用程序。

在某些情况下， ASP.NET 应用程序的安装方式可使它们无法被此过程检测出来。例如，将 ASP.NET 应用程序作为 Web 目录而不是虚拟目录安装的情况就是这样的示例。

安装完成后搜寻 ASP.NET 应用程序

如果您修改了现有 ASP.NET 应用程序部署或安装了新的 ASP.NET 应用程序，则可以请求重新扫描 IIS 配置。

要请求代理重新扫描 IIS 配置并更新 **probe_config.xml** 文件，请选择 “开始” > “HP Diagnostics .NET Probe” > “Rescan ASP.NET Applications”。

对搜寻到的 ASP.NET 应用程序的自动插桩和配置

.NET 代理安装程序会对代理进行配置，以便为检测到的每个 ASP.NET 应用程序捕获基本的 ASP.NET/ADO/WCF 工作负荷。代理将执行以下配置步骤：

- 创建特定于应用程序的捕获点文件模板。

  捕获点文件用于定义插桩，插桩可控制代理为每个应用程序捕获的工作负荷。您可以在捕获点文件中修改插桩，从而使代理捕获特定于应用程序的自定义方法的性能数据。

- 在 **< 探测器安装目录 >/etc** 目录下的 **probe_config.xml** 文件中创建 **appdomain** 标记。 **appdomain** 标记的属性将控制 .NET 代理的行为 （点和 enabled 属性）。有关详细信息，请参阅第 13 章，“了解 .NET 代理配置文件”。

注意：Diagnostics 通过将 **process** 标记中的 **enablealldomains** 属性设置为 **true**（将覆盖 appdomain 标记的 enabled 属性），来为所有搜寻到的应用程序启用插桩。有关为应用程序启用和禁用插桩的信息，请参阅“禁用日志记录”（第 288 页）。

为在 IIS 下部署的 ASP.NET 应用程序的 CI 填充搜寻 IIS 元数据

对于 Diagnostics 9.0x 或更高版本， Diagnostics 会在 Business Service Management 9.0 或更高版本运行时服务模型 (RTSM) 中填充有关应用程序基础结构元素和业务事务的 CI 和模型关系。

对于 CI 填充， .NET 代理安装程序会自动为在 IIS 6.x 或更高版本下部署的 ASP.NET 应用程序搜寻 IIS 配置元数据。搜寻到的 IIS 配置元数据将被写入 **< 探测器安装目录 >/etc** 目录下的 **iis_discovery_data.xml** 文件。安装 .NET 代理后，即可请求代理重新扫描 IIS 配置，以对任何新增或更改项进行更新。

注意：此信息用于与 Business Service Management 9.0 或更高版本的集成

➤ 在 IIS 下部署的 ASP.NET 应用程序的运行时填充 CI

在运行时， .NET 代理会在 **iis_discovery_data.xml** 文件中查询与所插桩的 appdomain 关联的 IIS 配置元数据。如果找到了关联的元数据，则代理会将数据转发到负责填充 .NET 应用程序的 Business Service Management 运行时服务模型 CI 的 Diagnostic 服务器。有关与 .NET 应用程序的 Business Service Management 9.0 运行时服务模型相集成的详细信息，请参阅第 21 章，“设置 Business Service Management 和 Diagnostics 之间的集成”。

➤ 在安装过程中搜寻 IIS 下部署的 ASP.NET 应用程序的 IIS 元数据

安装代理时，.NET 代理安装程序会在计算机上搜寻在 IIS 下部署的 ASP.NET 应用程序。.NET 代理安装程序将通过向 WMI (WMEB) 提供商查询 IIS 配置元数据来搜寻应用程序。然后，会将相关元数据写入 **iis_discovery_data.xml** 文件中。

➤ 在安装后搜寻 IIS 下部署的 ASP.NET 应用程序的 IIS 元数据

如果修改了现有 ASP.NET 应用程序部署或安装了新的 ASP.NET 应用程序，则必须请求重新扫描 IIS 配置元数据。要请求代理重新扫描 IIS 配置并编写新的 **iis_discovery_data.xml** 文件，请运行 “开始” > “HP Diagnostics .NET Probe” > “Rescan ASP.NET Applications” 快捷方式。请注意，新的 **iis_discovery_data.xml** 文件不应由用户进行编辑；执行此快捷方式将会覆盖用户所做的任何编辑。

➤ 对 IIS 下部署的 ASP.NET 应用程序进行搜寻的权限要求

用户必须在装有 .NET 代理的计算机上拥有管理员权限，否则 WMI 查询将失败，并且不会创建 **iis_discovery_data.xml** 文件。

➤ 对 IIS 下部署的 ASP.NET 应用程序的搜寻进行调试

如果未创建 **iis_discovery_data.xml** 文件，或者出于某种原因而怀疑某些元数据可能不准确，则可以创建详细调试文件，以检查 WMI 查询的结果。要创建详细调试文件，请将 “开始” > “HP Diagnostics .NET Probe” > “Rescan ASP.NET Applications” 快捷方式的 “目标属性” 的最后一个参数从 “false” 更改为 “true”。执行 “重新扫描 ASP.NET 应用程序” 快捷方式后，会创建 **< 探测器安装目录 >/log/AutoDetect.log** 文件。请注意，执行此快捷方式需要拥有管理员权限。可以将 **AutoDetect.log** 发送到 HP 支持以进行分析。

非 ASP.NET 应用程序

.NET 代理安装会自动搜寻 ASP.NET 应用程序，在 **probe_config.xml** 中为这些应用程序创建设置，并为它们创建模板点文件。对于每个非 ASP.NET 应用程序（例如，NT 服务、控制台应用程序、UI 客户端），必须在 **probe_config.xml** 设置中创建相应的设置，以配置 .NET 代理来监控应用程序，还必须创建点文件，以指示要在应用程序中监控的点。

以下是名为 **SimpleConsoleHost.exe** 的应用程序的 **probe_config.xml** 设置示例：

```
<process name="SimpleConsoleHost">
      <points file="SimpleConsoleHost.points"/>
      <logging level=" "/>
</process>
```

以下是名为 SimpleConsoleHost.exe 的应用程序的点文件设置示例：

```
[SimpleConsoleHost]
class = MyNamespace.SimpleConsoleHost
method = !.*
ignoreMethod = Main
layer = SimpleConsoleHost
```

有关更多详细信息，请参阅第 10 章，“.NET 应用程序的自定义插桩”。

Probe Aggregator 服务

可以在安装 .NET 代理时选择安装 Probe Aggregator 服务。它将作为 Windows 服务 **HP Probe Aggregator** 运行。

在将性能数据发送到 Diagnostics Mediator 服务器之前，Probe Aggregator 服务将每 5 秒聚合一次探测器数据。这在以下情况下十分有用：基于多个应用程序插桩所收集的数据量很大，而且如果不聚合数据则网络流量将太大。有关该数据处理的技术图，请参阅 “.NET Probe Aggregator 数据流”（第 850 页）。

如果仅进行基本 .NET 代理安装，而不包含 Probe Aggregator 服务，则会在方法开始和停止执行时将性能数据发送到 Diagnostics Mediator 服务器。

使用 Probe Aggregator 服务需要权衡性能，因此必须评估环境中的要求。例如，当同一系统中有两个或更多的 .NET 探测器实例运行时，请考虑使用 Probe Aggregator。实际网络开销取决于受监控的应用程序，因此需要确定是否可能节省网络带宽和 Mediator 负载，从而抵消应用程序系统中增加的内存使用量。

安装包含 Probe Aggregator 服务的 .NET 代理时，此服务将自动运行并等待与 .NET 探测器连接。会在 .NET 代理安装期间执行 Probe Aggregator 的标准配置。**< 探测器安装目录 >\ProbeAggregator\etc\probeaggregator.properties** 文件用于设置 Probe Aggregator 的配置参数 （例如，设置 SQL 趋势阈值）。

如果决定在安装完成后安装 Probe Aggregator 服务，可以再次运行 .NET 代理安装，并选择“更改”按钮。然后，选中用于安装“Probe Aggregator 服务”的复选框。

删除或卸载 .NET 代理的同时也会删除 Probe Aggregator 服务。请参阅“启用和禁用 Diagnostics 的 .NET 代理”（第 287 页）。

有关如何禁用和启用 Probe Aggregator 服务的信息，请参阅 “启用和禁用 Diagnostics 的 .NET 代理”（第 287 页）。

监控在 Azure Cloud 内部署的 NET 应用程序

Microsoft 为开发人员提供 Windows Azure SDK 以供其在 Microsoft Windows Azure Cloud 基础结构内创建并部署 Azure 应用程序。 Diagnostics .NET 代理利用 Azure SDK 实现了将 .NET 代理无缝部署到 Azure 基础结构中。一旦部署在 Azure Cloud 内运行的 .NET 代理监控应用程序后，便可为分析和问题检测收集性能数据并向 HP Diagnostics 服务器报告。有关安装和配置 .NET 代理以监控 Windows Azure Cloud 内的应用程序的详细信息，请参阅 NET Agent AzurePack zip 文件内的 **AzurePackReadMe.pdf**。

确定 .NET 代理的版本

在请求支持时，了解已安装的 Diagnostics 组件的版本十分有用。

要确定 .NET 代理 的版本，请执行以下操作：

➤ 右键单击 **< 代理安装目录 >\bin\HP.Profiler.dll** 文件，并从菜单中选择“属性”。在 “属性”对话框中选择 “版本”选项卡，可显示组件版本信息。

也可以在 Diagnostics UI 附录 D，“使用管理员的系统视图”中使用系统运行监控器）。

启用和禁用 Diagnostics 的 .NET 代理

安装完成后，便会启用 .NET 代理。在重新启动 Web 服务器并访问应用程序中的 URL 之后， .NET 代理将开始收集性能信息。

可以禁用 .NET 代理，使其不启动也不收集性能度量。

要禁用 .NET 代理，请执行以下操作：

➤ 选择“开始”>“所有程序”>“HP Diagnostics .NET Probe”>“Disable HP .NET Probe”。

要启用已禁用的 .NET 代理，请执行以下操作：

➤ 选择“开始”>“所有程序”>“HP Diagnostics .NET Probe”>“Enable HP .NET Probe”。

注意: 禁用 .NET 代理时仅会禁用探测器度量采集器和活动的探测器，而不会禁用系统度量采集器。启用或禁用系统度量采集器的过程由标准 Windows 服务管理器控制。只有在下次重新启动探测到的应用程序时，对探测器的启用或禁用操作才会生效。这对当前正在运行的应用程序不会产生任何影响。

安装并运行 Probe Aggregator 服务后，您可以从“开始”菜单禁用和启用它。选择“开始”>“所有程序”>“HP Diagnostics .NET Probe”>“Disable HP .NET Probe”或“Enable HP .NET Probe”。如果选择“Disable HP .NET Probe”，则除禁用 .NET 探测器之外，还会将 Probe Aggregator 服务标记为已禁用，但不会停止该服务（以防止还有正在运行的探测器）。如果选择“Enable HP .NET Probe”，则除启用 .NET 探测器之外，还会将 Probe Aggregator 服务更改为“自动”类型，并根据需要启动它。

禁用日志记录

可以通过更改 **probe_config.xml** 文件中 ASP.NET 进程部分的 **logging level** 标记，来禁用探测器应用程序日志记录，如下例所示：

```
<process name="ASP.NET">
      <logging level="off"/>
</process>
```

可以通过更改插桩部分的 **logging level** 标记来禁用探测器插桩日志记录，如下例所示：

```
<instrumentation>
      <logging level="off" />
</instrumentation>
```

启用和禁用应用程序的标准插桩

首次安装 .NET 代理时，会启用对所有已搜寻到的应用程序的标准 ASP.NET/ADO 插桩，但不会启用任何特定于应用程序的插桩。可以通过 .NET 代理的 **probe_config.xml** 文件中 <process> 元素的 enablealldoamins 属性的各个属性，以及 <appdomain> 元素中的属性，来控制要启用或禁用插桩的应用程序。

通过禁用应用程序的插桩，使您能够节省处理开销，并且不必关注 Diagnostics 视图中与要监控性能的环境无关的应用程序的信息。

通过启用所有应用程序的插桩，可以使 .NET 代理监控所有已检测到的应用程序的性能，以便您在 Diagnostics 视图和 Profiler 用户界面中查看所有应用程序的性能度量数据。

以下是 <process> 元素的 enablealldomains 属性的规则：

- enablealldomains = false：如果在 <appdomain> 列表中没有域，则不启用任何域。
- enablealldomains = false：如果在 <appdomain> 列表中有域且 <appdomain> 的 “enable” 属性设置为 true 或未定义该属性，则应启用域。
- enablealldomains = true：如果在 <appdomain> 列表中有域，则应当仅启用列表中的域，而不考虑其 “enable” 属性为何。
- enablealldomains = true：如果在 <appdomain> 列表中没有域，则应启用所有域。
- 未定义 enablealldomains 属性：与 enablealldomains = true 情况相同。

要启用或禁用对应用程序的插桩，请执行以下操作：

1. 将 **<process>** 元素中的 **enablealldomains** 属性设置为 **false**。这将允许各个 appdomain 标记的属性控制各个应用程序的插桩状态。如果没有应用程序域条目，则不会启用任何应用程序。

2. 对于要为其启用插桩的每个应用程序，将 **<appdomain>** 元素中的 **enabled** 属性设置为 **true**。

3. 对于要为其禁用插桩的每个应用程序，将 **<appdomain>** 元素中的 **enabled** 属性设置为 **false**。

下例显示了为一个应用程序启用而为另一个应用程序禁用的插桩：

```
<process name="ASP.NET" enablealldomains="false">
    <points file="ASP.NET.points" />
    <points file="ADO.points" />
    <appdomain name="1/ROOT/myApplication" website=”Default Web Site”
enabled="true">
        <points file="DefaultWebsite-myApplication.points" />
    </appdomain>
    <appdomain name="1/ROOT/myApplicationTwo" website=”Default Web Site”
enabled="false">
        <points file="DefaultWebsite-myApplicationTwo.points"/>
    </appdomain>
</process>
```

要为所有应用程序启用插桩，请执行以下操作：

➤ 将 **<process>** 元素中的 **enablealldomains** 属性设置为 **true**。这样将覆盖每个 **<appdomain>** 元素中的属性设置，从而不必设置多个属性，便可启用插桩。

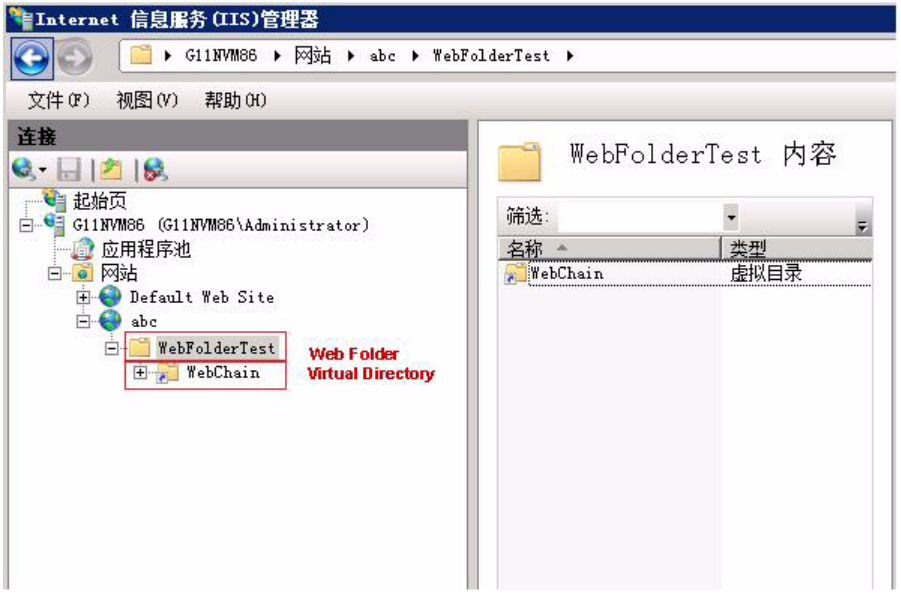
下例显示了为所有应用程序启用的插桩：

```
<process name="ASP.NET" enablealldomains="true">
    <logging level=""/>
    <points file="ASP.NET.points"/>
    <appdomain name="1/ROOT/myApplication" website=”Default Web Site”
enabled="false">
        <points file="DefaultWebsite-myApplication.points"/>
    </appdomain>
    <appdomain name="1/ROOT/myApplicationTwo" website=”Default Web Site”
enabled="false">
        <points file="DefaultWebsite-myApplicationTwo.points"/>
    </appdomain>
</process>
```

对未发现的 .NET Web 应用程序进行故障排除

在 Microsoft Windows 2003 服务器和 IIS 6 环境中，如果网站在 Web 文件夹下具有虚拟目录，则 .NET 代理可能无法发现此虚拟目录。这是因为 Diagnostics 用于向下遍历网站树的 Microsoft WMI 提供程序遇到了问题。 WMI 提供程序无法正确将 Web 文件夹识别为 IIS Web 目录，因此 Diagnostics 无法发现此文件夹下的虚拟目录。请见下方描述的示例。

此示例显示了网站 ABC 下的 Web 文件夹 WebFolderTest。在此 Web 文件夹下存在虚拟目录 WebChain。



由于 WMI 提供程序的问题，此网站 WMI 内的列表将不会显示 WebFolderTest/WebChain 虚拟目录。.NET 代理使用来自 WMI 提供程序的列表发现 Web 应用程序。因此，在类似情况下， .NET 代理将无法发现 Web 文件夹下的虚拟目录。

Microsoft 建议直接修改元数据库，或使用如下所示较为简单的脚本设置使用 ADSI 的文件夹样式：

```
Set objRoot = GetObject("IIS://localhost/W3SVC/1/Root/WebFolderTest")
objRoot.KeyType = "IIsWebDirectory"
objRoot.SetInfo()
```

若不使用脚本，您还可手动将 Web 文件夹配置为 IIS 内的应用程序。完成此配置后，便可将其转换为非应用程序，但此时将设置属性，且 Diagnostics 将能够发现 Web 应用程序。

另一种选择是在 **probe_config.xml** 文件内手动添加 ASP.NET appdomain 列表内不包含的 APPDOMAIN。

关于 .NET 代理的故障排除的其他提示

如果无法正确启动代理，则需检查以下事项：

- 确保重新启动了 Web 服务器，且已访问了应用程序内的 URL，这将触发代理开始收集数据。
- 检查是否已创建 probe_config.xml 文件，且是否格式正确 （即不缺失标签结束符等）。通过在 Web 浏览器内打开文件可完成此操作。
- 在 Windows 事件日志内查找任何名为 “HP Diagnostics” 的消息。此日志仅由 .NET 代理使用。每次尝试插桩应用程序时均应获得一条消息。

卸载 .NET 代理

要安装 .NET 代理，请执行以下操作：

1. 停止正在使用 SOAP 的所有 Web 应用程序。
2. 从 Windows “控制面板” 中选择 “添加或删除程序”，然后选择 **HP Diagnostics/ TransactionVision Agent for .NET** 进行卸载。
3. 重新启动 Web 应用程序。

要删除 Probe Aggregator 服务，您可以卸载 .NET 代理，这会同时删除 Probe Aggregator 服务。您也可以再次运行 .NET 代理安装，选择 “Change” 按钮，然后取消选中用于安装 “Probe Aggregator Service” 的复选框。

第 IV 部分

自定义插桩以监控 Java 和 .NET 应用程序

本部分包括：

- Java 应用程序的自定义插桩
- .NET 应用程序的自定义插桩

9

Java 应用程序的自定义插桩

本节描述如何控制 Diagnostics 应用到应用程序的类和方法的插桩，以支持 Java 代理收集性能度量数据。

本章包括：

关于插桩和捕获点文件

插桩是指，当虚拟机的类加载程序加载应用程序的类文件时，探测器在这些类文件中插入的字节码。通过插桩，探测器能够测量执行时间、计算调用数、检索参数、捕获异常以及关联方法调用和线程。可在捕获点文件中指定每个探测器实例的插桩点。

在安装 Java 代理时，将为要使用的平台安装预定义的默认捕获点文件，同时还会安装一组点。包含预定义 Java 点的该默认捕获点文件位于 **< 探测器安装目录 >\etc\auto_detect.points** 中。

您可以通过捕获点文件来控制插桩的范围，以便 Diagnostics 为您提供了解应用程序性能所需的所有信息，而避免被代价高昂或含糊不清的无关信息淹没。捕获点文件中的插桩定义称为 “点” 。这些点定义要插桩的方法、这些方法的插桩步骤，以及应安装的插桩。

点可以包括将 “通配”指令中的正则表达式，以便将它们应用于多个方法、类和程序包或名称空间规范。有关如何使用正则表达式的详细信息，请参阅 “使用正则表达式”（第 882 页）。

要添加自定义插桩，请先复制默认的 **auto_detect.points** 文件，并为副本文件指定不同的名称，然后使用该文件自定义所有插桩。此预防措施可防止您在升级到新版本 Java 代理时丢失自定义插桩，且安装程序将覆盖 auto_detect.points 文件。

您可以对捕获点文件中的点进行自定义，使其包括不属于默认点的应用程序区域的方法、类、程序包和名称空间。通常，当 J2EE 应用程序包含非派生自 javax.ejb.SessionBean 接口的业务逻辑时，可能需要自定义点。另外，当您希望覆盖默认点以改变其层或从特定调用方方法跟踪它时，也可能需要自定义点。

捕获点文件中的点将被划分成多个层。利用层，可以将性能度量组织成有意义的信息层，可在分类过程中对其进行比较。它们可控制插桩的收集行为。

与 Java 代理一起安装的捕获点文件中的点将被划分成多个默认层。您可以自定义默认层，也可以新建层。有关默认层的描述，请参阅“为典型 Java 类和方法定义的默认层”（第 383 页）。

注意：

- 默认捕获点文件名在 **<探测器安装目录>\etc\probe.properties** 中进行指定。
- 要覆盖默认文件名，以便使用其中包含自定义点的副本，请使用 -Dprobe.points.file.name=<newPointsFileName_NoExtension> JVM 属性。

对捕获点文件中的点进行编码

可使用以下参数定义捕获点文件中的点：

```
[Point-Name]            =<unique name for the point>
;-------------------------------------------------------------------------
class                   = <class name or regular expression>
method                  = <method name or regular expression>
signature               = <method signature or regular expressions>
ignore_cl               = <list of class names or regular expressions>
ignore_method           = <list of method names or regular expressions>
ignore_tree             = <list of class names or regular expressions>
method_access_filter    = <list of class names or regular expressions>
deep_mode               = <soft or hard mode>
scope                   = <list of methods or regular expressions>
ignoreScope             = <list of methods or regular expressions>
detail                  = <list of specifiers>
layer                   = <layer name>
layerType               = <layer type>
rootRenameTo            = <string>
keyword                 = <keyword>
priority                = <integer number>
active                  = <true, false>
```

以下各节将介绍这些参数。

- “强制点参数”（第 301 页）
- “可选的点条目”（第 302 页）

强制点参数

除 CLP、LWMD、RMI 和 SAP RFC、HttpCorrelation 及 JDBC SQL 的点之外的所有点都必须包含以下参数：

参数	描述
Point-Name	点的唯一名称。
class	指定要插桩的类或接口的名称。该名称应当包括完整的程序包 / 名称空间名称，各程序包级别之间使用句点隔开。可以使用任何有效的正则表达式。
method	指定要插桩的方法的名称。为保证成功，方法名称必须与 class 参数指定的类或接口中定义的方法匹配。可以使用任何有效的正则表达式。
signature	使用方法签名的 javap 符号编码 (<jdk_install>/bin.javap -s) 指定方法的签名 （参数和结果类型）。
layer	指定在其下分组此点中数据的层、子层或层级。层是插桩收集控制的一部分。 通过使用斜杠 (/) 分隔层名称，可将点中的层指定为嵌套层或子层。斜杠后面指定的层是斜杠前指定层的子层。一个子层还可以有自己的子层，方法是在子层名称后面再编写另一个斜杠和层名。 在 UI 中，各子层将显示在其父层下。例如，子层 JSP 和 Struts 显示在 web 层下。如果向下搜索，则从 web 到 JSP 和 Struts 都会显示出来。

以下是一个包含强制参数的自定义点示例：

```
[MyCustomEntry_1]
; comments here….
class = myPackage.myClass.MyFoo
method = myMethod
signature = !.*
layer = myCustomStuff
```

注意：正则表达式可用于点中的大多数参数，但是必须以感叹号开头。有关使用正则表达式的详细信息，请参阅 “使用正则表达式”（第 882 页）。

可选的点条目

点定义中可包含以下参数中一个或多个：

参数	描述
keyword	**keyword** 表示由特殊插桩类处理的插桩点。可在 **inst.properties** 中查找作为属性值 keyword，所找到的属性值是插桩类名称。特殊插桩点可使用未在此处说明的特定于插桩的参数，请参阅 inst.properties 文件内的注释。
ignore_cl	指定要忽略的以逗号分隔的类名或正则表达式列表。将不会使用与通过 **ignore_cl** 指定的类相匹配的任何类。
ignore_method	指定要忽略的以逗号分隔的方法列表。将不会插桩与通过 **ignore_method** 指定的方法相匹配的任何方法。

参数	描述
Ignore_tree	类或正则表达式的列表。将从插桩中排除与其中一个列表项匹配的类的任何子类。
method_access_filter	用逗号分隔的方法说明符列表。可用的说明符包括“静态”、“专用”、“受保护”、“程序包”和“公用”。如果方法的访问说明符与所列出的任一值相符，则不对与此点匹配的该方法进行插桩。
deep_mode	指定处理子类的方式。此参数接受三个值： ➤ **none** – “none”值相当于不指定 **deep_mode** 参数。它对子类的处理方式没有任何影响。 ➤ **soft** – “soft”值要求每个与该类、方法和签名条目匹配的类或接口，以及实施相同匹配方法和签名的任何子类或子接口必须插桩。 ➤ **hard** – “hard”值请求对于与类、方法和签名条目，以及任何深度的子类或子界面匹配的每个类或界面，都应插桩其方法。硬模式通常用于接口的点。**注意:** 硬模式可导致大量插桩，并会使探测器开销过大。
scope	约束执行插桩时所在的上下文。如果指定了该参数，则插入的字节码将是调用方。任何有效的正则表达式均可用于此参数的值。范围值是标准 Java 表示法的程序包、类和方法名称的以逗号分隔的列表。
ignoreScope	列出方法名称或正则表达式，并从范围参数指定的范围中所包含的程序包、类和方法中排除某些程序包、类和方法。

参数	描述
detail	指定更具体的捕获说明。该参数是以下各项的以逗号分隔的列表： ➤ **caller** – 导致在调用方执行插桩。如果未指定此关键字，则将执行默认插桩，也就是被调用方的插桩。 ➤ **args:n** – 调用 **n-th** 参数的 **toString()** 方法。将在 Diagnostics 控制台的方法参数字段中显示所返回的字符串。可将捕获的字符串用作层参数中的聚合参数。 **n** 的值可以是 1 到 256 之间的任意值。 ➤ **args:0** – 调用当前类实例或被调用方对象上的 **toString()**。静态方法可返回被调用方对象的类名。 ➤ **before:code:<代码密钥>** – 在符合点的方法的字节码起始处插入键中所指定的代码段。代码段运行时，堆栈上的末尾字符串值会在 Diagnostics 控制台的方法参数字段中显示，并且也可在层参数中用作聚合参数。代码密钥参数可指定为代码段生成的安全代码密钥，该代码段之前已由您为点创建。有关代码段的信息，请参阅 “定义具有代码段的点”（第 308 页）；有关代码密钥的信息，请参阅 “确保代码段安全”（第 322 页）。 ➤ **after:code:<代码密钥>** – 在符合点的方法的字节码每个退出点处插入键所指定的代码段。代码段在运行之后，不应在堆栈上留有任何值。

参数	描述
detail （续）	➤ **disabled** – 防止插入到字节码中的插桩报告数据。可使用插桩控制网页来动态启用禁用的点，以使该点开始报告数据。通过使用以下 Profiler URL 可访问此网页： http://< 探测器安装目录 >:< 探测器端口 >/inst/layer。 ➤ **outbound** – 对方法进行标记，以便在“出站调用”屏幕上将其列出。同时还将对此插桩条目的 Diagnostics 参数进行解析，以确定有关出站请求的其他信息是否可在 Diagnostics 控制面板中显示出来。 ➤ **no-correlation** – 与不使用关联支持技术的“出站”点一起使用。 ➤ **method-no-trim** – 表示在执行完由此点使用的方法后，不应发生基于延迟的清理。 ➤ **method-trim** – 表示此点使用的方法的所有调用均应为“已清理”，即未报告。但是，对应代码段通常会产生一些副作用 （如果有）。 ➤ **lifecycle** – 将该插桩点标识为与对象生命周期监控相关。 ➤ **no-layer-recurse** – 禁止录制从此点使用的方法调用的所有方法，除非被调用方属于不同的层。 ➤ **is-statement** – 将调用标记为 java.sql.Statement 类。 ➤ **is-prepare-statement** – 将返回 java.sql.Statement 对象的调用标记为捕获。 ➤ **method-cpu-time** – 导致收集此方法中除延迟之外的 CPU 非独占时间，除非彻底关闭 CPU 收集 (cpu.timestamp.collection.method = 0)。

参数	描述
detail （续）	➤ **condition** – 禁止此点使用的插桩，除非满足指定条件。条件是静态的，由 **inst.properties** （或命令行）**中的 details.conditional.properties 属性定义。** ➤ **when-root-rename** – 每当首先执行此点使用的方法时，指示探测器重命名服务器请求。 ➤ **diag** – 将该点标记为与 HP Diagnostics 相关 （默认）。 ➤ **tv:<键>** – 将该点标记为与 HP Transaction Vision 相关。 ➤ **no-tv** – 将该点标记为与 HP Transaction Vision 冲突。如果将 Transaction Vision 配置为处于活动状态，则会完全禁止此类点使用 Java 代码。 ➤ **add-field:<访问权限>:<类型>:<名称>** – 导致将指定字段添加到已使用的类中。 ➤ **gen-instrument-trace** – 每当将此点用于设备时，将线程堆栈跟踪打印到标准输出上。 ➤ **gen-runtime-trace** – 每当执行由此点使用的方法时，便可将线程堆栈跟踪打印到标准输出上。 ➤ **trace** – 导致每次进入或退出此点所使用的每个方法时，将详细的设备信息打印到 probe.log 中。 ➤ **sub-point:< 键 >** – 在使用时指定其他处理；键必须在 **inst.properties** 中，并且必须标识用于处理的类名。 ➤ **store-thread** – 导致在对应代码段中使用的所有特殊字段均以线程 - 本地数据结构存储。 ➤ **store-fragment** – 导致在对应代码段中使用的所有特殊字段均作为当前服务器请求的属性进行存储。

参数	描述
detail	➤ **store-method –** 导致在对应代码段中使用的所有特殊字段均作为此点所使用方法的调用属性进行存储。 ➤ **ws-operation –** 指定设备条目用于入站 Web 服务调用。同时还将对此设备条目的 Diagnostics 参数进行解析，以确定有关 Web 服务请求的其他信息是否可在 Diagnostics 控制面板中显示出来。
rootRenameTo	一旦 **when-root-rename** 详细信息生效，便可标识服务器请求。
layerType	为某些使用的方法指定特殊处理，并接受以下值： ➤ **method –** 无特殊处理 （默认值）。 ➤ **trended_method –** 标识要在 “已趋势化的方法”视图中显示的方法。 ➤ **Portlet –** 标识用于 “Portal 组件”视图的 portlet 生命周期方法。它们由 HP Diagnostics 设置，不得修改。 ➤ **sql –** 标识用于捕获 SQL 视图的 SQL 的方法。它们由 HP Diagnostics 设置，不得修改。
priority	一旦有多个可应用于给定方法的设备点，并且 Diagnostics Agent 无法自行解决冲突，则点的**优先级**将确定要使用的点。优先考虑较高优先级。默认值为 0。
active	激活或停用某个点。设为 True 时，即激活点。设为 False 时，该点处于非活动状态，且被探测器忽略。

定义具有代码段的点

自定义代码参数可指定要插入到点的字节码中的代码段。当通过调用对象的 **toString()** 方法 （如 **args:n** 参数中所指定）所返回的值不提供有关 Diagnostics 控制台的有用信息，或要求显示已用方法的多个参数时，将使用点中的代码段。

点中的代码段是通过使用该点详细信息参数中的关键字 **before:code:<代码密钥>** 或 **after:code:< 代码密钥 >** 声明的。before 和 after 用于在已用方法之前或之后执行代码段。使用代码密钥参数可防止对代码段进行未授权的修改，因而代码段通常比较安全。可使用任何运行的探测器的代码密钥生成器页面来生成代码密钥参数的值，而且这些值在进行任何 Java 代理安装时均有效。有关代码密钥的详细信息，请参阅 “确保代码段安全”（第 322 页）。

点的实际代码段均会输入到 **< 探测器安装目录 >/etc/code/custom_code.properties** 文件中。然后可使用代码密钥的值将这些段与捕获点文件中的点关联起来。代码段是使用伪 Java 代码创建的，而伪 Java 代码使用的语法与 OGNL 类似。使用代码段后，可从已用字节码向可通过已用方法进行访问的方法执行调用。代码段返回的对象可进行转换，也同样可以执行其方法。代码段必须以字符串或对象结尾，字符串或对象中的 **toString()** 可保留在要解析成字节码的语句的堆栈上。代码段的此末尾字符串用于显示在 Diagnostics 控制台中的返回的参数值。

代码段还可用于直接存储特定碎片的值，也可以存储可用于随后代码段的值。您可以通过一些特殊字段和关键字详细信息（如 **store-fragment** 和 **store-thread**）来使用这些功能。

注意：代码段是一个功能非常强大的工具，可能会对探测器的开销产生影响，所以使用时需小心谨慎。为此，使用时，Diagnostics 要求在探测器使用代码段之前指定代码段以及代码密钥。

本节包括：

使用代码段

要在 **< 探测器安装目录 >/etc/auto_detect.points** 中指定点时使用代码段，请参阅以下详细信息：

```
class      = javax.jms.TopicPublisher
method     = publish
signature  = !\(Ljavax/jms/Topic.*
deep_mode  = soft
layer      = Messaging/JMS/Producer
detail     = outbound,no-correlation,before:code:6d0f3088
```

详细信息参数中的 before:code 条目表示已为该点输入代码段。代码密钥值可保护代码段中的代码，并将该点与实际代码段联系在一起。

与该点关联的代码段必须输入到 **< 探测器安装目录 >/etc/code/custom_code.properties** 中，如以下示例所示：

```
# Used by [JMS-TopicPublisher2]
6d0f3088 = #topic =
@ProbeCodeSnippetHelper@.checkForTempName(#arg1.getTopicName()); \
"DIAG_ARG:type=jms&name=topic:"+ #topic + "&target=topic://" + #topic;
```

使用代码密钥的值将代码段与捕获点文件中的点关联起来。

代码段语法

下面介绍创建代码段所必须使用的语法。

➤ 文本

代码段中仅支持以下文本类型。

文本类型	语法示例
字符串	" 一个字符串 "
布尔	true、 false
整数	42
null 常量	null
无类型且无值的常量	void

➤ **字符串串联**

代码段中支持基本字符串串联。

串联类型	语法示例
两个字符串	" 一个字符串 " + " 其他字符串 "
一个字符串和一个文本	" 一个字符串 " + 42

➤ **本地成员**

默认的本地成员提供了允许代码段引用传递到已用方法的当前实例或对象的方法。这些本地成员可从这些引用中调用方法或检索值。

变量	方法
#callee	引用实例方法的被调用方对象。相当于 Java 的 “this” 引用。引用静态方法时，不得使用此变量。
#arg1, #arg2, ..., #argN	引用被调用方方法调用的参数。
#return	引用 after 代码段的方法结束后的返回值。
#classloader	保留以供 HP 软件内部使用。

注意：某些设备点支持 “特殊” 变量引用。例如，**CLApplicationDiscoveryPoint** 支持 #classloader 变量。

➤ DIAG_ARG 字符串

代码段允许将构建单个 DIAG_ARG 值的一系列值串联起来。此值允许通过在一个 DIAG_ ARG 格式的字符串中返回某特定类型的所有数据，来使用某些常见类型的支持数据，如 Web 服务和 JMS。

类型	字段 （必需）	定义
ws	&ws_name	Web 服务名称
	&ws_op	Web 服务操作名称
	&ws_ns	Web 服务名称空间
	&ws_port （仅限入站）	Web 服务端口名称
	&target （仅限出站）	出站 Web 服务目标
jms	&name	队列或主题名称
	&target	目标队列或主题名称

DIAG_ARG 字符串的格式包括一些类型字段，以及串联成一个字符串的值（本地变量），如下所示：

```
"DIAG_ARG:type=ws&ws_name="+ #servicename +"&ws_op="+ #operation +\
"&ws_ns="+ #ns +"&ws_port="+ #port;
```

不得将 DIAG_ARG 字符串与 Web 服务入站数据的 store-fragment 特殊字段（以 ##WS_inbound_* 开头的特殊字段）一起使用。请仅使用这两者之一来收集 Web 服务入站数据。

➤ **特殊字段 (store-fragment)**

默认特殊字段提供一种便捷的方式，供代码段传递与常见事件的碎片相关的数据。此机制可补充现有事件，但无法替换它们。碎片本地存储比自定义事件的开销成本更高。以下变量必须与 **store-fragment** 详细信息设置一起使用。

变量	方法
##WS_consumer_id	存储特定碎片的用户 ID。
##WS_SOAP_fault_code	存储 SOAP 错误代码。
##WS_SOAP_fault_reason	存储 SOAP 错误原因。
##WS_SOAP_fault_detail	存储 SOAP 错误详细信息。
##WS_inbound_service_name	存储入站 Web 服务名称。
##WS_inbound_operation_name	存储入站 Web 服务操作名称。
##WS_inbound_target_namespace	存储入站 Web 服务目标名称空间。
##WS_inbound_port_name	存储入站 Web 服务端口名称。

➤ **特殊字段 (store-thread)**

此外，特殊字段还提供了一种便捷的方式，供代码段存储线程寿命的相关数据。请小心使用这些线程本地存储变量，因为它们会导致开销。应仅将其与 store-thread 详细信息设置一起使用。

您可以通过使用 getThreadContextValue(" 字符串值 ") 或 getAndRemoveThreadContextValue(" 字符串值 ") 方法来调用探测器的 ThreadContextProxy 类，以便在随后代码段中检索这些变量，并将 " 字符串值 " 作为变量的名称，而无需在前面加上 ## 符号。最后一个检索的值应始终调用 getAndRemoveThreadContextValue(" 字符串值 ")，以便从内存中清除该值并删除下一个线程的值。

变量	方法
##SOAPHandler_wsname	存储 Web 服务名称，以便随后供 SOAP 处理程序使用。
##< 任意字符串 >	存储任意值，以便随后在接下来的代码段中进行检索。

➤ **类引用和静态成员**

通过使用 @ 符号预挂起类以将其标识为 “静态”，并按如下示例所示用 @ 符号标记被访问的方法，可以访问静态成员 / 方法：

```
@java.lang.System@.out ("Hello World");
```

```
@com.mercury.diagnostics.capture.metrics.countingCollector@.incrementCounter();
```

当 Java 类两边的标记可由分析程序获取时，代码段中的参数便支持 Java 类语法。以下示例展示如何使用 @ 符号作为标记：

```
@java.lang.System@
```

```
@java.lang.System@out (Static field)
```

代码段帮助程序

您很难甚至无法使用代码段中的有限语法对某些功能进行编码。

因此，代码段环境提供了两个帮助程序类：ProbeCodeSnippetHelper 和 ProbeCodeSnippetHelperV5。 CodeSnippetHelperV5 使用某些仅在 Java 5 或更高版本中可用的 API。

下面显示 ProbeCodeSnippetHelper 功能。

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 */

package com.mercury.opal.capture.proxy;

/**
 * Used to help out Code Snippets
 */
public class ProbeCodeSnippetHelper {

  /**
   * When a Special Field holds a reference to the string below,
   * it will not be stored in the Fragment Local Storage,
   * or Invocation Local Storage
   */
  public static final String DO_NOT_STORE = ...

  /**
   * Helper to convert an int to an Integer
   * @param i
   * @return a new Integer object having the value of i
   */
  public static Object intToInteger(int i) {
    ...
  }

  /*
   * Mark the current thread, if not marked yet
   * @return true, if and only if the thread had been already marked
   */
  public static boolean testAndSetRecursiveFlag() {
    ...
  }

  /*
   * Unmark the current thread
   */
  public static void clearRecursiveFlag() {
    ...
  }

  /**
   * Helper method to call ResourceBundle.getString() and catch any exceptions that
   *  might be thrown
   * @param theBundle the ResourceBundle on which to call getString
   * @param key the key to pass getString
   * @return the value returned from getString, or null if an exception occurred
   */
  public static String getStringFromResourceBundle(ResourceBundle theBundle, String key) {
    ...
  }
```

```
    /*
    * Helper methods to allow our cross-vm coloring to piggyback ride across
    * the custom outbound calls in which the application passes [only] a String.
    * The actual transport technology is irrelevant.
    * Instead of sending the original message, a composite message ("envelope")
    * will be passed. The composite message includes both the original message
    * and Diagnostics Probe ENCODED cross-vm coloring.
    * On the receiving end, the composite message will be received, but only
    * the original message will be passed to the application, and the coloring
    * will be retained by the probe.
    */

   /**
    * Create a composite message, given the coloring and the original message.
    * @param coloring - the correlation String obtained via the ENCODED coloring,
    * may be null
    * @param originalMessage - the original messsage sent by the application
    * @return - the composite message, null if and only if the originalMessage is null
    */
   public static String createDiagEnvelope(String coloring, String originalMessage) {
      ...
   }


   /**
    * Extract the coloring from the composite message (envelope).
    * @param envelope - the composite message or the original message
    * @return the coloring as created on the sender side, or null if not present
    */
   public static String extractColoringFromDiagEnvelope(String envelope) {
      ...
   }


   /**
    * Extract the original message from the composite message (envelope).
    * Works properly, even if the sender side has not been instrumented, and
    * there's no envelope.
    * @param envelope - the composite message or the original message
    * @return the original message (before coloring)
    */
   public static String extractOriginalMessageFromDiagEnvelope(String envelope) {
      ...
   }


}
```

下面显示 ProbeCodeSnippetHelperV5 功能。

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 */

package com.mercury.opal.capture.jdk15.agent;

/**
 * Used to help out Code Snippets using Java 5 or later
 */
public class ProbeCodeSnippetHelperV5 {

  /**
   * Get the annotation of the specified type from the class or its superclass,
   * or its implemented interfaces
   * @param theClass  The class to get the annotation for
   * @param annClass  The annotation class to look for
   * @return
   */
  public static Object getEndpointClassAnnotation(Class theClass, Class annClass) {
    ...
  }

  /**
   * Get the method annotation of the specified type from the class
   * or its superclass, or its implemented interfaces
   * @param theClass  the class
   * @param methodName  the method name
   * @param argCount  the argument count
   * @param annClass  the class annotation type
   * @param methodAnnClass  the method annotation type
   * @return
   */
  public static Object getEndpointMethodAnnotation(Class theClass, String methodName,
      String argCount, Class annClass, Class methodAnnClass) {
    ...
  }

  /**
   * Helper method to get an annotation element value. If the annotation
   * does not have the element, return null.
   * @param annClass   The class of the annotation
   * @param instance   The annotation instance object
   * @param elementName  The element name
   * @return  The element value for the annotation instance, or null
   */
  public static String getAnnotationElementValue(Class annClass, Object instance, String elementName) {
    ...
  }

  /**
   * This helper method is used to serialize a DOM document.
   * This method uses APIs available in DOM Level 3 or newer, which are
   * available with a 1.5 or later JVM.
   * @param document
   * @return The serialized form (XML) of the input DOM document
   */
  public static String serializeDOMToString(Document document) {
    ...
  }

}
```

➤ 使用方法调用的堆栈跨越多行

代码段中的方法调用的堆栈可跨越多行。如果构建字节码的分析程序必须继续分析语句的堆栈，则在每次回车前都需要添加一个“\”（反斜杠）。语句的代码段堆栈的最后一行不用包含反斜杠，只需在结束时回车即可。

```
@java.lang.System@.out ("Hello World");\
"Callee Name="+#callee.getName().toString();
```

➤ 转换

调用返回对象的方法时，通常需要进行转换，以在返回的对象上调用成员。对象参考支持转换。要将对象转换成其他类型，请根据对该对象的参考在符号“<”和“>”之间放置转换引用。以下是一些转换示例。

```
#arg1<com.myCompany.myFoo>.myMethod();

This is equivalent to the Java statement:

((com.myCompany.myFoo)arg1).myMethod();
```

```
@some.class.Foo@foo<com.myCompany.myFoo>.myMethod();

Would be equivalent to the java statement:

((com.MyCompany.myFoo)some.class.Foo.foo).doSomething();
```

```
#foo = #arg1<bar>.b(); #foo.toString();

Creates the following java equivalent:

String foo = ((Bar)arg1).b(); ((Object)foo).toString();
```

注意：某些特殊类型 （如 #classloader）不支持转换。

- **方法调用**

  段参数中可包括方法调用。方法调用的支持功能包括一些使用或不使用参数和方法链接的调用。以下是包括在代码段参数中的方法调用的示例：

```
#arg1.toString()
```

```
#arg2.getSomething().getSomethingElse()
```

```
#callee.getSomething("foo", #arg1).somethingElse()
```

```
@some.Class@.staticMethod()
```

  仍需在静态引用后面显示点，以便正确解析方法调用。

```
@java.lang.System@out.println("Here I am!")
```

  要加速生成运行时字节码（通过避免反射），您可以指定从方法返回的类型，如以下示例所示：

```
#arg1.getSomething()<some.class.Here>
```

  如果该方法获取了参数，或如果使用了静态字段，则此操作将无效。

- **多个语句**

  代码段可在单个代码段中包括多个语句。这对希望在堆栈上保留多个对象的设备（如 **CLApplicationDiscoveryPoint**）而言非常必要。在其他情况下也同样很方便。

```
@java.lang.System@out.println("Look out!");
#arg2.getSomething();
```

➤ **本地成员分配**

除了默认支持的 “local” 变量以外，您还可以创建自己的本地成员，以获取由调用的方法返回的对象参考。

要创建新的本地成员，请在本地成员的名称前输入 “#” 符号。分析程序可为您创建本地成员。

```
#myBar = #arg2.getName();\
#myUpperBar = #myBar.toUpper();\
"Target Name=http://"+myUpperBar+"/services";
```

➤ **特殊字段分配 (store-fragment)**

您可以使用预定义的特殊字段来存储调用的方法返回的对象参考。在特殊字段的名称前输入 “##” 符号，以及设备点上的 **store-fragment** 详细信息关键字。

```
##WS_SOAP_fault_code = #arg2;\
##WS_SOAP_fault_reason = #arg3;\
##WS_SOAP_fault_detail = (#arg4 == null ? null : #arg4.toString());"";
```

➤ **特殊字段分配 (store-thread)**

您可以使用特殊字段来存储调用的方法返回的对象参考。在特殊字段的名称前输入 “##” 符号，以及设备点上的 **store-thread** 详细信息关键字。

```
# Used by [SOA_Broker_Payload_Handler]
##SOA_Manager_Inbound_Payload=#callee.getRequestDocument();"";
```

在随后的代码段中，您可以通过使用以上开头无 ## 符号的特殊字段值来调用 getThreadContextValue，从而检索存储的值。

```
#temp_soam_payload=@com.mercury.opal.capture.proxy.ThreadContextProxy@.get
ThreadContextValue("SOA_Manager_Inbound_Payload");
```

在随后的代码段中，您可以通过使用以上开头无 ## 符号的相同值来调用 getAndRemoveThreadContextValue 方法，从而检索并删除存储的特殊字段值。调用 getAndRemoveThreadContextValue 释放内存并为下次出现的情况清除障碍，这一点非常重要。

```
#temp_soam_payload=@com.mercury.opal.capture.proxy.ThreadContextProxy@.
getAndRemoveThreadContextValue( ("SOA_Manager_Inbound_Payload");
```

- **条件逻辑**

  代码段语法允许相当于 Java if-else 语句的有条件逻辑。此语法允许您使用 == 和 != 运算符来比较相同类型或整型或布尔型基元的对象参考。使用此语法的文本值和其他基元比较均无效。

  下面是如何比较参考的示例：

  ```
  (value1 == value2 ? <if_True_codeSnippet>:<if_False_codeSnippet>)
  ```

  下面是在调用方法之前如何验证对象不为 Null 的示例：

  ```
  (#arg1 == null ? "Unknown" : #arg1.getSomething())
  ```

  这相当于以下 Java 语句：

  ```
  if (arg1==null) return "Unknown" else return arg1.getSomething();
  ```

- 异常处理

  以下语法提供了有限形式的异常处理：

  ```
  !{<code-snippet-code>}!
  ```

  已执行了指定的代码，并且以上表达式的值为抛出的异常，或如果在执行代码期间无任何异常，则值为 Null。

确保代码段安全

默认情况下，使用时，必须在探测器使用代码段之前指定有效的代码密钥以及代码段。指定代码密钥可防止意外引入可能会大幅度增加探测器开销的插桩。

生成代码密钥时，Diagnostics 会检查代码段的语法，确保在生成键之前该语法有效。当 Diagnostics 使用应用程序时，它会检查针对代码密钥参数而输入的值，确保该值与为点的代码段计算的代码密钥相匹配。如果代码密钥不匹配，则 Diagnostics 将忽略代码段，并且不会创建插桩点。

生成代码段代码密钥

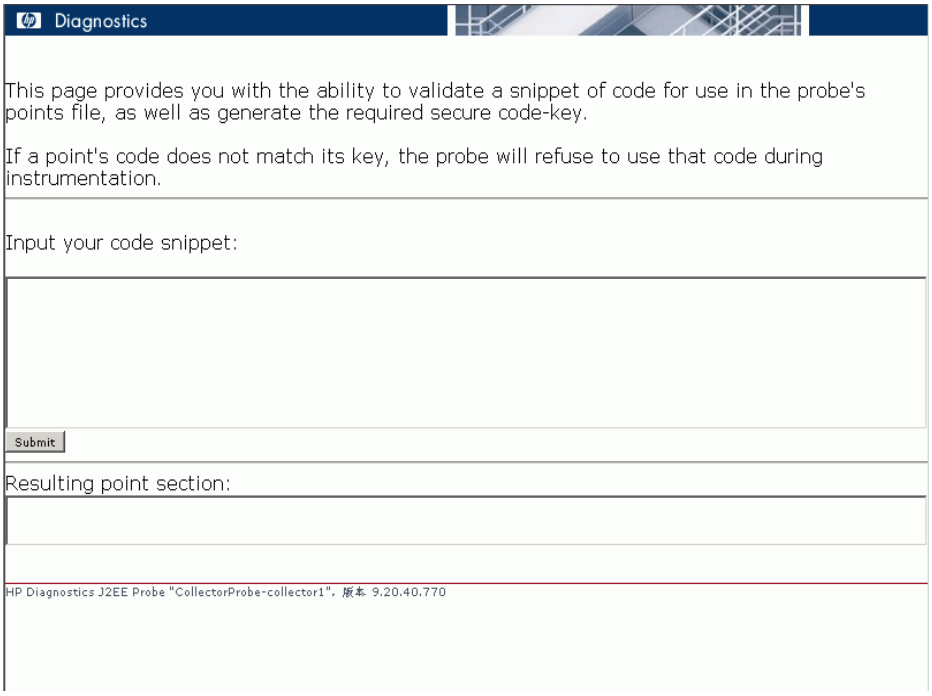
您可以使用工具来安装 Java 代理，该工具可从您所输入的代码段生成代码密钥。

要生成代码密钥，请执行以下操作：

1 在浏览器中打开 URL：

http://< 探测器主机 >:< 探测器端口 >/inst/code-key

此时 Diagnostics 将显示一个页面，您可以在该页面中验证代码段语法和生成代码密钥，如以下示例所示：



2 将在 **auto_detect.points** 文件的代码参数中指定的代码段输入到 “Input your code snippet” 文本框中，然后单击 “Submit”。

注意：代码段必须包括 code = 参数名称后面的所有文本。

---

3 Diagnostics 会在“Resulting point section”文本框中显示验证代码段和生成代码密钥的操作的结果。

如果代码段有效，则 Diagnostics 将同时显示代码密钥和代码参数的值。在捕获点文件中输入这些值。

如果代码段无效，则 Diagnostics 将显示一条错误消息，指出所检测到的问题。更正问题，然后再次单击 “Submit”，验证更正后的代码。

禁用代码密钥安全检查

默认情况下，Diagnostics 会验证代码密钥参数的值是否与其在插桩应用程序时所生成的值相匹配。通过将值为 false 的 **require.code.security.key** 属性插入到 **<探测器安装目录>/etc/inst.properties** 文件中，可以禁用此安全检查。

注意：在使用此属性时应格外小心。如果禁用此检查，则可能会引起意外的处理开销并造成不可预知的性能监控结果。

控制类映射捕获

类映射允许 Diagnostics 提供有关服务器请求所调用的类和方法的详细信息。此信息可帮助您缩小对导致性能问题的原因的搜索范围，并帮助您正确使用应用程序代码。创建映射时会在代理主机系统上产生额外的开销，这就是使用类映射时所需的成本。

默认情况下，属性 **use.class.map=false** 在 **probe.properties** 文件中设置。可将其更改为 **true** 后，以提供类映射。

插桩示例

本节中的示例说明如何通过创建和修改捕获点文件中的点，来自定义应用程序的插桩。

本节包括以下示例：

- 自定义层和子层
- 通配符方法
- 忽略指定的方法
- 已趋势化的方法视图的捕获方法
- 仅捕获类中的特定方法
- 捕获可返回字符串的特定方法
- 在控制的范围内进行捕获
- 硬和软 deep_mode
- 参数捕获
- 入站和出站 Web 服务
- 重命名根方法
- 向类中添加字段
- 将属性传递到实例树
- 根据访问标记筛选方法
- 不记录直接递归
- 执行调用方插桩
- 配置分配分析
- 配置轻量内存诊断 (LWMD)
- 配置收集泄漏定位
- 启用 JDBC 结果集的对象生命周期监控
- 添加已禁用的点并在运行时启用它
- 指定应不清理的方法

- 指定应始终清理的方法
- 为方法启用 CPU 时间收集
- 基于 SAP JCO 库版本更改 SAP RFC 插桩
- 输出点的插桩和运行时信息 （仅限调试）

自定义层和子层

- 以下点会为 myCompany.myFoo 类中的 myMethod 方法在名为 “FOO” 的层中创建一个名为 “BAR” 的自定义子层：

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
signature = !.*
layer = FOO/BAR
```

通配符方法

- 以下点可捕获 MyCompany.MyFoo 类中的所有方法：

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
signature = !.*
layer = FOO/BAR
```

忽略指定的方法

- 以下点可捕获 MyCompany.MyFoo 类中除 setHomeInterface 和 getHomeInterface 方法外的所有方法：

```
[myCompany.myFoo_AllMethodsExcept]
class = myCompany.myFoo
method = !.*
ignoreMethod = !setHomeInterface.*, !getHomeInterface.*
signature = !.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany 程序包 / 名称空间中的所有方法，但包含在 MyCompany.logging 类中的方法除外：

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
ignore_cl = MyCompany.logging
signature = !.*
layer = FOO/BAR
```

已趋势化的方法视图的捕获方法

➤ 以下点可捕获所需数据以填充 myMethod 方法的已趋势化的方法视图：

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
signature = !.*
layer = FOO/BAR
layertype = trended_method
```

仅捕获类中的特定方法

➤ 以下点可捕获 MyCompany.MyFoo 类的构造函数中的所有方法：

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = <init>
signature = !.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类的单个构造函数中的所有方法：

```
[myCompany.myFoo_Singleton]
class = myCompany.myFoo
method = <clinit>
signature = !.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类中的 setFoo 方法：

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
signature = !.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类中的所有 “set” 方法：

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !set.*
signature = !.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany 程序包 / 名称空间中的所有方法：

```
[myCompany_All_Methods]
class = !myCompany\..*
method = !.*
signature = !.*
layer = FOO/BAR
```

捕获可返回字符串的特定方法

➤ 以下点可捕获 MyCompany.MyFoo 类中可返回 java.lang.String 并且不包含参数的 getFoo 方法：

```
[myCompany.myFoo_GetFoo_String]
class = myCompany.myFoo
method = getFoo
signature = ()Ljava\lang\String
layer = FOO/BAR
```

在控制的范围内进行捕获

➤ 以下点可捕获 MyCompany 程序包 / 名称空间中从 MyCompany.logging 类调用的所有方法。有关更多详细信息，请参阅 “使用调用方插桩”（第 343 页）。

```
[myCompany_All_Methods_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
signature = !.*
scope = MyCompany.logging
layer = FOO/BAR
```

➤ ignoreScope 参数用于从 scope 参数指定的范围中排除某些特定程序包、类和方法。以下点可捕获 MyCompany 程序包 / 名称空间中从 MyCompany.logging 类调用的所有方法，但从 myMethod 方法调用的方法除外。有关更多详细信息，请参阅 “使用调用方插桩”（第 343 页）。

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
signature = !.*
scope = MyCompany.logging
ignoreScope = MyCompany.logging\myMethod
layer = FOO/BAR
```

硬和软 deep_mode

➤ 以下接口定义用于软和硬 deep_mode 示例：

```
public interface Interface1 {

    public void callerMethod();

}
```

➤ 以下类用于软和硬 deep_mode 示例：

```
public class Class1 implements Interface1 {
   public void callerMethod(){
      calleeMethod();
      calleeMethod2();
   }

   public void calleeMethod(){
      System.out.println("hello world");
  //more code lines here…
   }

   public void calleeMethod2(){
      System.out.println("hello world 2");
   }
}
```

➤ 以下点捕获 Class1 类中的 “callerMethod”：

```
[Training-1]
class      = Interface1
method     = !.*
signature  = !.*
deep_mode  = soft
layer      = Training
```

➤ 以下点可捕获类 1 中的所有方法，如 “callerMethod”、“calleeMethod1”和“calleeMethod2”：

```
[Training-1]
class      = Interface1
method     = !.*
signature  = !.*
deep_mode  = hard
layer      = Training
```

参数捕获

Diagnostics 中显示的参数是代码段在堆栈上留下的最後一个字符串。代码段必须以字符串或对象结尾，可将其中的 toString() 保留在要解析成字节码的语句的堆栈上。

重要信息：在使用参数捕获时要极其小心。除非所捕获参数的所有可能值集有限，否则代理将耗尽 Java 堆空间。

➤ 假设您插桩一个名为 myCompany.myFoo.myMethod() 的方法，而 myFoo 包含另一个名为 getComponentName() 的返回字符串的方法。以下示例将在 Diagnostics 中以参数形式显示 getComponentName() 的结果 （在此示例中，#callee 指实例方法的被调用方对象）。

```
[myCompany_componentName_as_argument]
class = myCompany.myFoo
method = myMethod
signature = !.*
detail = before:code: 8d2509eb
layer = FOO/BAR
```

custom_code.properties 文件中的代码段如下：

```
8d2509eb = #callee.getComponentName()
```

- 以下点可捕获 myMethod 的第一个参数，并在 Diagnostics 中将其显示为捕获的参数。同时，它还会将其用作子层名称。可通过在层中包括 ${ARG} 来实现上述目的。在此示例中，如果捕获的参数（此示例中指 myMethod 的第一个参数）具有值 myArg，则层为 FOO/myArg。

```
[myCompany_capture_firstArg_and_also_show_as_layer]
class = myCompany.myFoo
method = myMethod
signature = !.*
detail = before:code: 358f05d6
layer = FOO/${ARG}
```

custom_code.properties 文件中的代码段如下。如果使用 #arg2，则将捕获第二个参数。

```
358f05d6 = #arg1.toString()
```

入站和出站 Web 服务

当点中的详细信息参数包含“出站”或“ws-operation”关键字时，Diagnostics 会尝试解析代码段堆栈上的最後一个字符串，以便显示有关方法调用的其他信息。

- 对于入站 Web 服务（必须使用“ws-operation”详细信息），该字符串结构以下：

```
"DIAG_ARG:type=ws&ws_name=”+<WebServiceName>+"&ws_op=”+
<OperationName>+”&ws_ns=”+<TargetNameSpace>+”&wsOport=”+<wsPort>
```

- 对于出站 Web 服务（必须使用“出站”详细信息），该字符串结构如下：

```
"DIAG_ARG:type=ws&ws_name=”+<WebServiceName>+"&ws_op=”+
<OperationName>+”&target=”+<TargetName>
```

以下是一个示例：

```
class      = weblogic.wsee.ws.WsStub
method     = invoke
signature  = (Ljava/lang/String;Ljava/lang/String;Ljava/util/Map;Ljava/util/Map;)Ljava/
lang/Object;
layer      = Web Services
detail     = outbound,before:code:edd75e36
```

custom_code.properties 文件中的代码段如下：

```
edd75e36 = #service = #callee.getService().getWsdlService();\
#qname = #service.getName();\
"DIAG_ARG:type=ws&ws_name="+ #qname.getLocalPart() +"&ws_op="+ \
#callee.getMethod(#arg1).getOperationName().getLocalPart() +"&target="+ \
#callee.getProperty("javax.xml.rpc.service.endpoint.address");
```

重命名根方法

- 请注意以下点：

```
class      = Statement
method   = execute
layer      = Database/JDBC/Execute
detail     = when-root-rename
rootRenameTo   = mySuffix
```

如果此类方法以根方法结尾，则此类服务器请求的名称为 Background-mySuffix，服务器请求的类型为 RootRename。

- 请注意以下点：

```
class      = Statement
method   = execute
layer      = Database/JDBC/Execute
detail     = when-root-rename
```

请注意，这里忽略了 rootRenameTo 属性。此类服务器请求的名称为 Background-Database （因为 Database 是第一个子层），服务器请求的类型仍为 RootRename。

向类中添加字段

➤ 请注意以下点：

```
class      = com.corp.Foo
method   = bar
detail      = add-field:protected:Object:serviceName
```

其中的 detail 可导致将以下一个字段和两个公用 setter/getter 方法添加到类 com.corp.Foo 中：

```
protected transient Object serviceName
public void _diag_set_serviceName(Object arg)
public Object _diag_get_serviceName()
```

将属性传递到实例树

➤ 以下示例可将 my_attribute 名称附加到每个捕获的 com.corp.Foo.bar() 实例。

具有前缀 display_ 的名称及其对应值将显示在调用配置文件中。

```
class      = com.corp.Foo
method   = bar
detail      = store-method,code:f59f0c5c
```

代码段：

```
f59f0c5c = ##my_attribute="value-of-my-attribute";"";
```

根据访问标记筛选方法

➤ 以下示例将插桩类 com.corp.Foo 中除静态方法以外的所有方法。

```
class      = com.corp.Foo
method   = !.*
signature       = !.*
method_access_filter = static
```

不记录直接递归

➤ 在以下示例中，如果方法 com.corp.Foo.bar 调用其自身（或同一层中的任何项），则不会录制第二个调用。这是由 **detail = no-layer-recurse** 导致的结果。

但是，这仅适用于直接递归。如果 com.corp.Foo.bar 再次从调用此方法的其他层调用已插桩的方法，则会记录所有方法。

```
class     = com.corp.Foo
method   = bar
layer      = Example/MyBar
detail    = no-layer-recurse
```

执行调用方插桩

➤ 以下点可导致执行调用方插桩（与默认的被调用方插桩相反）。这是 **detail = caller** 的结果。

执行调用方一端插桩的另一种方式是使用 “范围”属性，如 “使用调用方插桩”（第 343 页）中所述。

```
class     = com.corp.Foo
method   = bar
detail    = caller
```

配置分配分析

下面的两个示例均跟踪程序包 com.mycompany.mycomponent 中 java.lang.Integer 的分配，但存在两个不同之处：

➤ 在第一个示例 **(detail = leak)** 中，将对跟踪过程进行管理。跟踪过程将在用户单击 Profiler 中的 “开始”时启动，在用户单击 “停止”时停止。在第二个示例 **(detail = deallocation)** 中，跟踪过程会随应用程序一起启动。

➤ 在第一个示例中，当点达到常规插桩时便会禁用该点。这意味着将不会在实例树中显示 “新整数”。但在第二个示例中，则会显示。

示例 1 - 受管。当用户单击 Profiler 中的“开始”时便启动跟踪，当用户单击“停止”时便停止跟踪：

```
[Leak]
scope    = !com\.mycompany\.mycomponent\..*
class     = java.lang.Integer
keyword = allocation
detail   = leak
active    = true
```

示例 2 - 未受管。跟踪随应用程序一起启动：

```
[Leak]
scope    = !com\.mycompany\.mycomponent\..*
class     = java.lang.Integer
keyword = allocation
detail   = deallocation
active    = true
```

以上两个点均不捕获反映的分配。要启用反映分配捕获，只需将详细信息“reflection”附加到点 (**detail = leak,reflection**) 中即可。

配置轻量内存诊断 (LWMD)

➤ 以下示例将为 com.mercury.mycomponent 程序包内部发生的收集打开收集诊断。可以在 **auto_detect.points** 文件中找到此示例。默认情况下，会将其设置为 active = false。

```
[Light-Weight Memory Diagnostics]
scope    = !com\.mycompany\.mycomponent\..*
class     = java.lang.Integer
keyword = lwmd
active    = true
```

您还需要设置 **dynamic.properties** 文件中的 **lwm.diagnostics.capture=true** 属性。有关详细信息，请参阅《HP Diagnostics User's Guide》中有关“收集和资源视图”的章节。

配置收集泄漏定位

不论 JRE 是何种版本，如果要在 Java 代理中使用收集泄漏定位 (CLP) 功能，就必须使用合适的应用程序服务器模式运行 JRE Instrumenter。有关插桩 JRE 的详细信息，请参阅第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”。

默认情况下，将在 **auto_detect.points** 文件中启用收集泄漏定位。

```
[Collection Leak Pinpointing]
keyword = clp
```

可以在 **dynamic.properties** 文件中设置，以下属性以配置收集泄漏报告。此外，也可以在 Java Profiler 的“配置”选项卡 UI 中设置这些相同值（请参阅 “启用和配置收集泄漏报告”（第 502 页））。

clp.diagnostics.reporting=true

在 Diagnostics UI 中启用报告。可以通过在报告功能对应的 UI 中取消选中复选框来禁用此功能。

clp.diagnostics.growth.time.threshold.flag = 60m

收集大小增长的时间段的阈值。如果收集的大小增长周期超过此阈值，则探测器会将此收集标记为内存泄漏。为避免误报，此值应大于应用程序完全初始化其所有缓存所需的时间。

clp.diagnostics.nongrowth.time.threshold.unflag = 60m

对于已标记的泄漏收集，如果其大小在此阈值时间段内停止持续增长，则探测器将取消其泄漏标记。

启用 JDBC 结果集的对象生命周期监控

一些预配置的插桩点允许进行对象生命周期监控，但默认情况下处于禁用状态。下面的示例将显示其中两个插桩点。

此示例显示如何启用 JDBC 结果集的对象生命周期监控。有关对象生命周期监控的详细讨论，请参阅《HP Diagnostics User's Guide》的“Analyzing Memory and Object Lifecycle”一章中有关分配 / 生命周期分析选项卡的部分。

在此示例中，需要以下两个操作：

1 转至 **inst.properties**，找到 details.conditional.properties。设置 mercury.enable.resourcemonitor.jdbcResultSet=true

2 指定对应打开插桩点中的范围 （如下所示）。

在下面的示例中，探测器在程序包 com.mycompany.mycomponent 内部执行 JDBC 结果集的对象生命周期监控。

```
[Lifecycle-JDBC-ResultSet-Open]
scope    = !com\.mycompany\.mycomponent\..*
class      = java.sql.Statement
method  = !(getResultSet.*)|(executeQuery.*)
signature  = !.*\)Ljava/sql/.*ResultSet;
detail = condition:mercury.enable.resourcemonitor.jdbcResultSet,lifecycle,caller

[Lifecycle-JDBC-ResultSet-Close]
class      =
!(java\.sql\.ResultSet)|(weblogic\.jdbc\.wrapper\.ResultSet)|(com\.ibm\.ws\.rsadapter\.jd
bc\.WSJdbcResultSet)
method  = !(close)|(closeWrapper)
signature  = !.*
deep_mode  = soft
detail =
condition:mercury.enable.resourcemonitor.jdbcResultSet,before:code:513a2b36,metho
d-trim
```

添加已禁用的点并在运行时启用它

➤ 在以下示例中，点已禁用。但这并不表示不能执行插桩。已进行过插桩，但未收集任何数据。这会大幅度降低此类点的运行时开销。

要在应用程序运行时启用数据收集，请转至 http://< 探测器主机 >:< 探测器端口 >/inst/layer 中的层页，或从 Profiler 选择 “配置” 选项卡，接着选择 “查看” 插桩，然后查找 **myLayer** 层，并单击 “启用”。

```
[My Example]
class      = Example
method = !.*
layer   = myLayer
detail    = disabled
```

如果不希望进行插桩，请使用 **active=false**。但是，不能在运行时启用此类点。

指定应不清理的方法

➤ 在以下示例中，不会向 Example.myMethod() 应用延迟清理。

```
[My Example]
class      = Example
method = myMethod
detail    = method-no-trim
```

指定应始终清理的方法

➤ 在以下示例中，不会报告方法 Example.myMethod()。但是，将始终执行与该点关联的任何代码段。

```
[My Example]
class      = Example
method = myMethod
detail    = method-trim, before:code:...
```

为方法启用 CPU 时间收集

➤ 在以下示例中，详细“method-cpu-time”可导致为方法 Example.myMethod() 收集 CPU 时间。

```
[My Example]
class      = Example
method = myMethod
detail    = method-cpu-time
```

基于 SAP JCO 库版本更改 SAP RFC 插桩

在 **< 探测器安装目录 >/etc/inst.properties** 文件内，有两个根据所用 SAP JCO 版本定义的点。注释掉未使用的版本。从版本 2.1.10 或更高版本，使用 com.mercury.opal.capture.inst.SapRfcinstrumentationPoint2_1_10。否则，默认设置将适用于版本 2.1.9 或更早的版本。

输出点的插桩和运行时信息 （仅限调试）

以下示例将在标准输出和 probe.log 中输出几条调试信息。

➤ 每当将此点用于插桩方法时， **gen-instrument-trace** 详细信息便可导致将线程堆栈跟踪输出到标准输出上。

➤ 每当运行 Example.myMethod() 时，**gen-runtime-trace** 便可导致将线程堆栈跟踪输出到标准输出上。

➤ 每当运行 Example.myMethod() 时，**trace** 详细信息便可导致将详细插桩信息输出到 probe.log 中。

```
[My Example]
class      = Example
method = myMethod
detail    = gen-instrument-trace, gen-runtime-trace, trace
```

了解自定义插桩的开销

创建自定义插桩时，注意不要过度插桩应用程序，因为这样会导致被探测的应用程序产生过多的延迟。当插桩的类越来越多时， classloader 延迟便会增加，从而导致过多延迟。自定义插桩对方法延迟或 CPU 开销并不会产生这样的影响，因为每个方法的插桩开销几乎是固定的，这是由于字节码数量几乎总是相同。这意味着根据方法运行所耗用的时间长短不同，CPU 和延迟开销的物理百分比也会按比例地发生变化。

例如，如果方法耗用的时间是 100 毫秒，而插桩运行方法所耗用的时间是 101 毫秒，则开销是 1%。如果方法耗用的时间是 10 毫秒，而插桩将其响应的时间更改为 11 毫秒，则开销是 10%。如果调用此方法的频率不高，则它对应用程序的总体延迟影响很小。但是，频繁调用的使用方法的总体延迟可能会对应用程序的响应延迟产生影响，即使其开销百分比更小。

与传统的探测器不同， HP Diagnostics 使用的是字节码插桩。这样便可选择默认插桩，从而将插桩造成的开销降至最低，平均为 3-5%。对于具有较高插桩延迟开销的方法，仅当在它们相对于应用程序中的其他组件较少地被调用时，并且当插桩可提供分类活动 （如 JNDI 查找）所需的特定信息时，才会插桩它们。

在自定义应用程序插桩时，还应当考虑 Diagnostics 数据开销。使用的方法越多，探测器需要序列化并通过网络传递到 Diagnostics 服务器的数据就越多。您可以调整 Java 探测器的默认配置，以便它可以调整 Diagnostics 的数据量，从而避免对受监控系统的性能造成不必要的影响。探测器调整方法不当可能会在安装 Java 代理的物理计算机上产生 CPU、内存和网络开销。有关管理延迟、 CPU、内存和网络开销的详细信息，请参阅第 12 章，“高级 Java 代理与应用程序服务器配置”。

基于每层的插桩控制

默认情况下，在捕获点文件中定义的层处于启用状态。如果点中包含 details=disabled 参数，则会在启动探测器时禁用层。

JDK 1.5 中的类映射提供一种通过 JVMTI 接口动态插桩方法和类，而无需重新启动 JVM 实例的功能。所有其他虚拟机均要求重新启动 JVM 实例，以将所做的更改应用到捕获点文件。

一旦在方法中放置了插桩，便可基于每层对其数据收集以及运行的 CPU 和方法延迟开销进行控制 （请参阅下面的 “插桩的层” 页面）。

您可以从 URL

http://< 探测主机 >:< 探测端口 >/inst/layer 访问 “Instrumented Layers” 页面。

要在 “Instrumented Layers” 页面中禁用某个层，请在页面中单击与该层关联的 “Disable” 链接。该层将被禁用，并且链接将切换成 “Enabled”，这样便可在需要时再次启用该层。

高级插桩示例

本节包括：

使用调用方插桩

默认情况下，Diagnostics 中的所有插桩均是被调用方的插桩，其中字节码将放置在方法调用中。调用方插桩指的是：将用于度量的字节码放置在要插桩的方法调用周围 （而不是放置在方法中）的过程。

调用方插桩允许您更好地控制插桩的放置位置，但会增加应用程序 classloader 时间，因为必须检查在范围中指定的每个类是否引用了在点中指定的类 / 方法。

调用方一端插桩常用于对 **rt.jar** 内的方法进行插桩调用。使用 bootclassloader 而非常规类加载程序加载到虚拟机中的类无法直接插桩。要对这些方法进行调用，您必须使用调用方的插桩。

在以下示例中，通过将字节码置于调用周围以解析任何类中的任意 (!.*) 方法，可插桩 **javax.xml.parsers.SAXParser** 和 **javax.xml.parsers.DocumentBuilder** 的解析方法。必须执行调用方插桩，这是因为 **javax.xml.parsers.SAXParser** 和 **javax.xml.parsers.DocumentBuilder** 类均包含在 rt.jar 中，并由 bootclassloader 加载到虚拟机中。

```
[XML-DOM-JDK14]
;---------- Interface --------------
Class = !javax\.xml\.parsers\.(SAXParser|DocumentBuilder)
method     = parse
signature  = !.*
scope = !.*
layer   = XML
```

以下示例插桩对 javax.naming.Context 的 “查找” 方法 （从 com.myCompany.myFoo 类调用）的调用，并将这些调用置于 FOO 层的 JNDI 子层中。

```
[JNDI-lookup-FOO]
;-------------- Server side JNDI hook --------------
class      = javax.naming.Context
method     = lookup
signature  = (Ljava/lang/String;)Ljava/lang/Object;
scope = !com\.myCompany\.myFoo\..*
deep_mode  = soft
layer   = FOO/JNDI
```

注意：

- 要验证是否通过点正确放置了字节码，请检查 < **探测器安装目录** >/log/< **探测器名称** >/**detailReport.txt** 文件中是否包含条目 “唯一标头名称”（即 [JNDI-lookup-FOO]）。
- 在性能问题的最终分类步骤中，对由应用程序可疑区域所调用的每个方法均使用类映射和各生成点可能不切实际。通常会使用一个或多个级别的调用方插桩来标识在具有可疑瓶颈的单个或多个方法中所花费的时间。此方法对于在 Diagnostics 中填入下一级的 “调用配置文件”非常有用。

以下示例插桩 **com.myCompany.myFoo** 类中由 “myMethod”方法执行的任何方法调用：

```
[MethodsCalledByFoo.myMethod]
class = !.*
method = !.*
scope = !com\.myCompany\.myFoo\.myMethod.*
layer = FOO/other
```

以下示例还捕获 **com.myCompany.myFoo** 类中由 “myMethod”方法调用的任何 “set”方法的参数：

```
[SetMethodsCalledByFoo.myMethod]
class = !.*
method = !set.*
scope = !com\.myCompany\.myFoo\.myMethod.*
detail = args:1
layer = FOO/other
```

URI 聚合插桩

应用程序通常使用相同的 URL 来访问不同的工作流程。如果应用程序使用 URI（例如，http://<myserver>/myApplication?page=home）参数来区分工作流程，则可以将 Diagnostics 配置为将不同的 URI 解析并视为不同的服务器请求。

URI 聚合通过 [HttpCorrelation] 点进行控制。应当为每种 URI 模式创建 args_by_class 的有效正则表达式条目。

以下示例允许在 Diagnostics 控制台中唯一地显示 ServerRequests：

```
http://<myserver>/myApplication?page=home
http://<myserver>/myApplication?page=openReport
```

```
[HttpCorrelation]
args_by_class=!.*&page
```

以下示例显示可将多个 URI 参数用于 URI 解析：

```
args_by_class=!.*&page&role
```

注意：请避免使用会话参数或高度唯一的 URI 值，因为这可能会影响开销和数据存储。

在 WebLogic 环境中使用 URI 聚合时，可在 **< 探测器安装目录 >/etc/inst.properties** 中设置 **use.weblogic.get.parameter=true**，以防止 URI 聚合消耗 ServletRequest 的 InputStream。

CORBA 跨 VM 插桩

利用公共对象请求代理体系结构 (CORBA) 标准，可以使以不同计算机语言编写、在不同系统中运行的组件协同工作。

< 探测器安装目录 >\etc\auto_detect.points 文件中提供了用于跨 VM 实例树关联 CORBA 的插桩。

请按照以下步骤为 CORBA 启用跨 VM 实例树：

1. 取消对 **auto_detect.points** 文件中 Corba 跨 VM 点的注释。
2. 在应用程序服务器启动时指定以下 JVM 参数：

   -Dorg.omg.PortableInterceptor.ORBInitializerClass.com.mercury.opal.javaprobe.handler.corba.CorbaORBInitializer

3. 将以下 jar 文件放入类路径：

   <Java 代理安装目录 >/lib/probeCorbaInterceptors.jar

使用 RMI 插桩

默认情况下，捕获点文件中的 RMI （跨 VM）点处于非活动状态。必须先激活此点，才能捕获应用程序中的跨 VM 处理。如果 Java 探测器中的此点在 RMI 调用两端均处于激活状态，则 Diagnostics 可将这两个虚拟机的调用树数据关联起来。

```
[RMI]
keyword = rmi
layer   = CrossVM
active  = false
```

群集环境中的 RMI 插桩

< 探测器安装目录 >/etc/inst.properties 文件中的 **weblogic.t3.rmi** 属性可控制 RMI 插桩捕获跨 VM RMI 性能度量的方式。默认情况下， **weblogic.t3.rmi** 设为 false，这样可使用常规 RMI 插桩来收集性能度量。在群集环境中，群集中的所有服务器都必须打开 RMI 插桩，以避免在 **weblogic.t3.rmi** 设为 false 时出现应用程序故障。

将 **weblogic.t3.rmi** 设为 true 后，会禁用常规 RMI 插桩，且只能使用 WebLogic 的 T3 协议捕获 RMI 跨 VM。这样， Java 探测器只能使用已探测启用了 RMI 插桩的群集中的某些服务器。

使用线程本地存储来存储 SOAP 负载

以下示例演示线程本地存储的用法。该示例特别说明了如何存储 （和清除）线程本地存储中的 SOAP 负载。默认情况下，只会为特定的应用程序服务器捕获 SOAP 负载。有关支持列表的详细信息，请参阅 “配置 SOAP 错误有效负载数据”（第 491 页）。

以下示例仅适用于某些应用程序服务器，这些服务器中的 Diagnostics 不会捕获现成的负载。

首先，需要明确可从何处访问负载。假设 payload 是方法 DispatchController.dispatch() 的第二个参数。

关键字 store-thread 会导致 Java 探测器将对应代码段 （在此示例中为 My_Inbound_Payload）中的特殊字段存储到线程本地存储中。这可以从不同的代码段进行引用，但前提是同一个线程中有这两个点。下一个示例 （“使用碎片本地存储来存储 Web 服务字段”（后续））演示查找负载的步骤。

```
[MyAppServer-SoapPayload-Capture]
class      = com.myCompany.DispatchController
method = dispatch
signature = !\(Ljava/lang/Object;Ljava/lang/Object;\).*
layer   = Web Services
detail    = before:code: ae7f0a58,store-thread


# Used by [MyAppServer-SoapPayload-Capture]
ae7f0a58 = ##My_Inbound_Payload=#arg2;"";
```

跨多个线程执行关联

异步服务器请求是在服务器请求起始和结束事件之间切换线程的服务器请求。最简单的情况是：一个线程接收请求后对其进行部分处理，然后将其交给另一个线程以完成处理，并将响应发送回请求方。

Diagnostics 可通过代码段提供以下两个操作，以允许 Java 代理跨多个线程执行关联：

- parkFragment(Object anchor)

执行此操作即表示当前线程将不再运行当前服务器请求。所有由 Java 代理记录的方法调用均会在此点处人为终止。这表明，即使其中某些方法会继续执行，其活动也将与当前服务器请求无关。此外，即使当前线程在调用 parkFragment 之后会调用某些已插桩的方法，也不会报告这些调用。服务器请求将不再视为在运行，但随后可能会由另一个线程继续执行，而应用程序会将指定对象 (anchor) 用作该服务器请求的唯一标识。

- resumeFragment(Object anchor)

  执行此操作表示当前线程继续执行之前停止的服务器请求。参数 (anchor) 用于标识服务器请求。所有活动的方法调用均会将其起始时间人为地重置为当前时间。这表明，即使这些方法在执行时可能已花费了一些时间，其执行过程也与恢复执行的服务器请求无关。如果当前线程已运行服务器请求，则将忽略（丢弃）该请求。

  使用这些操作时，应用程序专家必须识别出正确的 anchor 对象和线程切换点，这一点非常重要。

  请注意竞态条件：如果报告显示碎片“停放”太晚，则在执行相应的恢复操作之后，碎片将丢失（会在 probe.log 中记录一条警告消息）。可以通过以下两个有用技巧来避免竞态条件：首先，在当前线程确实要放弃服务器请求之前调用 parkFragment。其次，尝试附加应用程序内置同步，该应用程序内置同步经常用于将对象从一个线程交至另一个线程。

  可以使用挂起碎片 servlet 查看“停放”的碎片，将显示为“PARKED SERVER REQUEST”而不是当前运行的方法。

此功能通常需要您识别受监控应用程序中的线程切换点，并使用代码段提供对应插桩点。我们为 BEA AquaLogic 提供了现成支持。

下面显示了具有对应代码段的两个插桩点的示例。它们是 AquaLogic 支持的一部分。

一旦 AquaLogic 向另一个服务器发送子请求，便会执行下面显示的第一个点。如果子请求发送成功，则使用的方法 PipelineContextImpl.dispatch(...) 将返回 true。发送子请求的线程不会等待响应，但是会继续从管道接收下一个服务器请求。

因此，代码段会检查返回值，如果该值为 true，则会向探测器发送信号：当前服务器请求将挂起。服务器请求由 MessageContext 对象标识， AquaLogic 将为每个传入服务器请求创建该对象。

```
[BEA_ALSB_AsyncDispatch]
; instrumentation point for AquaLogic Service Bus asynchronous dispatch
class      = com.bea.wli.sb.pipeline.PipelineContextImpl
method     = dispatch
signature  = !\(Lcom/bea/wli/sb/context/MessageContext;.*
detail     = after:code:549b1b59
layer      = Service Bus/AquaLogic

# Used by [BEA_ALSB_AsyncDispatch]
# Asynchronously dispatches a subrequest for a service, the response will be
# processed on another thread
549b1b59 = (#return == true ?
@ThreadContextProxy@.parkFragment(#location,#arg1) : void);
```

从子请求接收了响应后， AquaLogic 可能会在另一个线程上执行 RouterCallback.onReceiveResponse(...)。对原始服务器请求的处理将继续进行，并且该处理过程将由代码段以信号形式发送至探测器。

在此情况下，代表服务器请求的 MessageContext 对象将不能作为已用方法的参数，且需要从 RouterCallback 对象中进行提取。

```
[BEA_ALSB_ProxyService_Callback_Response]
; instrumentation point for AquaLogic Service Bus callback function
class      = com.bea.wli.sb.pipeline.RouterCallback
method     = !(onError)|(onReceiveResponse)
signature  = !.*
layer      = Service Bus/AquaLogic
detail     = before:code:dba72078

# Used by [BEA_ALSB_ProxyService_Callback_Response]
# Resume processing of a server request when the reply for a subservice comes back
# (or when the server request was moved to the response pipeline internally)
dba72078 =
@ThreadContextProxy@.resumeFragment(#location,#callee._context.getMessageCon
text());"";
```

使用碎片本地存储来存储 Web 服务字段

以下示例演示点和代码段的几个功能：

- 如何使用碎片本地存储来存储特定于 Web 服务的字段（ws_name、ws_op 等）。这是另一种用于指定 “DIAG_ARG” 字符串的方法。
- 如何从上一个示例中存储的线程本地存储中检索 （和删除）所存储的负载。
- 如何从 SOAP 负载中提取用户 ID。
- 如何使用碎片本地存储来存储用户 ID。

  因为对 Web 服务的处理方式比较特殊，所以必须捕获多个字段。“代码段语法”（第 310 页）中对这些字段进行了介绍。

第一步是查找可用于访问必需字段 （Web 服务名称、操作、名称空间、端口名称）的插桩点。假设已插桩应用程序中有一个可访问所有这些字段的类，并假设此类名为 com.myCompany.MyWSContext。设置完所有以上字段后，我们需要访问此类的实例。其中可能会有很多选项。假设有这样一个选项：何时将 MyWSContext 作为方法 MyWSFactory.create() 的第一个参数进行传递。这便是我们要使用的方法。

以下是插桩点 （后面将对每行进行说明）：

```
class      = com.myCompany.MyWSFactory
method   = create
signature  = !\(Lcom/myCompany/MyWSContext;.*
layer      = Web Services
detail    = ws-operation, before:code: f334f0b4,store-fragment
```

以上所示点的前三行可使探测器插桩任何与 com.myCompany.MyWSFactory.create(MyWSContext, *) 匹配的项。

第四行指定此点的层。

第五行向探测器提供有关此点的其他信息 （详细信息）：

- 第一条详细信息 (ws-operation) 非常重要，因为它会使探测器将其视为入站 Web 服务。
- 第二条详细信息 (before:code: f334f0b4) 使探测器在符合此点的方法的开始处插入对应代码段。后面将显示实际的代码段。数字 f334f0b4 是通过访问

  http://< 探测器主机 >:< 探测器端口 >/inst/code-key 并在文本框中粘贴代码段生成的。
- 第三条详细信息 (store-fragment) 使探测器将在对应代码段中找到的所有特殊字段 (##) 存储为服务器请求的属性。

以下是对应的代码段 （后面将分别说明该代码段的每一行）。

```
f334f0b4 = #wsContext=#arg1;\
##WS_inbound_service_name=#wsContext.getServiceName();\
##WS_inbound_operation_name=#wsContext.getOperationName();\
##WS_inbound_target_namespace=#wsContext.getNamespaceURI();\
##WS_inbound_port_name=#wsContext.getEndpoint();\
#soap_payload =
@com.mercury.opal.capture.proxy.ThreadContextProxy@.getThreadContextValue("My
_Inbound_Payload");\
#consumer_id = (#soap_payload == null ? null :
@com.mercury.opal.capture.proxy.ProbeCodeSnippetHelper@.getConsumerIdFromDo
cument(##WS_inbound_service_name<java.lang.String>,#soap_payload<org.w3c.do
m.Document>));\
##WS_consumer_id = (#consumer_id == null ?
@ProbeCodeSnippetHelper@DO_NOT_STORE : #consumer_id);"";
```

第一行：f334f0b4 = #wsContext=#arg1;\

前面已提到，数字 f334f0b4 是通过转至 http://< 探测器主机 >:< 探测器端口 >/inst/code-key 并在文本框中粘贴代码段生成的。实际代码段从 f334f0b4 = 之后开始。第一个表达式为 #wsContext=#arg1。它仅将方法的第一个参数（此示例中为 MyWSContext 对象）分配到本地变量 (wsContext)。

第二行：##WS_inbound_service_name=#wsContext.getServiceName();\

此表达式使用碎片本地存储来存储服务名称。请注意，应使用正确的变量名称 (WS_inbound_service_name)。在 “代码段语法”（第 310 页）的 “特殊字段” 一节中记录了这些变量名称。

第三行：##WS_inbound_operation_name=#wsContext.getOperationName();/

此表达式使用碎片本地存储来存储 ws 操作。请注意，应使用正确的变量名称 (WS_inbound_operation_name)。在 “代码段语法”（第 310 页）的 “特殊字段” 一节中记录了这些变量名称。

第四行：##WS_inbound_target_namespace=#wsContext.getNamespaceURI();\

此表达式使用碎片本地存储来存储 ws 名称空间。请注意，应使用正确的变量名称 (WS_inbound_target_namespace)。在“代码段语法”（第 310 页）的“特殊字段”一节中记录了这些变量名称。

第五行：##WS_inbound_port_name=#wsContext.getEndpoint();\

此表达式使用碎片本地存储来存储 ws 端口名称。请注意，应使用正确的变量名称 (WS_inbound_port_name)。在“代码段语法”（第 310 页）的“特殊字段”一节中记录了这些变量名称。

以上的前五行可成功地捕获入站 Web 服务。代码段的其余部分负责从 SOAP 负载捕获用户 ID。此为可选部分，仅当所插桩的应用程序服务器是不支持捕获现成 SOAP 负载的应用程序服务器时才可用。有关详细信息，请参阅上一示例。在接下来的示例中，将使用在上一示例中捕获的 SOAP 负载。

第六行：#soap_payload =
@com.mercury.opal.capture.proxy.ThreadContextProxy@.getAndRemoveThreadContextValue("My_Inbound_Payload");\

此表达式从线程本地存储检索并删除所存储的负载 （有关存储方法，请参阅上一示例），并将负载存储在本地变量 (soap_payload) 中。

第七行：#consumer_id = (#soap_payload == null ? null :
@com.mercury.opal.capture.proxy.ProbeCodeSnippetHelper@.getConsumerIdFromDocument(##WS_inbound_service_name<java.lang.String>,#soap_payload<org.w3c.dom.Document>));\

此表达式设置 consumer_id 本地变量。如果负载为 Null，则将 consumer_id 设置为 Null。否则，将根据 consumer.properties 条目使用服务名称来执行用户 ID 匹配。有关用户 ID 匹配的详细信息，请参阅 “配置用户 ID”（第 480 页）。

第八行：##WS_consumer_id = (#consumer_id == null ?
@ProbeCodeSnippetHelper@DO_NOT_STORE : #consumer_id);"";

在最后一行中，该用户 ID 本地变量变成了此服务器请求的用户 ID。请注意，应使用正确的变量名称 (WS_consumer_id)。在 “代码段语法”（第 310 页）的 “特殊字段”一节中记录了这些变量名称。

对自定义插桩使用注释

使用 1.5 版或更高版本 JVM 的应用程序只需使用 Diagnostics Java 代理库目录中 **annotation.jar** 文件所含的自定义注释 (InstrumentationPoint)，便可 “强制”方法插桩。使用 InstrumentationPoint 注释编译类时，请在类路径中放入此文件的副本。注释定义如下 (InstrumentationPoint.java)：

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 * ------------------------------------------------------------------------
*/
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.METHOD)
public @interface InstrumentationPoint {
  String layer();
  String keyword() default “”;
  String layerType() default “method”;
  String detail() default “”;
  String code() default “”;
  Boolean active() default true;
}
```

此功能要求将 **inst.properties** 中的 **look.for.annotations** 属性设为 true（默认）。

开发

1. 将 Diagnostics Java 代理库目录中的 **annotation.jar** 文件路径添加到应用程序构建类路径中，或将 jar 复制到应用程序中。
2. 为需要监控的任何方法导入类：

   ```
   import com.mercury.diagnostics.common.api.InstrumentationPoint;
   ```

3. 标识要监控的方法，并添加注释：

```
@InstrumentationPoint(ARGS)
public void launchTest4()
```

在此实例中， ARGS 包含以下内容 （有关这些参数含义的详细信息，请参阅点文件文档）：

- layer="layer name"
- keyword= "keyword"
- layerType="type"
- detail="details"
- active="true/false"

示例

以下示例显示了使用 InstrumentationPoint 注释的代码。

```
/*
 * (c) Copyright 2008 Hewlett-Packard Development Company, L.P.
 ------------------------------------------------------------------------
 */

 import  com.mercury.diagnostics.common.api.InstrumentationPoint;

 Ö

 @InstrumentationPoint(layer=” my_app” ,detail=”
 diag,method-no-trim,method-cpu-time” )
 public void myMethod1(Object x, String y) {
    Ö
 }
```

在该示例中， myMethod1 将得到插桩，并在所有实例树中显示为一个节点。即使其延迟低于最低方法延迟阈值（默认为 51 毫秒），也不会清理此方法。系统将测量和报告此方法消耗的非独占 （包括子级） CPU。

为新的或自定义的技术配置跨 VM 关联

Diagnostics 可显示所有跨多个 Java 虚拟机 (JVM) 的调用配置文件。这些跨 VM 调用配置文件和拓扑在客户端和服务器出现性能问题时非常有用。对于间歇性问题，如果要知道是哪个应用程序导致产生问题，则单独查看客户端或服务器的调用配置文件可能不会有帮助，因为它们不关联。跨 VM 调用配置文件将在单个调用树内显示互相关联的客户端和服务器。

即开即用式 Java 代理可支持此功能用于多种不同的技术：例如， JMS、 HTTP/S（仅限 Web 服务）、 RMI、 SAP、 TIBCO 和 Corba。在最新版本的 Diagnostics 中添加了其他支持，用于帮助您为新的或自定义的技术配置跨 VM 关联。

跨 VM 关联技术以代码段的方式公开，从而允许您准备插桩点和代码段以为几乎所有的内部进程通信建立关联，其中包括自产型和旧有的通信方法。对这些通信技术的唯一要求是，其消息可承载将作为颜色的附加字符串。

颜色字符串由 Java 代理在客户端创建，并会由客户编写的代码段附加到传出消息。收到消息后，服务器端上用户编写的代码段便会从消息提取颜色，并将其传递到服务器端代理用于解析和处理。

但是，与跨 VM 通信方法相关的用户职责则主要局限于将颜色植入传出消息，并从收到的消息提取颜色。当然，用户职责中还包括识别客户端 （出站点）和服务端 （入站点）的代码位置 （插桩点）。有关详细示例，请参阅 “自定义技术的跨 VM 关联的自定义配置指南关联”（第 363 页）。有关所提供的用于帮助配置自定义跨 VM 关联的三种 API 的信息，请参阅 “用于简化自定义传输跨 VM 关联的 API”（第 361 页）。

客户端

对于出站调用 （客户端），请使用新的 **outbound:< 颜色类型 >** 详细信息。

可用的颜色类型有：

- default
- sap
- none
- snippet

对于除 **none** 之外的所有颜色类型，必须具有关联的颜色段，该段将提供包含技术类型、调用目标名称和标识的特殊参数。

此参数具有以下形式：

DIAG_ARG:type=< 类型 >&name=< 名称 >&target=< 目标 >

其中，< 类型 > 为用于远程调用的技术类型，< 名称 > 和 < 目标 > 为技术依赖值。该技术类型应与入站插桩点使用的相同 （请参阅 “服务器端”（第 361 页））。

对于除 **snippet** 之外的所有颜色类型，探测器将生成对应的颜色，并将该颜色报告至 Diagnostics 服务器用于今后的关联。但是，当前传出消息仍为未标识状态。

对于除 **none** 之外的颜色类型，其他插桩点 （在出站点之后、更可能是在出站方法执行过程中被点击）的代码段必须将生成的颜色附加到传出消息。

最新生成的颜色可通过调用 **ICorrelationColor RemoteCaptureProxy.getCurrentColor(#location)** 获取。

在为自己的跨 VM 通信开发支持时，可以选择 **snippet**，也就是说颜色将由来自代码段的直接调用明确创建。对于段颜色，则上述顺序会反向，也就是说将在出站点点击之前生成颜色 （且在大部分情况下，会直接附加到消息）。请注意，这将包含一种情况，即出站点**之前**的代码段会创建颜色，因为代码段将在调用代理之前执行。

要从代码段创建颜色，请执行以下操作：

1 对 **ICorrelationColor RemoteCaptureProxy.createColoring(#location, < 类型 >, <diag-arg>)** 进行调用

对于 type，请使用

RemoteCaptureProxy.ENCODED_COLORING 用于 **default**

RemoteCaptureProxy.SAP_R3_COLORING 用于 **sap**

如果不确定要使用的类型，则请使用默认值。

2 在步骤 1 中返回的对象上调用 **grabCorrelationString()**，并将返回的字符串插入外传消息 （使用技术依赖方法）。在此，您可发挥您的创造力。

提示：如果使用字符串消息，则请使用以下帮助程序 API 完成此步骤：

ProbeCodeSnippetHelper.createDiagEnvelope(String coloring, String originalMessage)

3 点击包含 **outbound:snippet** 详细信息的插桩点。此操作将自动使用最新创建的颜色，而非创建新颜色。执行出站点会提醒探测器颜色实际已被使用，并识别将被看做跨 VM 所有配置文件的连接点的方法。对于同步跨 VM 通信，建议使用同时用于发送消息和接收通知的方法的 “出站”详细信息，因此便可正确捕获出站调用的延迟。

服务器端

对于站内调用 （服务器端），请使用 **inbound:< 技术类型 >** 详细信息。在支持新的跨 VM 技术时，请使用自己的技术类型名称。请核查以避免与现有技术名称（服务器请求类型）产生冲突。服务器请求类型的示例包括：ADO、 CICS、 Corba、 HTTP、JDBC、JMS、MSMQ、RMI、Remoting (.NET)、SAP ABAP 类型和 Web 服务。此外，可能会发现名为 Pseudo 和 RootRename 的服务器请求类型。

before 代码段必须执行以下步骤：

1 使用与用于出站调用对应的技术依赖方法从传入消息提取关联字符串。

提示：如果之前使用了 ProbeCodeSnippetHelper.createDiagEnvelope()，则请使用 ProbeCodeSnippetHelper.extractColoringFromDiagEnvelope(String envelope) 获得关联字符串。

并使用 ProbeCodeSnippetHelper.extractOriginalMessageFromDiagEnvelope(String envelope) 获取原始消息。

2 在堆栈上保留以下两个字符串：捕获参数 （符合任何 **before** 代码段），以及提取的关联字符串。

用于简化自定义传输跨 VM 关联的 API

已添加三个帮助程序 API 用以简化自定义传输跨 VM 关联（请参阅以上章节中的提示，关于其使用方法的信息，请参阅 “代码段帮助程序”（第 314 页）。）此外，“自定义技术的跨 VM 关联的自定义配置指南关联”（第 363 页）也演示了一个示例。

➤ ProbeCodeSnippetHelper.createDiagEnvelope(String coloring, String originalMessage)

- ProbeCodeSnippetHelper.extractColoringFromDiagEnvelope(String envelope)
- ProbeCodeSnippetHelper.extractOriginalMessageFromDiagEnvelope(String envelope)

HTTP/S 支持

对于服务器端 HTTP/S 的支持是内置的，且默认状态下为启用状态。Java 代理会自动识别 HttpServlet 的标准 J2EE 执行，以及 Jetty 和 Apache Catalina 执行。如果使用了这些技术之一，则在服务器端不需要用户操作。

对于客户端， Java 代理将自动从 java.net.URL 类插桩 openConnection 方法，以将最新生成的颜色（如果存在）嵌入传出 HTTP 请求。 HTTP 请求标头之一将用于承载颜色。该标头将由服务器端代理识别。

但是，客户端上的 HTTP 支持属于以上规则的例外。您仍然需要提供出站点和对应 DIAG_ARG，但无需将颜色嵌入传出消息。

例如， Diagnostics Mediator 使用以下点：

```
[RemoteHttpComponent-Outbound-1]
class      = com.mercury.diagnostics.common.net.registrar.RemoteHttpComponent
method     = getConnection
signature  = (Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/
String;Ljava/lang/String;)Ljava/net/HttpURLConnection;
priority   = 1
detail     = method-no-trim,outbound:default,before:code:7b1125e2
layer      = Network.RemoteHttpComponent
```

getConnection 方法的首个参数为显示连接 URL 的字符串。引用的代码段将从其中提取主机名和端口，并用于目标识别。由 RemoteCaptureProxy 提供的特殊实用工具方法用于按照与 HTTP/S 支持内置部件一致的方式简化此转换。

```
7b1125e2 = #target=@RemoteCaptureProxy@.getTargetFromUri(#arg1); \
"DIAG_ARG:type=http&name="+#target+"&target="+#target;
```

自定义技术的跨 VM 关联的自定义配置指南关联

本指南将使用共享拦截队列的简化客户端服务器应用程序作为其自定义传输解决方案。客户端会通过将 “String”添加至队列发送 “String”消息。服务器则通过将其从队列移除接收 “String”消息。

尽管此示例运行单个 JVM （以保持其简单性），但它会使用两个线程以模拟应用程序在两个 JVM 内运行。（如果要关联单个 JVM 内运行的两个线程，有一种较为简便的方法帮助您实现。有关更多详细信息，请参阅 “跨多个线程执行关联”（第 349 页））。

实例代码如下所示：

```
public class SimulatedCrossVM {


 private static int INTERVAL = 5 * 1000; // 5 seconds
 private static BlockingQueue<String> queue = new LinkedBlockingQueue<String>();

 private static class ReceiverSide extends Thread {

  public ReceiverSide() {
   super("Receiver");
  }

  public void execute(String receivedString) throws InterruptedException {
   System.out.println("Executing message: " + receivedString);
   sleep(2 * INTERVAL);
  }
```

```
  private void receiveAndHandleMessage() throws InterruptedException {
    String message = null;
    message = queue.take();
    // Handle it
    execute(message);
  }

  public void run() {
    try {
      while (true) {
        receiveAndHandleMessage();
      }
    }
    catch (Throwable t) {
      // oops
      t.printStackTrace();
    }
  }
}

private static class SenderSide extends Thread {

  // For simulated TCP connection
  private String destHost;
  private int destPort;

  public SenderSide(String host, int port) {
    super(host + ":" + port);
    destHost = host;
    destPort = port;
  }

  public void sendMessage(String origMessage) throws InterruptedException {
    queue.put(origMessage);
  }

  private String generateMessage() {
    String message = "T" + System.currentTimeMillis();
    return message;
  }

  private void generateAndSendMessage() throws InterruptedException {
    sleep(2 * INTERVAL);
    // generate message
    String message = generateMessage();
```

```
    System.out.println("Sender's original message: " + message);
    // And send it (outbound call)
    sendMessage(message);
    sleep(INTERVAL);
   }

   public void run() {
    try {
     while (true) {
      generateAndSendMessage();
     }
    }
    catch (Throwable t) {
     // oops
     t.printStackTrace();
    }
   }
  }

  public static void main(String[] args) {
   SenderSide sender = new SenderSide("fake-host", 12345);
   ReceiverSide receiver = new ReceiverSide();

   sender.start();
   receiver.start();
  }
}
```

执行此代码将得到以下输出：

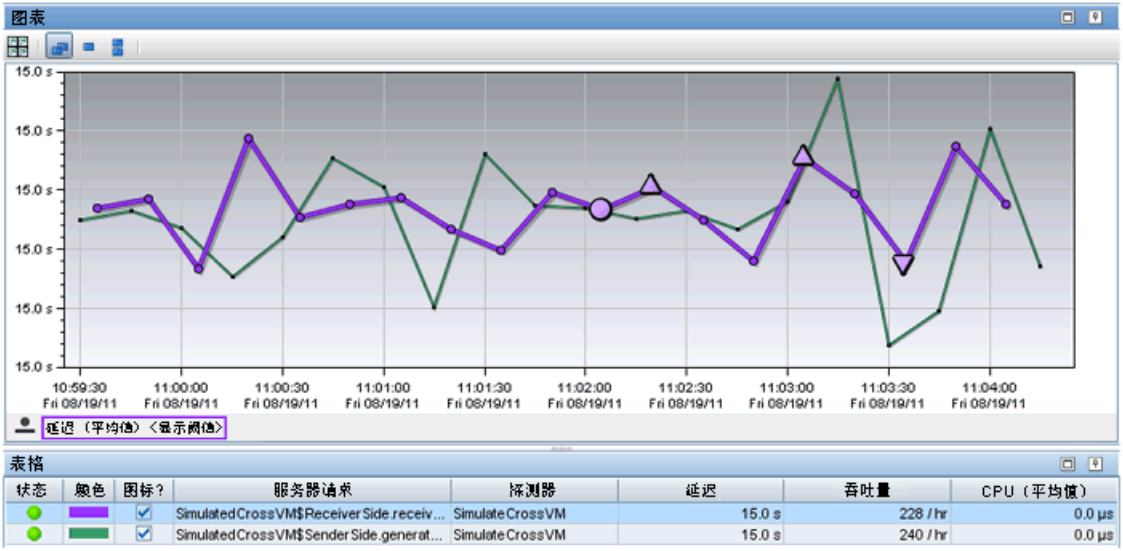
```
Sender's original message: T1313785958127

Executing message: T1313785958127
```

步骤 1：插桩方法

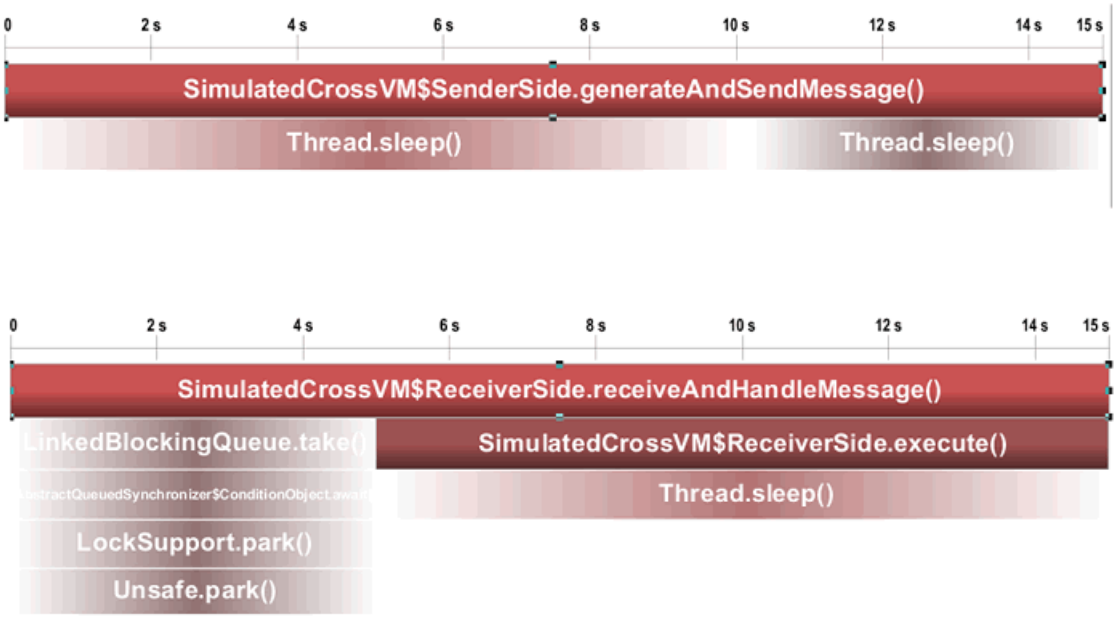
通过插桩方法，便可使 Diagnostics 了解重要的方法有哪些。由于这些方法是自定义的，因此可直接使用的插桩点将不进行任何操作。通过添加以下插桩点编辑 etc/autodetect.points 文件。有关定义插桩点的指南，请参阅“维护 Java Profiler UI 的插桩”（第 372 页）。

```
[SimCrossVM-Sender]
class      = SimulatedCrossVM$SenderSide
method     = generateAndSendMessage
signature  = !.*
layer      = Sending


[SimCrossVM-Outbound]
class      = SimulatedCrossVM$SenderSide
method     = sendMessage
signature  = !.*
layer      = Sending

[SimCrossVM-Receiver]
class      = SimulatedCrossVM$ReceiverSide
method     = receiveAndHandleMessage
signature  = !.*
layer      = Receiving


[SimCrossVM-Inbound]
class      = SimulatedCrossVM$ReceiverSide
method   = execute
signature  = !.*
layer      = Receiving
```

结果：运行此插桩测试程序，您将看到以下服务器请求：



以下是为发送方和接收方显示的调用配置文件。



步骤 2：向发送方逻辑添加 “颜色”

在此步骤中，要向由客户端发送的消息添加“颜色”。当插桩服务器收到此有色消息后，HP Diagnostics 会为它们建立关联。如果对代码段语法不太熟悉，则此部分会相对较难理解，可在 “定义具有代码段的点”（第 308 页）内参阅相关说明。

首先，将方法标记为使用代码段的出站点 (outbound:snippet)，然后在调用方法 (before:code:5ea4753f) 之前识别要执行的代码段。由于要使用首个参数，提供更加明确的签名 (!\(Ljava/lang/String;.*) 是个不错的选择。

```
[SimCrossVM-Outbound]
class      = SimulatedCrossVM$SenderSide
method     = sendMessage
signature  = !\(Ljava/lang/String;.*
layer      = Sending
detail     = outbound:snippet,before:code:5ea4753f
```

对应的代码段显示如下。行 1 将创建包含主机名和服务器目标端口的字符串 (#target)。行 2 将定义遵循特殊语法（DIAG_ARG:type=< 类型 >&name=< 名称 >&target=< 目标 >）的新字符串 (#diagArg)。“类型”即技术类型，可以是您所选择的任何名称；该类型将用于下一步骤。“名称”和 “目标”为技术依赖值，将在 UI 内显示；这些值也可以是您所选择的任何值。行 3 将定义第三个字符串 (#color)，用于识别任何其他方法调用的此特定调用。行 4 将用有色字符串更新方法的首个参数，从而导致 sendMessage 发送修改过的字符串。最后，行 5 会将颜色置于堆栈供 HP Diagnostics 使用。

```
1  5ea4753f = #target=#callee.destHost+":"+#callee.destPort; \
2  #diagArg =
   "DIAG_ARG:type=CB-TCP&name=Senders.sendMessage&target="+#target; \
```

3 #color = (null == #arg1 ? null :
@RemoteCaptureProxy@.createAndGrabColor(#location,
@RemoteCaptureProxy@ENCODED_COLORING, #diagArg.toString())); \

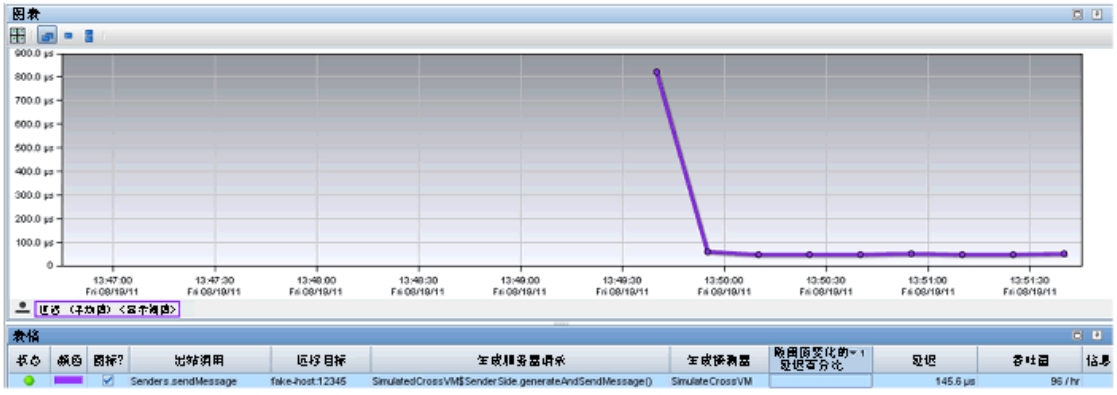
4 #arg1 = @ProbeCodeSnippetHelper@.createDiagEnvelope(#color, #arg1); \

5 #diagArg;

运行此示例会按如下所示更新输出。请注意，接收端收到的字符串消息与发送的并不相同。这是由代码段的第 4 行造成的。在许多情况下，接收端可能无法正常处理该行。当客户端和服务器未使用相同插桩，尤其是两者并非均进行了插桩时，这种现象会偶然发生，注意接收方的行为是个很好的办法。

Sender's original message: T1313786970403
Executing message: HP_DIAG1_!Dhf/
ABAABKrh3Qf0cy7yaLsAAAAAAA9mYWtlLWhvc3Q6MTIzNDUAYTEzMTM3ODY5NjAzODgmU2ltdWxhdGVDcm9zc1ZNJlNpbXVsYXRlZENyb3NzVk0kU2VuZGVyU2lkZS52b2lkIGdlbmVyYXRlQW5kU2VuZE1lc3NhZ2UoKSZcMCZcMCZcMCY=:T1313786970403

此时，您将在 UI 内发现的唯一改变是某些 “出站调用” 。请注意，位于 “出站调用”和 “远程调用”列内的值即您在代码段 “名称”和 “目标”内提供的值。



步骤 3：从接收端删除颜色

最后一步为从接收端删除颜色，因此接收方便可从发送方获取原始的“未着色的”消息。首先，用步骤 2 内定义的代码段内使用的技术类型将点标识为入站点，然后在调用此方法之前，分配要运行的代码段。同样，由于将在代码段内使用参数，还要指定一个更为明确的签名。

```
[SimCrossVM-Inbound]
class      = SimulatedCrossVM$ReceiverSide
method   = execute
signature  = !\(Ljava/lang/String;.*
detail     = before:code:d2c83d3c,inbound:CB-TCP
layer      = Receiving
```

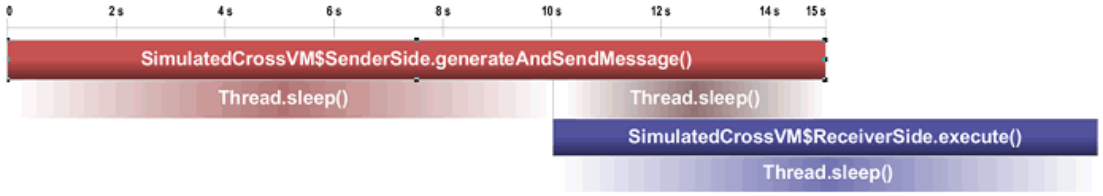
对应的代码段显示如下。第 1 行将从传入消息提取颜色。第 2 行会更新方法的首个参数，并将其存储在由客户端发送的原始消息内。第 3 行会将颜色置于堆栈（以及空白字符串），以供 HP Diagnostics 使用。

```
1 d2c83d3c =
  #coloring=@ProbeCodeSnippetHelper@.extractColoringFromDiagEnvelope(
  #arg1); \
2 #arg1=@ProbeCodeSnippetHelper@.extractOriginalMessageFromDiagEnvelope(
  #arg1); \
3 "";#coloring;
```

程序的输出现已存储在原始消息内：

```
Sender's original message: T1313789287234
Executing message: T1313789287234
```

“服务请求”视图此时显示有跨 VM 调用配置文件可用于发送方的“generateAndSendMessage”。打开此调用配置文件，观察客户端和服务器调用配置文件现已关联在一起。在此实例应用程序中，这些配置文件并无太大功用，但在真实应用程序中则可发现是否在客户端或服务器内、或者两者同时发生了性能问题。



此调用配置文件看似不常见，但却典型用于异步应用程序。客户端不会等待响应，但会继续执行其他处理（5 秒的错误休眠）。在此休眠过程中，服务器会处理请求并在几秒后完成。可在树形结构内查看方法的时间段，如下所示。还请注意包含数字 2 的菱形，它显示的是 JVM 深度。如果您的服务器已执行了另一个出站调用，则该深度可能为 3 或更深！在这些情况下，请使用跨 VM 关联，因为非常实用。可以想象尝试横跨数多 JVM 寻找性能问题的源头会有多麻烦！

维护 Java Profiler UI 的插桩

您可以使用 Java Diagnostics Profiler 中的 “配置” 选项卡来维护插桩点以及编辑探测器配置，而无需手动编辑 Java 代理捕获点文件或属性文件。不管配置文件是否已启动，都可以从 Java Diagnostics Profiler 访问 “配置” 选项卡。

您可以在 Java Diagnostics Profiler 的 “设备” 部分中，查看和更新探测器所监控的应用程序的插桩限。使用编辑对话框，可查看和编辑捕获点文件中定义的插桩点， Diagnostics 使用这些插桩点来插桩应用程序。

检查当前插桩

要检查因当前捕获点文件中的点而插桩的层、类和方法，请在 “配置” 选项卡的 “设备” 部分中单击 “查看 ...”。此时 Profiler 将显示 “Instrumented Layers” 页面：


Diagnostics

Instrumented layers (no particular sorting)

Layer	Hits	Active Points	Actions
(Other)	97539	1 / 1	[Disable] [Clear # Hits]
(keyword) http	78356	13 / 13	[Disable] [Clear # Hits]
(keyword) lwmd	1004363	2861 / 2861	[Disable] [Clear # Hits]
(keyword) remote-http	0	12 / 12	[Disable] [Clear # Hits]
(keyword) soap fault	0	1 / 1	[Disable] [Clear # Hits]
Business Tier/EJB/Entity Bean	436449	596 / 596	[Disable] [Clear # Hits]
Business Tier/EJB/Session Bean	48922	110 / 110	[Disable] [Clear # Hits]
Database/JDBC/Connection	93203	57 / 57	[Disable] [Clear # Hits]
Database/JDBC/Execute	45968	64 / 64	[Disable] [Clear # Hits]
Directory Service/JNDI	479	5 / 5	[Disable] [Clear # Hits]
HttpStatus	0	20 / 20	[Disable] [Clear # Hits]
Legacy/JCA/Connection	23075	1 / 1	[Disable] [Clear # Hits]
Legacy/JCA/ECIConnectionFactory	22918	2 / 2	[Disable] [Clear # Hits]
Legacy/JCA/ManagedConnectionFactory	20	2 / 2	[Disable] [Clear # Hits]
Messaging/JMS/Listener	0	1 / 1	[Disable] [Clear # Hits]
SOAPHandler	0	1 / 1	[Disable] [Clear # Hits]
Web Services	0	1 / 1	[Disable] [Clear # Hits]
Web Tier/Servlet	24073	23 / 23	[Disable] [Clear # Hits]
Web Tier/Struts	0	2 / 2	[Disable] [Clear # Hits]

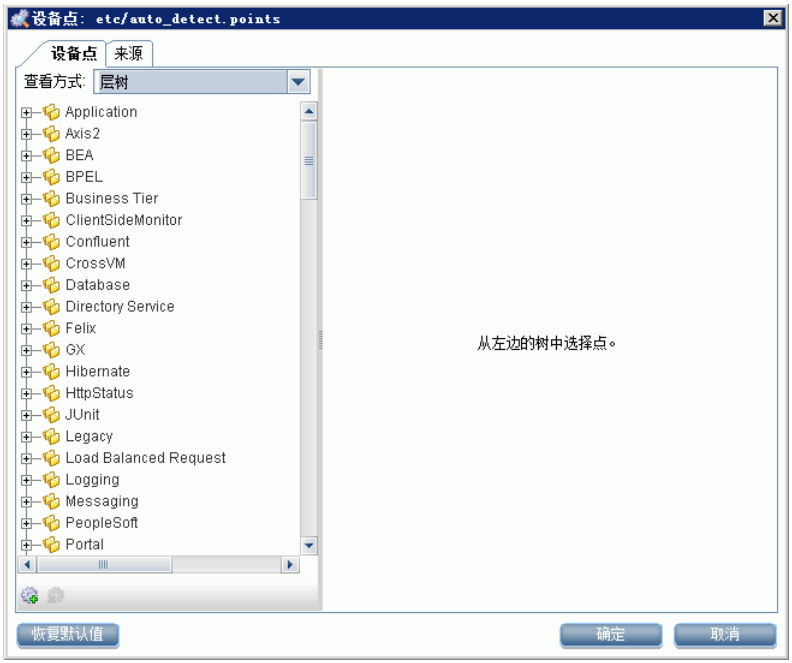
HP Diagnostics J2EE Probe "P81_WAS5_Plants_ovrntt152_W2k", version 9.00.70.1002

“Instrumented Layers”页面列出了已插桩的层、层中的插桩点的触发次数，以及层中当前处于活动状态的点数。提供了以下列：

列	描述
Layer	列出已插桩的层。此列中的层名称将链接到一个页面，该页面提供了有关探测器所监控的层中的处理操作的详细信息。**注意：**仅会列出在已实际插桩的点中定义的的层。
Hits	包含对所列出层中的点监控的类和方法的调用次数。可以使用“操作”列中的“清除点击次数”链接来重置该计数。
Active Points	包含当前处于活动状态的点数以及为特定层定义的总点数。
Actions	包含可供您用于处理所列层的信息的链接。可用操作有： ➤ **Disable：**禁用所选层中的所有点，使它们不再捕获数据。插桩仍在原位，并且可再次启用。该启用或禁用点操作仅在应用程序下次重新启动之前有效。要更改点的默认启用状态，请参阅“对捕获点文件中的点进行编码”（第 300 页）。 ➤ **Clear # Hits**：重置在“Hits”列中为所选层显示的点击计数。

维护插桩点

要维护用于提供插桩指令（指示探测器在应用程序中的监控对象）的点，请导航至 Java Diagnostics Profiler 中的“配置”选项卡，并单击“共享设备”或“实例设备”的“编辑 ...”。此时将打开“设备点”对话框：

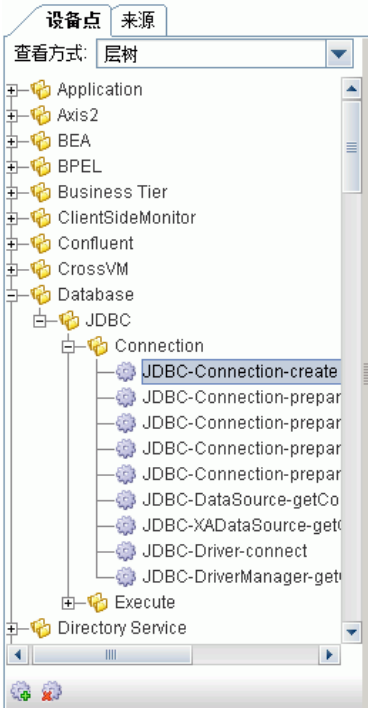


可以使用两种方法编辑插桩：直观地，可使用“设备点”选项卡中的点列表或点树进行编辑；或者通过“来源”选项卡中的捕获点文件的源进行编辑。

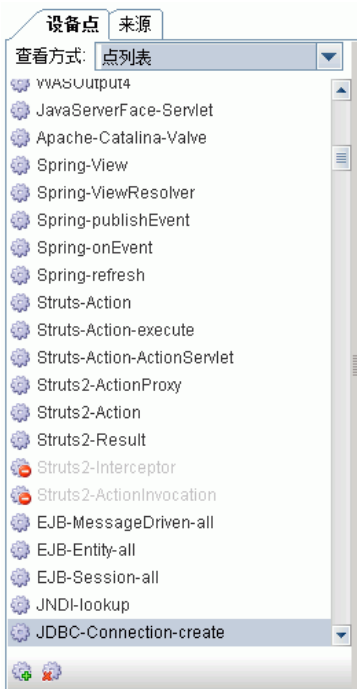
选择和查看现有点

可使用“设备点”对话框中的导航栏在捕获点文件中查找要维护的点。可以从“查看方式”下拉列表中选择点的列出格式。

从下拉列表中选择 “层树”后，将根据分配到点的层和子层以树结构形式列出捕获点文件中的点：



从下拉列表中选择 “点列表”后，将按字母顺序升序排列捕获点文件中的点：

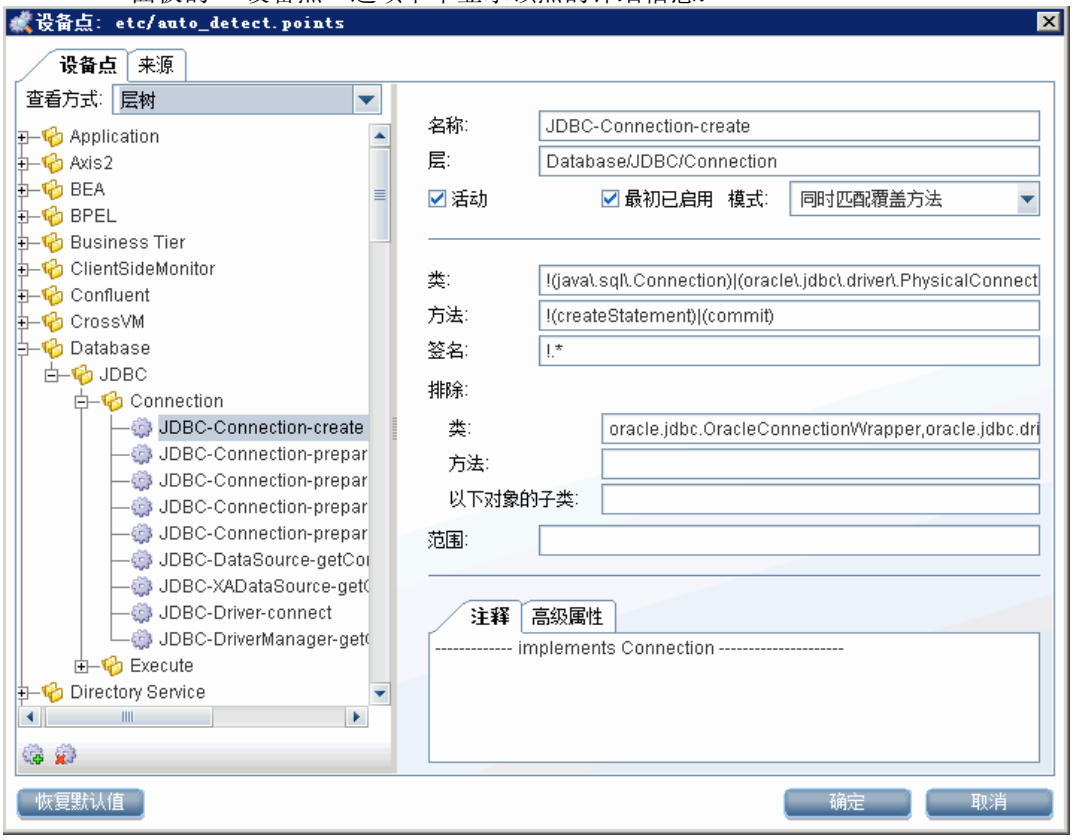


找到要查看或维护的点后，请在导航栏中选择该点。然后，就可以在 “查看 / 编辑”面板中查看所选点的详细信息，并可在该面板中对点进行维护。

更新现有点

从导航栏中选择层或子层后，“查看 / 编辑”面板将仅包含一条提示信息，提醒您选择点。

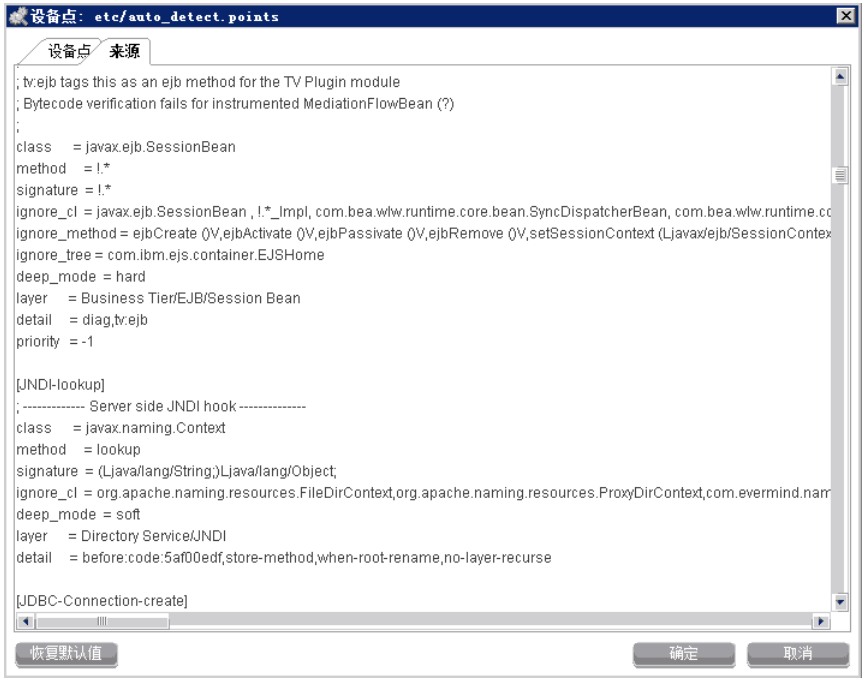
要更新现有点，请从导航栏中选择一个点，这样， Profiler 便会在“查看 / 编辑”面板的 “设备点”选项卡中显示该点的详细信息：



在定义捕获点文件中的点时常用的参数将显示为独立的数据字段，以便轻松进行任何必要的更新。将在页面底部的 “高级属性”选项卡中显示更多高级参数。对点的注释将显示在 “注释”选项卡中。更改之后，请单击 “确定”。最后请记得单击 “应用更改”，以应用通过 “配置”选项卡进行的所有更改。

“对捕获点文件中的点进行编码”（第 300 页）中对可用于在捕获点文件中定义点的参数进行了说明。

以下是 “来源”选项卡的一个示例：



删除现有点或层

您可以删除导航栏中所列的点或层。

要删除点或层，请执行以下操作：

1 从 “设备点”选项卡的导航栏中选择点或层。

2 单击 “删除点”。 Profiler 将从导航栏的列表中删除选定实体。

在从 Profiler 的 “配置”选项卡中应用所有插桩点更新之前，选定实体实际上并未从捕获点文件中删除。

3 单击 “确定”，关闭 “设备点”对话框。

4 单击 “应用更改”，以应用通过 “配置”选项卡进行的所有更改。

添加新点

您可以向插桩中添加点。

要添加点，请执行以下操作：

1 单击 “新建点”。 Profiler 将显示 “选择新点类型”对话框：

2 从下拉列表中选择相应的点类型，然后单击 “确定”。

Profiler 将显示已初始化 “查看 / 编辑”部分的 “设备点”选项卡，以创建选定点类型的新点。

3 在选项卡的相应位置输入新点的参数和注释。

输入 “层”信息后，导航栏中对应于新点的条目将更新，将在正确的现有层中显示该点；或者如果指定的层尚不存在，则将在全新的层中显示该点。

在从 Profiler 的 “配置”选项卡中应用所有插桩点更新之前，新点实际上并未添加到捕获点文件中。

4 单击 “确定”，关闭 “设备点”对话框。

5 单击 “应用更改”，以应用通过 “配置”选项卡进行的所有更改。

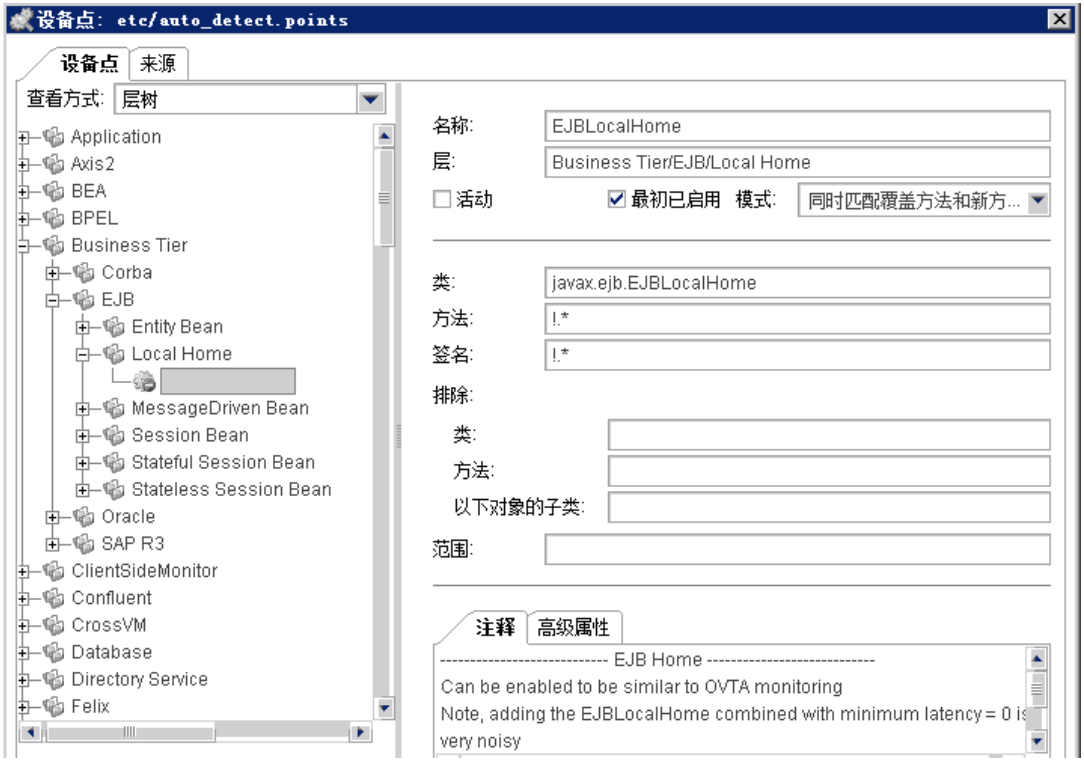
激活类似 OVTA 的点

点包含在 Servlet 筛选器和 EJB 本地主方法的 Java 探测器插桩中。这些插桩点可提供类似于 OVTA （OpenView 事务分析器） Java Monitor 的附加功能。

ServletFilter 点还要求激活 HttpCorrelation2 点，以便正确地监控服务器筛选器。这是因为有时候，Servlet 筛选器是 Diagnostics 在首次处理 HTTP 服务器请求时所使用的筛选器。

默认情况下，EJBLocalHome、ServletFilter 和相关 HttpCorrelation2 插桩点均处于非活动状态。非活动点将通过在插桩点旁的图标上标记一个红色符号来表示，如下图所示。要使用这些点，请通过 UI 或通过直接编辑文件的方式在 **auto_detect.points** 文件中设置 active=true。

按照 “选择和查看现有点”（第 376 页）所述在 Profiler UI 中找到这些点，然后导航至 **Business Tier>EJB>LocalHome>EJBLocalHome** 点或 **Web Tier>Servlet>ServletFilter** 点以及 **HttpCorrelation2** 点。



要将这些点设置为活动状态，请执行以下操作：

1. 从“设备点”导航栏中选择点，以便 Profiler 显示该点的详细信息。选中“活动”复选框。
2. 单击 “确定”，关闭 “设备点”对话框。
3. 单击 “应用更改”，以应用通过 “配置”选项卡进行的所有更改。重新启动应用程序服务器 （将重新启动探测器），使新激活的点生效。

恢复默认点

使用 Profiler 或 HP Diagnostics 诊断完问题后，可以恢复默认插桩，以免造成更多插桩开销。

要恢复插桩的默认设置，请执行以下操作：

1. 单击 “恢复默认值”。

   在从 Profiler 的 “配置”选项卡中应用所有插桩点更新之前，插桩点实际上并未添加到捕获点文件中。

2. 单击 “确定”，关闭 “设备点”对话框。

3. 单击 “应用更改”，以应用通过 “配置”选项卡进行的所有更改。

为典型 Java 类和方法定义的默认层

根据捕获点文件中提供的指令，HP Diagnostics 会将类和方法的性能度量分组为 “层”和 “子层”。默认层已定义，这样就可以同时报告使用相似系统资源的应用程序在处理过程中的性能度量。这些层支持您更轻松地隔离并确定可能会影响性能问题的系统区域。

下表列出为典型 Java 类和方法定义的默认层和子层。

捕获点文件中还定义了一些特定于平台的层。这些层中的大多数是下表中所定义的顶级父层的子层。您可以在 Diagnostics UI 的 “加载视图”中查看这些层的性能数据。

Java EE 层

层	子层	父层
Web Tier	JSP Servlet Struts Session Spring Struts2	
Business Tier	EJB Corba	
Web Services		
EJB	Entity Bean Session Bean Local Home Stateless Session Bean Stateful Session Bean MessageDriven Bean	Business Tier
Directory Service	JNDI	
Database	JDBC	
JDBC	Execute Connection	Database
Messaging	JMS Spring	
JMS	Producer Listener Consumer	Messaging
Spring	Producer Consumer	Messaging
Hibernate		

Portal 层

Diagnostics 会将与 Portal 处理相关的类和方法调用的性能度量分成多个“Portal 组件”层。每个“Portal 组件”层均细分成 Portal 生命周期方法的多个层。有关 Portal 层的详细信息，请参阅 《HP Diagnostics User's Guide》。

10

.NET 应用程序的自定义插桩

本节描述如何控制 HP Diagnostics 应用到应用程序的类和方法的插桩，以支持 .NET 代理收集性能度量。

本章包括：

关于插桩和捕获点文件

插桩是指，当 CLR 加载应用程序的类文件时，探测器在这些类文件中插入的字节码。通过插桩，探测器能够测量执行时间、计算调用数、捕获异常以及关联方法调用和线程。可在捕获点文件中指定每个探测器的插桩点。

您可以通过捕获点文件来控制插桩的范围，以便 Diagnostics 为您提供了解应用程序性能所需的所有信息，而避免被代价高昂或含糊不清的无关信息淹没。捕获点文件中包含的插桩定义称为 “点”，它告诉探测器要插桩哪些方法、如何进行插桩以及应当插桩哪些内容。

点可以包括 “通配”指令的正则表达式，以便将它们应用于多个方法、类或名称空间规范。有关如何使用正则表达式的详细信息，请参阅 “使用正则表达式”（第 882 页）。

您可以对捕获点文件中的点进行自定义，使其包括不属于默认点的应用程序区域的方法、类和名称空间。

除了 Web 和 WCF 方法的插桩， .NET 的 Microsoft 规范并不包括业务逻辑应当实现的统一接口或建议接口。这意味着 .NET 探测器始终需要捕获点文件中的自定义点，才能在 .NET 应用程序中收集关于业务逻辑类和方法性能的有意义度量。

捕获点文件中的点将被划分成多个层。利用层，可以将性能度量组织成多个有意义的信息层，这些信息层可作为分类流程的一部分并能控制插桩的收集行为。

捕获点文件中的点将被划分成多个默认层。您既可以自定义默认层，也可以新建层 （请参阅 “典型 .NET 应用程序的默认层”（第 422 页））。

查找 .NET 捕获点文件

安装 .NET 代理时，会安装预定义的默认捕获点文件。

ASP.NET 应用程序的默认捕获点文件在 **< 探测器安装目录 >\etc** 中，包括 **Asp.Net.points**、 **Ado.points**、 **WCF.points** 以及下表中显示的其他点文件。

此外，.NET 代理安装程序会为它所检测的每个 IIS 部署的 ASP.NET 应用程序域自动创建一个单独的捕获点文件。您必须修改自动检测和创建的点文件，才能为应用程序域启用自定义插桩点。这些捕获点文件在 **< 探测器安装目录 >\etc\< 应用程序域 >.points** 文件中。这些点文件和默认点文件由 .NET 代理读取。

在安装中，仅会*启用* **Asp.Net.points**、 **Ado.points** 和 **WCF.points** 默认点文件。以下默认 .NET 点文件安装在 **< 探测器安装目录 >/etc** 目录中，但*未启用*：

默认点文件 （初始处于禁用状态）	插桩目标
Asp.Net.IExecutionStep.points	IIS5、 IIS6 和 IIS7。此文件将使 IIS 点不再可用。
IIS.points	IIS5 和 IIS6
Lwmd.points	Lightweight Memory Diagnostics
Msmq.points	Microsoft Message Queuing （MSMQ 插桩）
Remoting.points	.NET Remoting
WebServices.points	ASP.NET Web 服务

通过在 **probe_config.xml** 文件 appdomain 范围中的 **< 点 >** 元素中添加对点文件的引用，可以启用这些点文件。有关 **probe_config.xml** 文件中每个元素的详细信息，请参阅第 13 章，“了解 .NET 代理配置文件”。

有关特定于 TransactionVision 的 .NET 探测器插桩的信息，请参阅《HP TransactionVision Deployment Guide》。

对捕获点文件中的点进行编码

以下参数可用于定义点文件中的点：

```
[Point-Name] =<unique name for the point>
;-------------------------------------------------------------------------
class        = <class/package name/s to capture>
method     = <method name/s to capture>
signature  = <signature/s of method/s>
ignoreClass          = <classes to ignore>
ignoreMethod= <method prototypes to ignore>
ignoreTree= <class hierarchy to ignore>
deep_mode= <soft or hard mode>
scope       = <comma separated list of methods>
ignoreScope= <comma separated list of methods>
detail                  = <list of specifiers>
keyword    = <keyword>
layer         = <layer name>
layerType  = <layer type>
```

注意：请不要修改任何默认点文件，因为在安装升级时修改会丢失。您需要将特定于您应用程序的插桩点存储在自定义的捕获点文件中。

所有可以被指定为正则表达式列表的参数的最长有效限制均为 260 字符，如果超过此限制，将会导致值被截断。以下各节对这些参数进行了描述。

强制点参数

所有点都必须包含以下参数，但 LWMD、HttpCorrelation、WSCorrelation 和 WCF 的点除外：

参数	描述
Point-Name	点的唯一名称。
class	指定要插桩的类或接口的名称。该名称应当包括完整的名称空间名称，名称空间和类级别之间使用句点隔开。可以使用任何有效的正则表达式。
method	指定要插桩的方法的名称。为保证成功，方法名称必须与 class 参数指定的类或接口中定义的方法匹配。可以使用任何有效的正则表达式。
layer	指定在其下分组此点中数据的层、子层或层级。层是插桩收集控制的一部分。 通过使用斜杠 (/) 分隔层名称，可将点中的层指定为嵌套层或子层。斜杠后面指定的层是斜杠前指定层的子层。一个子层还可以有自己的子层，方法是在子层名称后面再编写另一个斜杠和层名。

以下是一个包含强制参数的自定义点示例：

```
[MyCustomEntry_1]
; comments here….
class = myNameSpace.myClass.MyFoo
method = myMethod
layer = myCustomStuff
```

注意：正则表达式可用于点中的大多数参数，但是必须以感叹号开头。有关如何使用正则表达式的详细信息，请参阅 “使用正则表达式”（第 882 页）。

可选的点条目

点定义可包含以下一个或多个参数：

参数	描述
keyword	表示特殊插桩。关键字参数可用于启用特定功能；例如，WCF 关键字可打开 WCF 功能。关键字参数还可以将点定义与特殊功能关联；有关此功能的示例是 RemotingServer 关键字和 Remoting.points 文件。 ➤ **HttpCorrelation：**通过 HTTP 打开客户端 / 服务器方法调用关联。 ➤ **WsCorrelation：**在客户端打开 Web 服务关联逻辑，并跨 .NET 和 Java 技术，打开原始 HTTP 客户端请求调用关联。 ➤ **WCF：**打开 WCF 功能。 ➤ **REST：**打开 WCF REST 服务插桩。 ➤ **lwmd：**打开 lwmd 插桩。 ➤ **Remoting：**打开 .NET Remoting 框架插桩。 ➤ **RemotingServer：**将 .NET Remoting 服务器中的点与这些点的特殊 .NET Remoting 逻辑关联。请参阅 “如何为 .NET Remoting 配置插桩”（第 411 页）。
ignoreClass	指定要忽略的以逗号分隔的类列表。将不会插桩与通过 **ignoreClass** 指定的类相匹配的任何类。

参数	描述
ignoreMethod	指定要忽略的以逗号分隔的方法列表。将不会插桩与通过 **ignoreMethod** 指定的方法相匹配的任何方法。
ignoreTree	对于从指定类继承的类，忽略在该类上实现的任何插桩方法。因此，整个类层次结构树的插桩方法都会被忽略。
deep_mode	指定处理子类的方式。此参数接受三个值： ➤ **none** - 此值相当于不指定 **deep_mode** 参数，它对子类的处理方式没有任何影响。 ➤ **soft** - 该值要求对于每个类或与类、方法和签名条目匹配的接口，也应对实施相同匹配方法和签名的任何子类或子接口进行插桩。 ➤ **hard** - 该值要求对于每个类或与类、方法和签名条目匹配的接口，对应对任何深度的子类或子接口的所有方法进行插桩。硬模式通常用于接口的点。**注意：**“hard”模式可导致大量插桩，并会使探测器开销过大。
scope	约束执行插桩时所在的上下文。如果指定了该参数，则插入的字节码将是调用方一端。任何有效的正则表达式均可用于此参数的值。范围值表示为以逗号分隔的方法名列表。
ignoreScope	从 scope 参数指定的方法范围中排除某些方法。任何有效的正则表达式都可用于此参数的值。 ignoreScope 值表示为以逗号分隔的方法名列表。

参数	描述
detail	提供更具体的捕获说明。 返回的以下字符串显示在“调用配置文件”视图详细信息窗格的方法“参数”字段中。该参数是以下各项的以逗号分隔的列表： **args:n** – 捕获匹配方法的所有支持参数类型。值为“n”表示捕获所有参数。您还可以为 n 输入一个从 1-256 的值。 **args:0** – 调用当前类实例或被调用方对象上的 ToString()。它对静态方法无效。 ***args:1** – 如果方法是顶级请求，则在参数上标记 (*)，表示该参数是服务器请求的关键参数。 详细信息参数同样使用下列值： **tv:user_event** – 为匹配的方法生成 TransactionVision 事件。作为 TransactionVision 事件的一部分，将方法的参数作为请求有效负荷收集，将返回值作为响应有效负荷收集。显示的值为由参数或返回值对象返回的 ToString() 值。请注意，可能并非收集所有的参数和返回值。 通过从任何 .NET 应用程序内的几乎任何一种给定方法启用 TransactionVision 事件生成，为事务跟踪提供广泛支持。指定希望 TransactionVision 事件生成的方法。强烈建议一次仅为事件生成指定一种方法，以避免在 TransactionVision 内过多的事件和性能降级。避免使用通配符规范 （但为方便起见，通配符规范仍受支持）。

参数	描述
layerType	为某些插桩的方法指定特殊处理，并接受三个值： ➤ **trended_method** – 标识要在“已趋势化的方法”视图中显示的方法。 ➤ **sql** – 标识用于捕获 SQL 视图中 SQL 的方法。它们由 HP Diagnostics 设置，不得修改。
签名	指定签名 （返回任意参数类型）；例如， System.String（System.int32、 System.String）。可以使用任何有效的正则表达式。

插桩示例

以下示例说明如何通过创建和修改在捕获点文件中的点，来自定义应用程序的插桩。

本节包括：

- “如何为非 ASP.NET 或 Windows 应用程序配置和设置点”（第 407 页）
- “如何为 .NET Remoting 配置插桩”（第 411 页）

自定义层和子层

- 以下点会为 myCompany.myFoo 类中的 myMethod 方法在名为 “FOO”的层中创建一个名为 “BAR”的自定义子层：

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
layer = FOO/BAR
```

通配符方法

- 以下点捕获 MyCompany.MyFoo 类中的所有方法：

```
[myCompany.myFoo_AllMethods]
class = myCompany.myFoo
method = !.*
layer = FOO/BAR
```

忽略指定的方法

- 以下点可捕获 MyCompany.MyFoo 类中除 setHomeInterface 和 getHomeInterface 方法外的所有方法：

```
[myCompany.myFoo_AllMethodsExcept]
class = myCompany.myFoo
method = !.*
ignoreMethod = setHomeInterface,getHomeInterface
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany 名称空间中的所有方法，但包含在 MyCompany.logging 类中的方法除外：

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
ignoreClass = MyCompany.logging
layer = FOO/BAR
```

已趋势化的方法视图的捕获方法

➤ 以下点可捕获所需数据以填充 myMethod 方法的已趋势化的方法视图：

```
[myCompany.myFoo_customLayer]
class = myCompany.myFoo
method = myMethod
layer = FOO/BAR
layertype = trended_method
```

仅捕获类中的特定方法

➤ 以下点可捕获 MyCompany.MyFoo 类的所有非静态构造函数方法：

```
[myCompany.myFoo_Constructor]
class = myCompany.myFoo
method = .ctor
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类的所有静态构造函数方法：

```
[myCompany.myFoo_Singleton]
class = myCompany.myFoo
method = .cctor
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类中的 setFoo 方法：

```
[myCompany.myFoo_setFoo]
class = myCompany.myFoo
method = setFoo
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany.MyFoo 类中名称包括 “set”的所有方法：

```
[myCompany.myFoo_AllSets]
class = myCompany.myFoo
method = !.*set.*
layer = FOO/BAR
```

➤ 以下点可捕获 MyCompany 名称空间中的所有方法：

```
[myCompany_All_Methods]
class = !myCompany\..*
method = !.*
layer = FOO/BAR
```

捕获可返回字符串的特定方法

➤ 以下点可捕获 MyCompany.MyFoo 类中返回 System.String 的 getFoo 方法：

```
[myCompany.myFoo_GetFoo_String]
class = myCompany.myFoo
method = getFoo
signature = !System.String\(.*
layer = FOO/BAR
```

调用方插桩

默认情况下，Diagnostics 中的所有插桩都是被调用方的插桩，其中字节码放在方法调用中。调用方的插桩指的是：将用于度量的字节码放在要使用的方法调用周围 （而不是放置在方法中）的过程。

调用方插桩允许您更好地控制插桩的放置位置，但会增加应用程序的初始化时间，因为必须检查在范围中指定的每个类是否引用了在点中指定的类 / 方法。

scope 和 ignoreScope 参数用于指定应对什么样的调用方执行插桩。以下两个示例为调用方的插桩。

- 以下点可捕获 MyCompany 名称空间中从 MyCompany.logging 类调用的所有方法。

```
[myCompany_All_Methods_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
scope = !MyCompany.logging.*
layer = FOO/BAR
```

- ignoreScope 参数用于从 scope 参数指定的范围中排除某些特定类和方法。以下点可捕获 MyCompany 名称空间中从 MyCompany.logging 类调用的所有方法，但从 myMethod 方法调用的方法除外。

```
[myCompany_All_Methods_except_from_MyCompany_Logging]
class = !myCompany\..*
method = !.*
scope = !MyCompany.logging.*
ignoreScope = MyCompany.logging.myMethod
layer = FOO/BAR
```

参数捕获

要捕获的参数是在点文件部分的 detail 关键字中指定的。

以下示例调用 n-th 参数的 ToString() 方法。返回的字符串显示在 “调用配置文件”视图的方法参数字段中。 detail=args:1,...args:4, *args:3

有几个特殊值要注意：

- args:n – 捕获匹配方法的所有支持参数类型。值为“n”表示捕获所有参数。您还可以为 n 输入一个从 1-256 的值。
- args:0 – 调用当前类实例或被调用方对象上的 ToString() 方法。
- 将 * 添加到 args 元素 (*args:1)，将其标记为关键参数。

要查看每个方法调用的参数，请不要指定关键参数。这可让您获取有关已捕获的实例树的更详细信息，并有助于回答此类问题：为何此实例是 MAX 树，或者出现异常时应传递哪些值？

要按参数对方法的服务器请求分组，请指定一个关键参数。关键参数可以聚合具有独特值的服务器请求。具有大量独特值的参数不是最佳的关键参数备选项，因为这将导致每个独特值都有唯一的服务器请求。

注意：即使尚未指定参数捕获，当调用树中的方法抛出异常时，也会捕获参数。这些参数显示在 “调用配置文件”视图 “异常详细信息”页的 “堆栈跟踪”部分中。有关更多详细信息，请参阅 “调用配置文件视图”联机帮助。

以下参数捕获示例与后面显示的代码相关：

```
[ILTest]
class = !ILTest_NameSpace.ILTest_Class
method = methodWithParams
detail = args:0, *args:3, args:5, args:7
layer = myFunctionLayer
```

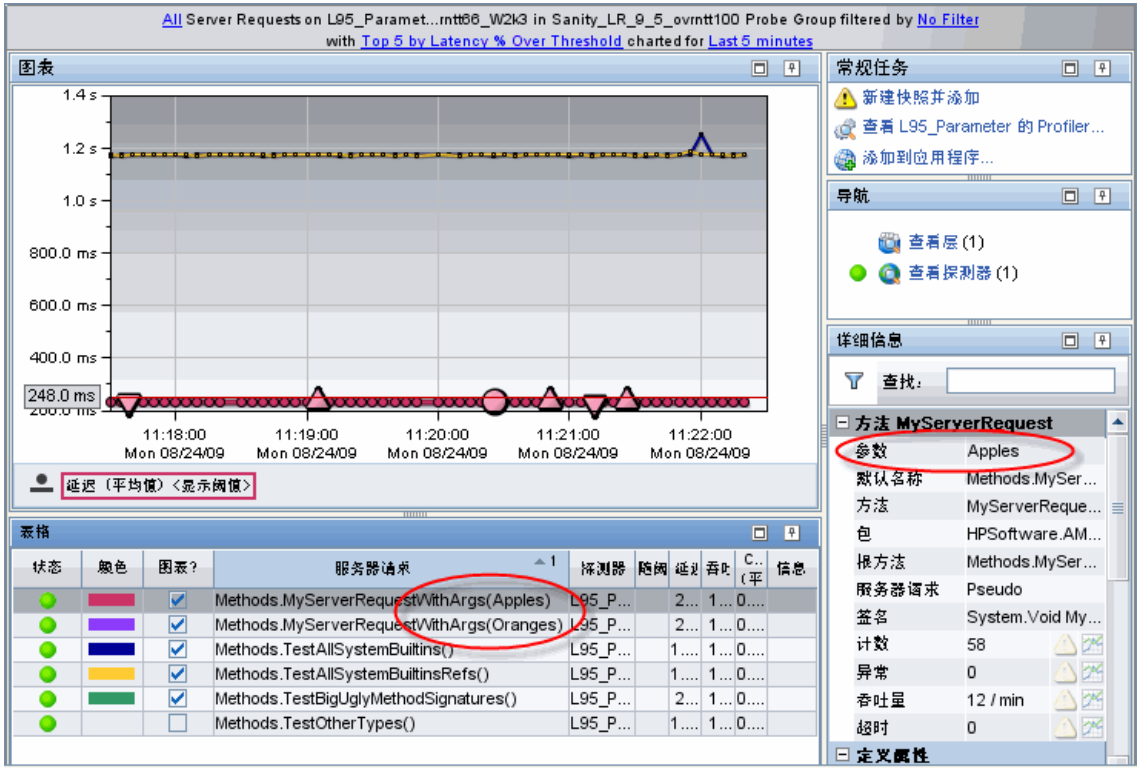
下面是相关代码示例：

```
class ILTest_Class
{
public bool methodWithParams
(string param1, int param2, string QnameParam3, long param4, object param5, int
param6, double param7)
{

... some implementation

}
}
In this example the defined detail will capture ILTest_Class.ToString(args:0)
param1, QnameParam3, param5 and
param7.
```

如果顶级方法是 methodWithParams，则 QnameParam3 的值将是服务器请求标识的一部分。

当要捕获的参数被标记为关键参数 （带有星号 *）并且方法是顶级方法时，参数值会成为服务器请求标识的一部分。

例如，如果“交付类型”是处理不同货物的方法参数，并且将“交付类型”参数指定为关键参数，您将看到由方法处理的每种不同货物（“Apple”和“Orange”）的聚合视图。



指定关键字参数时，“调用配置文件”视图在 “详细信息”窗格的 “参数”字段中显示关键参数。您还可以在“详细信息”窗格的“方法参数”看到显示的参数。

如果要捕获的参数未标记为关键参数 （不带星号 *），这些参数则仅显示在 “调用配置文件”视图的 “方法参数”下。

为监控配置 WCF REST 服务

对于 .NET 探测器，基于 **WCF.points** 文件内默认启用的 **keyword=REST** 值，WCF REST 服务默认受到监控。这些 REST 服务将作为 Web 服务受到监控，且其性能数据将在 Diagnostics UI SOA 服务视图内显示。

随后还可按照以下章节所述对 REST 服务进行配置。

REST 服务配置

在 WCF REST 样式服务中，有时候会将操作编码为 url 参数。例如：

HTTP 方法：PUT Url: http://localhost:81/RestNOSvc/AccountsRESTService/{ID}?op={OPERATION} op can be "deposit" or "withdrawal"

为能够辨别这些服务类别内的操作，可以将 REST 服务方法的操作参数指定为**关键参数**，以允许其作为单独的操作显示。有关参数捕获的常规描述，请参阅 “参数捕获”（第 400 页）。

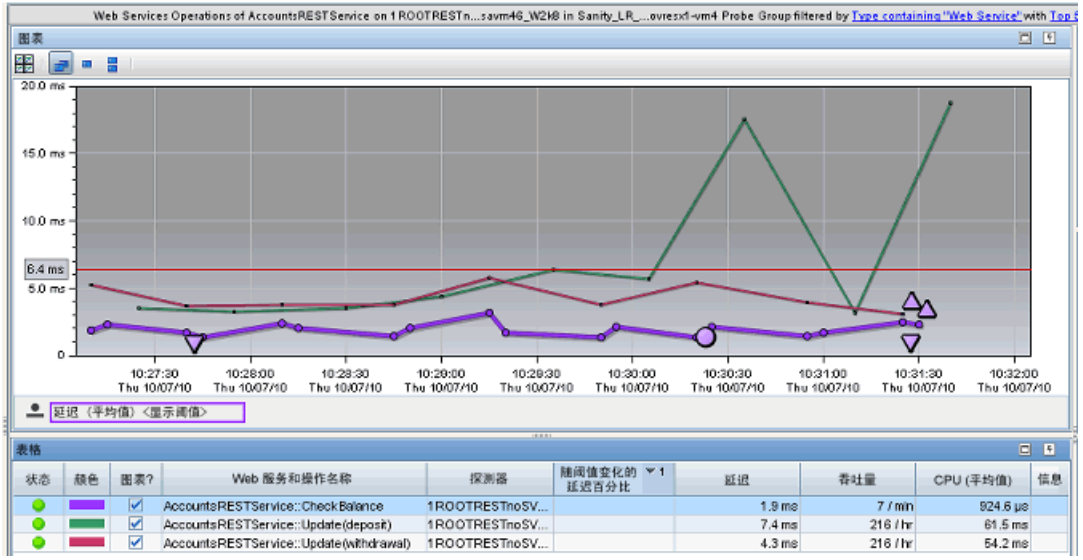
例如，对于参数

```
[WebInvoke(UriTemplate = "{id}?op={operation}", Method = "PUT")]
public TransactionResult Update(string id, string operation, long Amount)
```

操作为关键参数，且可在点文件内指定为：

```
[WebSite2-RestNOSvc]
class = !HP.Test.WcfRestService.*
method = Update
detail = *args:2
layer = WebSite2-RestNOSvc
```

下方 “SOA 服务操作” 视图示例显示了通过表格内的单独操作实现的此配置的结果。



REST 客户端配置

REST 服务客户端与 HTTP 客户端调用相同，无法区分。因此，对于监控属于 REST 服务客户端的 .NET 应用程序，应该在 **probe_config.xml** 内设置配置选项 **<httpclient showurl=”false”/>**，以避免大量的出站调用及可能的符号表激增。调用数量取决于客户端访问的唯一 url 数，通常包含以 url 编码的 ID。

例如：

/RestNOSvc/AccountsRESTService/
8FFD2F34-E334-4E1E-A940-50FCCCACE1D1

其中 Guid 表示不同的帐户 ID。

Deep_mode 示例

以下接口定义用于 soft deep_mode 和 hard deep_mode 示例：

```
public interface Interface1 {

    public void callerMethod();

}
```

以下类用于软和硬 deep_mode 示例：

```
public class Class1 implements Interface1 {
    public void callerMethod(){
        calleeMethod();
        calleeMethod2();
    }

    public void calleeMethod(){
        Console.WriteLine("hello world");
      //more code lines here…
    }

    public void calleeMethod2(){
        Console.WriteLine("hello world 2");
    }
}
```

➤ 以下点可捕获 Class1 类中的 callerMethod：

```
[Training-1]
class      = Interface1
method     = !.*
deep_mode  = soft
layer      = Training
```

➤ 以下点可捕获类 1 中的所有方法，即 callerMethod、 calleeMethod1 和 calleeMethod2：

```
[Training-1]
class      = Interface1
method     = !.*
deep_mode  = hard
layer      = Training
```

如何为非 ASP.NET 或 Windows 应用程序配置和设置点

本节说明如何配置 **probe_config.xml** 文件和自定义点文件，这两种文件用于为非 ASP.NET 或 Windows 应用程序启用插桩。Windows 服务、控制台应用程序、Windows 窗体应用程序和 WPF 应用程序的插桩都可被视为 Windows 应用程序，被引用时也是如此。

Windows 应用程序设计

当考虑如何配置要监控的 Windows 应用程序时，关键是要将 .NET 探测器设计为可监控长时间运行的进程。因此，如果 Windows 应用程序被设计为运行几秒后即退出，则您可能无法看到有关该运行的任何数据。因为当 Windows 应用程序快速退出时，在它能够与 Diagnostics 服务器或 Diagnostics .NET Profiler 建立并保持通信之前， appdomain 和探测器也会关闭。

下面这个简单的 Windows 应用程序说明了在配置 Windows 应用程序的插桩时，要考虑的诸多关键概念：

```
namespace Hello_dotNet_nameSpace
{
      class someclass
      {
            static void Main(string[] args)
            {
// do something

                  // read form commandline then exit
                  clReader myClReader = new clReader();
                  String cl;
                  cl = myClReader.readCl();
            }
      }
      // Command Line Reader
      public class clReader
      {
            public String clread;

            public String readCl()
            {
                  System.Console.WriteLine("Continue?");
                  clread = Console.ReadLine();
                  return clread;
            }
      }
}
```

Hello_dotNet.exe Windows 应用程序中的 Main() 将调用方法，等待用户在命令行上输入命令，然后退出。该应用程序退出后，探测器将开始处于活动状态。

创建 Hello_dotNet.points 文件

在 **<探测器安装目录>\bin** 文件夹中有一个 **Reflector.exe** 命令行实用程序，您可以对 Hello_dotNet.exe Windows 应用程序运行该实用程序，以获取建议的点文件。有关 Reflector 实用程序的详细信息，请参阅“搜寻应用程序中的类和方法”（第 590 页）。

当 Reflector.exe 和 Hello_dotNet.exe 应用程序位于同一文件夹时，可使用以下命令：

```
Reflector.exe Hello_dotNet.exe
```

输出会发送到 stdout。您将在其他信息中看到以下建议的 Hello_dotNet.points:

```
---------------------------------------------------------------------
Sample .points by Namespace
---------------------------------------------------------------------
[Hello_dotNet_nameSpace]
class = !Hello_dotNet_nameSpace.*
layer = Hello_dotNet_nameSpace
```

上述建议点可以按原样使用，除非 Windows 应用程序有类似于 Main() 的方法；也就是说，方法如果经过插桩，它在应用程序退出之前不会返回 exit。在这种情况下，方法将跨越应用程序的生存期，因此在应用程序退出前不会报告任何数据。由于探测器会在应用程序退出时卸载，因此您可能无法从插桩点获取任何数据。

要解决此问题，请构造一个点文件，以不使用 Main() 方法或类似的任何方法。以下 Hello_dotNet.points 文件显示如何执行此操作。假设在 someclass 中实现了 Main()。

Hello_dotNet.points:

```
[Hello_dotNet_nameSpace]
class = !Hello_dotNet_nameSpace.*
ignoreClass = Hello_dotNet_nameSpace.someclass
layer = Hello_dotNet_nameSpace

[ignore]
class = Hello_dotNet_nameSpace.someclass
ignoreMethod = Main
layer = Hello_dotNet_nameSpace
```

此类点文件的重要方面以粗体显示。如果在忽略 Main() 方法时，Hello_dotNet_nameSpace.someclass 中有其他方法，则 [ignore] 部分会使用其他方法。

为插桩配置 Windows 应用程序

要配置 .NET 探测器以使用 Hello_dotNet.exe Windows 应用程序，请将以下 XML 添加到 **probe_config.xml** 文件中。您可以将它添加到文件底部的 **</probeconfig>** 条目上方。

```
<process name="Hello_dotNet">
      <points file="Hello_dotNet.points" />
      <instrumentation>
        <logging level="" />
      </instrumentation>
      <logging level="" />
</process>
```

注意：在更改 **probe_config.xml** 文件之前，必须将 **Hello_dotNet.points** 文件放入 **< 探测器安装目录 >\etc** 文件夹。

唯一需要的子元素是点文件，插桩、记录和模式则是可选的。以下插桩设置在诊断哪些方法已使用、哪些方法未使用时十分有用：

```
    <instrumentation>
       <logging level="points ilasm" />
    </instrumentation>
```

如何为 .NET Remoting 配置插桩

可以配置 .NET 探测器，以添加支持 .NET Remoting 客户端和服务器应用程序插桩的自定义插桩。支持的配置为：

- HTTP 和 TCP 绑定
- 二进制和 SOAP 格式化

配置

默认情况下，并不会启用 .NET 探测器来插桩 Remoting 应用程序。因此，您必须为客户端和服务器应用程序添加自定义插桩点。

有两个插桩关键字与 Remoting 相关：

Remoting：Remoting 关键字能为 Remoting 框架中的各种点启用插桩。

RemotingServer：RemotingServer 关键字标识实现 Remoting 方法的类，并将该类上的方法插桩与其他类似方法的非预期插桩隔离开。

客户端示例

下面这个简单的 Windows 应用程序示例说明了在配置 Remoting 客户端应用程序的插桩时，要考虑的诸多关键概念：

```
namespace HPSoftware.AM.Tests.Remoting.SimpleRemoting
{
   class SimpleConsoleClient
   {
      [STAThread]
      static void Main(string[] args)
      {
         const string msg1 = "How are you?";

         String filename =
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile;
         RemotingConfiguration.Configure(filename, false);

         MyRemotableObject remoteObject = new MyRemotableObject();

         doit(remoteObject, myMsg);

         Console.WriteLine();
         Console.WriteLine("(Press any key to exit)");
         Console.ReadKey();
      }

      public static void doit(MyRemotableObject obj, String message)
      {
            Console.WriteLine(obj.GetEnlightenment(message));
      }
}
```

如“如何为非 ASP.NET 或 Windows 应用程序配置和设置点”（第 407 页）中所述，可以使用 Reflector 实用程序来帮助确定如何配置 Remoting 客户端点文件。

要配置探测器以使用 SimpleConsoleClient Remoting Windows 应用程序，请将以下 XML 添加到 **probe_config.xml** 文件：

```
<process name="SimpleConsoleClient">
        <points file="Remoting.points" />
        <points file="SimpleConsoleClient.points" />
        <instrumentation><logging level="" /></instrumentation>
        <logging level="" />
 </process>
```

必须添加 **<points file="Remoting.points" />** 条目。

如果您所在的目录中包含 SimpleConsoleClient.exe，并且 Reflector.exe 在 PATH 中，则可以在命令行上执行 Reflector，以查看 SimpleConsoleClient.exe 的实现分解及建议的点文件设置：

```
Reflector SimpleConsoleClient.exe
```

此命令的输出将包含以下内容：

```
---------------------------------------------------------------------
Sample .points by Namespace
---------------------------------------------------------------------
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
---------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
---------------------------------------------------------------------
HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient (8
Methods)
   Equals System.Boolean(System.Object)
   Finalize              System.Void()
   GetHashCode           System.Int32()
   GetType               System.Type()
   doit                  (method signature information unavailable))
   Main                  System.Void(System.String[])
   MemberwiseClone  System.Object()
   ToString              System.String()
```

建议的 SimpleConsoleClient.points 为：

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

但是，这些设置不会创建可产生任何数据的插桩。因为如 “如何为非 ASP.NET 或 Windows 应用程序配置和设置点”（第 407 页）中所讨论的，您必须忽略类似于 Main() 的方法。如果考虑到需要忽略 Main() 的情况，则您会有以下可能的点文件设置：

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
ignoreMethod = Main
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

尽管这些设置可能有用并且能产生数据，您也应使它们更精确。这主要是因为探测器性能会受到影响：使用的方法越多，所使用的应用程序上的探测器性能受到的影响越大。例如，如果可以从设置中删除通配符 “!.*”，则设置的范围将更明确。

请注意， Reflector 的输出中实际上只有一个已实现的类：

HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient

可以按如下方法从类设置删除通配符：

```
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient
```

另请注意，Reflector 输出不包含方法设置，无方法设置的默认含义是使用所有方法。由于以下大多数方法都是从 System.Object 继承的，所以将它们显示在此处，但您实际上不太可能使用这些方法：Equals、Finalize、GetHashCode、GetType、MemberwiseClone、ToString。不过，您可能会使用 doit 方法，因为它将包装 Remoting 客户端调用。以下设置是 SimpleConsoleClient.points 文件的建议设置：

```
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleClient
method = doit
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting
```

服务器示例

下面这个 Windows 应用程序示例说明了在配置 Remoting 客户端应用程序的插桩时，要考虑的诸多关键概念：

显示远程对象（它是 Remoting 客户端和服务器之间的共享对象）和 SimpleConsoleServer.exe Remoting 服务器应用程序的 C# 代码段。

下面是远程对象的 C# 代码段：

```
HPSoftware.AM.Tests.Remoting.SimpleRemoting
{

    public class MyRemotableObject : MarshalByRefObject
    {
      const string response = "I'm just fine!";

        public MyRemotableObject()
        {
        }

        public String GetEnlightenment(string message)
        {
            return response;
        }
    }
}
```

以下是 SimpleConsoleServer.exe 的 C# 代码片段：

```
namespace HPSoftware.AM.Tests.Remoting.SimpleRemoting
{
   class SimpleConsoleServer
   {
      [STAThread]
      static void Main(string[] args)
      {
         String filename =
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile;
         RemotingConfiguration.Configure(filename, false);

         Console.WriteLine("Server is running...    press any key to exit");
         Console.ReadKey();
      }
   }
}
```

要配置探测器以使用 SimpleConsoleServer Remoting WIndows 应用程序，请将以下 XML 添加到 **probe_config.xml** 文件：

```
<process name="SimpleConsoleServer">
        <points file="SimpleConsoleServer.points" />
        <instrumentation><logging level="" /></instrumentation>
        <logging level="" />
</process>
```

不需要添加 **<points file="Remoting.points" />** 条目。

Remoting 服务器的点文件可以包含一个或多个部分。第一个部分与远程对象相关，是必需部分。第二个部分与 Remoting 服务器插桩相关，可以添加。此外，还可以添加其他可选部分，以使用可以由 Remoting 方法或 Remoting 服务器调用的其他方法。我们将首先构造远程对象部分。

远程对象在某些程序集中，我们假设它在 RemotableObjects.dll 中。

对 RemotableObjects.dll 运行 Reflector 时，可以看到输出中包括以下内容：

```
-------------------------------------------------------------------------
Sample .points by Namespace
-------------------------------------------------------------------------
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting

-------------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
-------------------------------------------------------------------------
HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject (17
Methods)
   __RaceSetServerIdentitySystem.Runtime.Remoting.ServerIden…)
   __ResetServerIdentity              System.Void()
   CanCastToXmlType                   System.Boolean(System.String,System…)
   CreateObjRef                       System.Runtime.Remoting.ObjRef(Syste…)
   Equals                             System.Boolean(System.Object)
   Finalize                           System.Void()
   GetComIUnknown                     System.IntPtr(System.Boolean)
   GetEnlightenment                   System.String(System.String)
   GetHashCode                        System.Int32()
   GetLifetimeService                 System.Object()
   GetType                            System.Type()
   InitializeLifetimeService          System.Object()
   InvokeMember                       System.Object(System.String,System…)
   IsInstanceOfType                   System.Boolean(System.Type)
   MemberwiseClone                    System.MarshalByRefObject(System…)
   MemberwiseClone                    System.Object()
   ToString                           System.String()
```

与 Remoting 客户端示例一样，不能仅使用建议的点设置，您必须确定已标识实现远程对象的类。确定这一点的方法是：观察是否需要远程对象来继承 System.MarshalByRefObject 以及远程对象上是否有以下方法：CreateObjRef、GetLifetimeService、 InitializeLifetimeService、 MemberwiseClone。在上面的 Reflector 输出中，可以看到 HPSoftware.AM.Tests.Remoting.SimpleRemoting。 MyRemotableObject 类是实现远程对象的类的明显候选项。

远程对象部分必须包括 **keyword = RemotingServer** 条目。此条目表示探测器的 Instrumenter 应当为此部分中的点设置执行特殊处理。此特殊处理负责完成两个任务，它将插桩从 System.MarshalByRefObject 继承的类上的所有方法。因此，无需指定要插桩哪些 Remoting 方法，所有 Remoting 方法都将会插桩。同时，这也解释了此部分中不需要方法条目的原因。其次，此关键字将在 System.MarshalByRefObject 继承类上实现的方法插桩与指定的类隔离开。这一点很重要，因为有很多系统类和用户类也是从 System.MarshalByRefObject 继承的，您并不想在无意中使用它们。

根据上述说明，下面列出了建议的远程对象部分：

```
[RemotableObject]
keyword = RemotingServer
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject
layer = RemotableObject
```

现在你就可以构造可选的 Remoting 服务器部分。如果要监控不通过 Remoting 方法就可调用的服务器逻辑，则只需要创建此部分。

对 SimpleConsoleServer.exe 运行 Reflector 时，可以看到输出中包括以下内容：

```
------------------------------------------------------------------------
Sample .points by Namespace
------------------------------------------------------------------------
[HPSoftware.AM.Tests.Remoting.SimpleRemoting]
class = !HPSoftware.AM.Tests.Remoting.SimpleRemoting.*
layer = HPSoftware/AM/Tests/Remoting/SimpleRemoting


------------------------------------------------------------------------
(1 classes)  Namespace: HPSoftware.AM.Tests.Remoting.SimpleRemoting
------------------------------------------------------------------------
HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleServer (7
Methods)
   Equals                System.Boolean(System.Object)
   Finalize              System.Void()
   GetHashCode           System.Int32()
   GetType               System.Type()
   Main                  System.Void(System.String[])
   MemberwiseClone       System.Object()
   ToString              System.String()
```

如“如何为非 ASP.NET 或 Windows 应用程序配置和设置点”（第 407 页）中所述，您不能仅使用建议的点设置，而必须忽略 Main() 方法。

根据上述说明，以下设置是对 SimpleConsoleServer.points 文件的建议设置：

```
[RemotableObject]
keyword = RemotingServer
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.MyRemotableObject
layer = RemotableObject

[RemotingServer]
class = HPSoftware.AM.Tests.Remoting.SimpleRemoting.SimpleConsoleServer
ignoreMethod = Main
layer = RemotingServer
```

最后，还可以添加其他可选部分，以使用可以由 Remoting 方法或 Remoting 服务器调用的其他方法。

了解自定义插桩的开销

创建自定义插桩时，注意不要过度使用应用程序，因为这样会导致被探测的应用程序产生过多的延迟； 自定义插桩对方法延迟或 CPU 开销并不会产生这样的影响，因为每个方法的插桩开销几乎是固定的，这是由于字节码数量几乎总是相同。根据执行方法时所耗用的时间长度不同，CPU 和延迟开销的物理百分比也会按比例发生变化。

例如，如果方法耗用的时间是 100 毫秒，而插桩执行该方法所耗用的时间是 101 毫秒，则开销是 1%。如果方法耗用的时间是 10 毫秒，而插桩将其响应的时间更改为 11 毫秒，则开销是 10%。如果调用此方法的频率不高，则它对应用程序的总体延迟影响很小。但是，频繁调用的插桩方法的总体延迟可能会对应用程序的响应延迟产生影响，即使其开销百分比更小。

与分析每个被调用方法的传统分析器不同， HP Diagnostics 使用的是字节码插桩。它允许用户选择默认插桩，以便将插桩造成的开销降至最低，平均为 3-5%。对于具有较高插桩延迟开销的方法，仅当它们相对于应用程序中的其他组件较少地被调用时，并且当插桩可提供分类活动所需的特定信息时，才会插桩它们。

在自定义应用程序插桩时，还应当考虑 Diagnostics 数据开销。使用的方法越多，探测器需要序列化并通过网络传递到 Diagnostics 服务器的数据就越多。您可以调整探测器的默认配置，以便它可以调整 Diagnostics 的数据量，从而避免对受监控系统的性能造成不必要的影响。探测器调整方法不正确可能会在探测器所在的物理计算机上产生 CPU、内存和网络的开销。有关管理延迟、CPU、内存和网络开销的详细信息，请参阅第 14 章 “高级 .NET 代理配置”。

典型 .NET 应用程序的默认层

根据在点文件中提供的说明， HP Diagnostics 将类和方法的性能度量分组为“层”和“子层”。已定义了默认层，以便集中报告那些使用相似系统资源的应用程序中的处理性能度量。这些层支持您更轻松地隔离并确定可能会影响性能问题的系统区域。

下表列出了为典型 .NET 应用程序定义的默认层和子层。

.NET 层

层	子层	父层
Web 层	IIS	
IIS	ExecutionSteps	
数据库	ADO	
ADO	执行 连接 填充 更新 缓存	数据库
消息	发送方 接收方	
Web 服务	Soap Http WCF	
LWMD		
HTTP 客户端		
出站调用		

11

Diagnostics 服务器高级配置

本章描述 Diagnostics 服务器的高级配置。应由深入了解该产品的经验丰富的用户执行高级配置。请谨慎修改任何组件属性。

本章包括：

- 配置 Business Service Management 样本队列大小和 Web 服务 CI 频率（第 452 页）
- 使用 Diagnostics 服务器配置 Diagnostics 配置页面 （第 453 页）
- 在生产过程中优化 Diagnostics 服务器以处理 更多探测器 （第 453 页）

同步 Diagnostics 组件之间的时间

要正确存储和关联 Diagnostics 数据，必须在 Diagnostics 组件之间同步时间。为了便于数据同步，会调整 Diagnostics 数据并保存将其到 Commander 模式下的 Diagnostics 服务器的同步 GMT 时间。无论 UI 所处的位置的时间如何，都可通过同步来正确地显示数据。

以下各节描述时间同步过程的工作方式，以及如何正确配置组件以使时间同步。

对于在 VMware 主机中运行的探测器收集过程，需要满足其他一些时间同步要求。请参阅 “在 VMware 上运行的探测器的时间同步”（第 468 页）。

了解时间同步

在执行 Diagnostics 时间同步时，首先由 Diagnostics 命令服务器确定其时间和指定时间源所提供的 GMT 时间之间的差异。可使用 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **timemanager.time_source** 属性设置要使用的时间源。

timemanager.time_source 属性的有效值包括：

- **NTP。**表示将 NTP 服务器用作 GMT 时间源。这是默认值。

  要使用的 NTP 服务器将在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中作为 **timemanager.ntp.servers** 属性的值列出。

注意：请确保可从 Diagnostics 服务器访问列表中的 NTP 服务器之一，或将本地 NTP 服务器添加为列表中的第一个服务器。

➤ **BAC。**表示将已注册的 Business Service Management 网关服务器用作 GMT 时间源。

注意：如果将 Business Service Management 配置为使用数据库时间，则还应将 Diagnostics 命令服务器配置为使用此设置作为时间源。

➤ **SERVER。**表示将 Diagnostics 命令服务器用作时间源。

只有在独立模式下使用 Diagnostics 服务器时，才可使用该值。

Mediator 模式的 Diagnostics 服务器可通过确定 Mediator 模式的 Diagnostics 服务器和 Commander 模式的 Diagnostics 服务器之间的时间差异，来同步其时间。

如果 Commander 模式的 Diagnostics 服务器尚未与时间源同步，则 Mediator 模式下的 Diagnostics 服务器也将视为 “未同步” 。 Mediator 模式下未同步的 Diagnostics 服务器会每隔 15 秒尝试同步其时间一次，直至成功同步时间为止。

当 Diagnostics 探测器连接到 Mediator 模式或 Commander 模式下的 Diagnostics 服务器时，便会确定 Diagnostics 服务器与探测器之间的时间差异。

如果探测器尝试连接仍然 “未同步”的 Diagnostics 服务器，则此探测器连接将被禁止并被断开。

由于数据存储基于 GMT，因此，各种组件的时区或夏令时差异并不会导致问题。例如，可以调整在 Diagnostics UI 中显示的数据，使其能够在 UI 运行时所在时区正确显示。

注意：所有数据都将被调整并保存到 Commander 模式 Diagnostics 服务器的同步 GMT 时间中。如果在尚未与时间源正确同步时间的计算机上运行 UI，则 UI 中显示的数据会根据 UI 计算机与同步 GMT 时间之间的时间差异发生偏移。

在 Diagnostics 服务器上配置时间同步

您可以通过执行以下操作来同步 Diagnostics Commander 服务器。

注意：Mediator 模式 Diagnostics 服务器的时间同步设置会被忽略，因为它们的时间会自动与 Commander 模式的 Diagnostics 服务器同步。

要确保 Commander 模式的 Diagnostics 服务器上的时间可以同步，请执行以下操作：

1 设置 Diagnostics 服务器的默认配置，使 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **timemanager.time_source** 属性的值为 **NTP**。

如果 Diagnostics 服务器有 Internet 连接，并且可以连接到 **timemanager.ntp.servers** 属性中指定的可用 NTP 服务器列表中的服务器，则默认配置将可正常工作，而无需任何更改。

由于 Business Service Management 在默认情况下还会使用 NTP 进行时间同步，因此这是推荐设置。

2 如果 Diagnostics 服务器没有 Internet 连接，或者无法连接到 **timemanager.ntp.servers** 属性中指定的可用 NTP 服务器列表，则必须执行以下操作之一：

➤ 设置 Commander 模式 Diagnostics 服务器可访问的本地 NTP 服务器。在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 的 **timemanager.ntp.servers** 属性中，将此本地 NTP 服务器列为第一个条目。

注意：应设置后备 NTP 服务器，以防主 NTP 服务器不可用。

➤ 如果要在 Business Service Management 或 HP “软件即服务” (SaaS) 环境中使用 Diagnostics，则可以将 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **timemanager.time_source** 属性设置为 **BAC**，以表示 Business Service Management。这会使 Diagnostics 服务器连接到已注册的 Business Service Management 核心服务器以确定时间。

注意：要设置 Business Service Management 以使用 Diagnostics，请参阅第 21 章，“设置 Business Service Management 和 Diagnostics 之间的集成”。

➤ 如果要在独立模式下使用 Commander 模式的 Diagnostics 服务器，而不将它与 Business Service Management 配合使用，并且也没有可用于与 NTP 服务器同步时间的 Internet 连接，则可以将 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **timemanager.time_source** 属性设置为 **SERVER**。这会使 Diagnostics 服务器将自己的时间用作时间源。

注意：使用指定的时间源捕获并保存数据之后，建议您不要更改 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **timemanager.time_source** 属性值。如果在捕获数据后更改时间源，可能会对已捕获并保存的数据造成严重损坏。这是因为已保存的数据可能是在 Diagnostics 服务器未与 GMT 同步时捕获的。如果随后捕获的数据是在 Diagnostics 服务器与 GMT 同步后捕获的，则数据将被重新聚合多次，或者被记录到它不属于的时间存储区中。

针对大型安装配置 Diagnostics 服务器

如果要将 Mediator 模式下的 Diagnostics 服务器与超过 20 个探测器配合使用，则建议您修改 Diagnostics 服务器的默认配置。

注意：对于 Commander 模式下的 Diagnostics 服务器，则无需进行这些配置更改，除非它已分配了探测器，使其还充当 Mediator 模式下的 Diagnostics 服务器。

调整堆大小

堆的大小可影响 Mediator 模式 Diagnostics 服务器的性能。如果堆太小，则 Mediator 模式的 Diagnostics 服务器可能会“挂起”一段时间。如果堆太大，则 Mediator 模式的 Diagnostics 服务器可能产生较长时间的垃圾收集延迟（尤其如果没有足够的可用 CPU 资源，如多个 CPU/ 核心或快速的 CPU）。

堆大小的默认值是 512 MB。可在 **server.nanny** 文件中设置堆大小，该文件位于：**<Diagnostics 服务器安装目录 >\nanny\windows\dat\nanny** (Windows) 或 **<Diagnostics 服务器安装目录 >/nanny/solaris/launch_service/dat/nanny/** (Solaris)。

使用以下 VM 参数设置大小 （其中 **???** 表示大小，单位为 MB）：

-Xmx???m

如果遇到 Mediator 模式 Diagnostics 服务器挂起的问题，可以通过更新 **-Xmx???m** 选项中指定的值来增加堆大小。

要调整 Mediator 模式下的 Diagnostics 服务器的堆大小，请执行以下操作：

1 可使用下表确定 Mediator 模式下的 Diagnostics 服务器所需的堆大小：

探测器数	建议堆大小
最多 50 个 Java 探测器	512 MB
最多 100 个 Java 探测器	1400 MB
最多 200 个 Java 探测器	3000 MB （64 位）
最多 10 个 .NET 探测器	350 MB
最多 20 个 .NET 探测器	700 MB
最多 50 个 .NET 探测器	1400 MB

注意：建议的堆大小不应超过计算机物理内存的 75%。如果计算机有 1 GB 内存，则堆大小不得超过 768 MB。

强烈建议在拥有两个以上 CPU 或核心 （建议四个核心）的系统上运行 Diagnostics。在这样的环境中，更改垃圾收集器以同时进行标记和清理：
-XX:+UseConcMarkSweepGC

对于 64 位 JVM，请确保启用此选项：

-XX:+UseCompressedOops

对于 VMware 安装，请遵循 VMware 的 《Enterprise Java Applications on VMware Best Practices Guide》中所描述的最佳实践进行操作。实质上，请使用多个 vCPU 和固定内存分配 （不对磁盘进行扩展和交换），并确保安装 VMware Tools。

2 打开要编辑的 **server.nanny** 文件。此文件位于：

<Diagnostics 服务器安装目录 >\nanny\windows\dat\nanny (Windows)，或 **<Diagnostics 服务器安装目录 >/nanny/solaris/launch_service/dat/nanny/** (Solaris)。

3 在相应的 start_< 平台 > 行中，用计算出的最佳堆大小替换在 **-Xmx???m** 选项中指定的堆大小：

-Xmx???m

继续上面的示例，用 768 MB 替换 **???** 表示的当前堆大小。

-Xmx768m

在 **server.nanny** 文件中修改此行之前，其内容如下：

```
start_nt="C:\MercuryDiagnostics\Server\jre\bin\javaw.exe" -server -Xmx512m
-Dsun.net.client.defaultReadTimeout=70000
-Dsun.net.client.defaultConnectTimeout=30000
"-javaagent:C:\MercuryDiagnostics\Server\probe\lib\probeagent.jar"
-classpath "C:\MercuryDiagnostics\Server\lib\mediator.jar;
C:\MercuryDiagnostics\Server\lib\loading.jar;
C:\MercuryDiagnostics\Server\lib\common.jar;
C:\MercuryDiagnostics\Server\lib\mercury_picocontainer-1.1.jar"
com.mercury.opal.mediator.util.DiagnosticsServer
```

在 **server.nanny** 文件中修改此行之后，其内容如下：

```
start_nt="C:\MercuryDiagnostics\Server\jre\bin\javaw.exe" -server -Xmx768m
-Dsun.net.client.defaultReadTimeout=70000
-Dsun.net.client.defaultConnectTimeout=30000
"-javaagent:C:\MercuryDiagnostics\Server\probe\lib\probeagent.jar"
-classpath "C:\MercuryDiagnostics\Server\lib\mediator.jar;
C:\MercuryDiagnostics\Server\lib\loading.jar;
C:\MercuryDiagnostics\Server\lib\common.jar;
C:\MercuryDiagnostics\Server\lib\mercury_picocontainer-1.1.jar"
com.mercury.opal.mediator.util.DiagnosticsServer
```

调整从探测器获取的数据量

大型调用配置文件不仅占用探测器和服务器之间的大量网络带宽，还占用服务器上的大量 CPU 资源。

如果网络成为瓶颈 （例如，在 Windows 任务管理器中观察到 Mediator 上的网络利用率高于 25%，或者在探测器均已运行的情况下，报告的可用性仍然不足 100%），同时探测器系统的 CPU 尚未充分利用，则您应当通过剪裁来减少所生成的数据，以启用压缩。您还可以减少服务器从探测器获取数据的频率。

探测器上的主要剪裁参数包括：

- 在 **capture.properties** 文件中：
  - maximum.stack.depth = 25
  - maximum.method.calls = 1000（例如，可以设置为 25，以限制调用配置文件中的方法总数）
  - minimum.method.latency = 51ms
- 在 **dispatcher.properties** 文件中：
  - minimum.fragment.latency = 51ms （例如，可以增加到 101ms）。但是请注意，默认情况下，剪裁不会影响综合事务 (BPM/vugen/LoadRunner/Performance Center)，所以将会报告所有这些服务器请求。

有关剪裁的详细信息，请参阅“配置基于服务器请求名称的剪裁”（第 441 页）（服务器）、“配置延迟剪裁和阻断”（第 597 页）和“配置深度剪裁”（第 602 页）（.NET 代理），以及“控制代理上的自动方法剪裁”（第 464 页）（Java 代理）。

要启用压缩，请在探测器上设置 **webserver.properties：rhttp.gzip.replies = true**。这将大幅减少服务器上的网络流量。但是，探测器和服务器需要额外的 CPU 资源来执行压缩。

减少网络流量的另一种方式是更改从探测器获取数据的频率。默认情况下，每隔 5 秒获取趋势一次，每隔 45 秒获取树（调用配置文件）一次。要降低调用树的频率，请更改 Mediator 上 **server.properties** 文件中的 **probe.trees.pull.interval**，例如，根据调用配置文件中包含的方法数，更改为 90 秒或 240 秒。首先，降低调用树的获取频率。如果这还不够，请通过更改 **probe.trends.pull.interval** 来降低趋势获取频率，例如更改为 10 秒。

更改这些参数中的任意一个都需要重新启动探测器或服务器。

其他调整

如果连接了多于 50 个的探测器，请增加用于从探测器获取数据的线程数。对于每个 Mediator，请设置 **probe.pull.max.threads=30**，然后重新启动服务器。

您还可以设置 **webserver.properties, jetty.threads.max=500**，以增加可用于 jetty 的线程数。

如果调用树和趋势文件（另请参阅附录 E，“Diagnostics 数据管理”）在未压缩状态下过大（大于 4 GB），请将一些探测器卸载到新 Mediator。否则，文件的聚合和压缩可能会由于数据量很大而出现延迟。

当很多探测器连接到服务器时，默认的 5 GB 清除设置可能不够。有关详细信息，请参阅 “数据保留”（第 834 页）。

覆盖默认 Diagnostics 服务器主机名

开启防火墙或 NAT 时，或者 Mediator 模式 Diagnostics 服务器的主机被配置为多域设备时， Commander 模式 Diagnostics 服务器可能无法使用在安装 Mediator 模式的 Diagnostics 服务器时指定的主机名，来与 Mediator 模式 Diagnostics 服务器通信。通过 **registered_hostname** 属性，您可以覆盖 Mediator 模式 Diagnostics 服务器用于向 Commander 模式 Diagnostics 服务器注册的默认主机名。

要覆盖 Mediator 模式 Diagnostics 服务器的默认主机名，请将 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中的 **registered_hostname** 属性设置可让 Commander 模式 Diagnostics 服务器与 Mediator 模式 Diagnostics 服务器进行通信的备用计算机名或 IP 地址。

更改默认 Diagnostics 服务器端口

如果 Diagnostics 服务器主机的配置不允许使用默认 Diagnostics 端口，请为 Diagnostics 服务器与探测器和其他 Diagnostics 服务器之间的通信选择其他端口。

重要信息：请确保新端口号未被其他应用程序使用，并且其他 Diagnostics 组件可以与此端口进行通信。

如果决定在部署 Diagnostics 之后使用其他端口号，则必须为部署中的每个指定组件，使用新端口号更新下表中的属性，以确保能够正常通信。

组件类型	属性
Diagnostics 命令服务器	**<Diagnostics 服务器安装目录 >/etc** ➤ webserver.properties – jetty.port ➤ server.properties – commander.url ➤ probe/etc/dispatcher.properties – registrar.url
Diagnostics Mediator 服务器	**<Diagnostics 服务器安装目录 >/etc** ➤ server.properties – commander.url **<Diagnostics 服务器安装目录 >/probe/etc** ➤ dispatcher.properties – registrar.url
探测器	**< 探测器安装目录 >/etc** ➤ dispatcher.properties – registrar.url

将 Diagnostics 服务器从一个主机迁移到另一个主机

以下步骤显示如何将 Diagnostics 服务器从一个主机迁移到另一个主机，并假设新的主机名不同于旧的主机名。

要将 Diagnostics 服务器从一个主机迁移到另一个主机，请执行以下操作：

1. 通过验证进程列表中不存在任何 Java /Javaw 进程，来确保现有 Diagnostics 服务器已经关闭。在 Windows 系统上，可以使用任务管理器进行此操作；在 UNIX 系统上，可以使用 ps 进行此操作。
2. 从 Business Service Management 取消注册 Diagnostics Commander 服务器。
3. 在新的主机上安装新 Diagnostics 服务器。
4. 在 Windows 中，会在安装程序完成后自动启动 Diagnostics 服务器，因此必须关闭 Diagnostics 服务器。

而在 UNIX 中则不会自动启动，因此无需关闭它。

通过验证进程列表中不存在任何 Java /Javaw 进程，来确保 Diagnostics 服务器已经关闭。在 Windows 系统上，可以使用任务管理器进行此操作；在 UNIX 系统上，可以使用 ps 进行此操作。

请确保知道旧 Diagnostics 服务器的主机名 （可在 /archive 目录中查找名称）。

5 删除新 Diagnostics 服务器上的 <Diagnostics 服务器安装目录 >/archive 目录。

6 将 <diagnostics 服务器安装目录 >/archive 文件夹和所有子文件夹从旧服务器复制到新服务器 <diagnostics 服务器安装目录 >/ 中。

7 如果新 Diagnostics 服务器主机名不同于旧 Diagnostics 服务器主机名，则必须重命名 <diagnostics 服务器安装目录 >/archive/mediator-< 主机名 >，使 < 主机名 > 反映新 Diagnostics 服务器主机名。例如，如果旧主机名为 oldhost 而新主机名为 newhost，则应将

<diagnostics 服务器安装目录 >/archive/mediator-<oldhost> 更改为 <diagnostics 服务器安装目录 >/archive/mediator-<newhost>。

8 删除新 Diagnostics 服务器的 <Diagnostics 服务器安装目录 >/storage/ 目录。

9 将 <diagnostics 服务器安装目录 >/storage/ 文件夹和所有子文件夹从旧服务器复制到新服务器 <diagnostics 服务器安装目录 >/ 中。

10 在新服务器上重命名 <diagnostics 服务器安装目录 >/storage/server-< 主机名 >，使 < 主机名 > 反映新 Diagnostics 服务器主机名。例如，如果旧主机名为 oldhost 而新主机名为 newhost，则应将

<diagnostics 服务器安装目录 >/storage/server-<oldhost> 更改为 <diagnostics 服务器安装目录 >/storage/server-<newhost>。

11 将 <diagnostics 服务器安装目录 >/etc 文件夹从旧服务器复制到新服务器 <diagnostics 服务器安装目录 >/ 中，并将新许可证复制到 etc 文件夹。

12 启动新 Diagnostics 服务器，并在 Business Service Management 中注册新 Diagnostics 服务器。

13 如果新服务器为 Commander，则需要在所有 Mediator 上扫描 etc 文件夹，并将旧服务器名更改为新服务器名。仔细检查 dispatcher.properties 文件，确保 Commander 服务器主机名已更改。然后，重新启动所有 mediator。

除非探测器直接向 Commander 服务器报告或您正在迁移探测器所连接的 Mediator 服务器，否则没有必要更改探测器。如果发生这种情况，请扫描探测器系统上的 etc 文件夹，并将旧服务器名更改为新服务器名 （仔细检查 dispatcher.properties 文件，确保 Mediator 服务器主机名已更改）。

针对多域环境配置 Diagnostics 服务器

可以为装有 Diagnostics 服务器的计算机配置多个网络接口卡 (NIC)。 Diagnostics 服务器进程将侦听其主机上的所有接口。某些用户环境不允许应用程序侦听计算机上的所有网络接口。如果您的环境存在这样的限制，请按以下说明配置 Diagnostics 服务器，以侦听特定网络接口。

设置事件主机名

如果 Diagnostics 服务器主机有多个网络接口，并且需要指定 Diagnostics 服务器侦听的主机名，则必须设置 **event.hostname** 属性。

此属性位于以下文件中：

<Diagnostics 服务器安装目录 >/etc/server.properties

取消对 **event.hostname** 属性的注释，并指定主机名的值。

默认情况下，未设置 **event.hostname** 属性。这意味着 Diagnostics 服务器将侦听所有主机名。

修改 jetty.xml 文件

jetty.xml 文件中的一个部分专门用于定义允许 Diagnostics 服务器侦听的接口。默认情况下， Diagnostics 服务器附带的 **jetty.xml** 文件中未定义侦听器。Diagnostics 服务器将侦听所有接口。

要将 Diagnostics 服务器配置为侦听计算机上的特定网络接口，请执行以下操作：

1. 打开 <**Diagnostics 服务器安装目录** >/**etc/jetty.xml**，并找到以下行：

   <Configure class="org.mortbay.jetty.Server">

2. 在此行后面添加以下代码块，并更改 <Set name="Host">……</Set> 以包含 NIC 的 IP 地址。

```
<Call name="addListener">
   <Arg>
    <New class="org.mortbay.http.SocketListener">
     <Set name="Host">127.0.0.1</Set>
     <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
     <Set name="MinThreads">1</Set>
     <Set name="MaxThreads">5</Set>
     <Set name="MaxIdleTimeMs">30000</Set>
     <Set name="LowResourcePersistTimeMs">5000</Set>
     <Set name="ConfidentialPort">8443</Set>
     <Set name="IntegralPort">8443</Set>
    </New>
   </Arg>
 </Call>
```

3 重复上一步骤，为 Diagnostics 服务器要侦听的每个接口添加新代码块并设置 NIC 的 IP 地址。

请确保最后一个 NIC 的侦听器代码后面有 **</Configure>** 标记。

注意：应确保访问 Diagnostics 服务器的组件可以将 Diagnostics 服务器的主机名解析为您在 **jetty.xml** 文件中为主机值指定的 IP 地址。某些系统可能会将主机名解析为 Diagnostics 服务器主机上的其他 IP 地址。有关详细信息，请参阅 “覆盖默认 Diagnostics 服务器主机名”（第 433 页）。

示例 jetty.xml 文件

以下示例显示了 Diagnostics 服务器的 **jetty.xml** 文件，其中， Diagnostics 服务器将侦听系统的环回接口和一个 IP 地址。

```
<!-- Configure the Jetty Server                                             -->
<!-- ======================================================= -->
<Configure class="org.mortbay.jetty.Server">
<!--======================================================= -->
<!-- Configure the Request Listeners                                        -->
  <!--======================================================= -->
<Call name="addListener">
   <Arg>
    <New class="org.mortbay.http.SocketListener">
      <Set name="Host">127.0.0.1</Set>
      <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
      <Set name="MinThreads">1</Set>
      <Set name="MaxThreads">5</Set>
      <Set name="MaxIdleTimeMs">30000</Set>
      <Set name="LowResourcePersistTimeMs">5000</Set>
      <Set name="ConfidentialPort">8443</Set>
      <Set name="IntegralPort">8443</Set>
    </New>
   </Arg>
 </Call>

<-Listen on specific IP Address on this machine for incoming Commander calls->
<Call name="addListener">
  <Arg>
   <New class="org.mortbay.http.SocketListener">
     <Set name="Host">10.241.3.109</Set>
     <Set name="Port"><SystemProperty name="jetty.port" default="2006"/></Set>
      <Set name="MinThreads">1</Set>
      <Set name="MaxThreads">5</Set>
      <Set name="MaxIdleTimeMs">30000</Set>
      <Set name="LowResourcePersistTimeMs">5000</Set>
      <Set name="ConfidentialPort">8443</Set>
      <Set name="IntegralPort">8443</Set>
    </New>
  </Arg>
 </Call>
</Configure>
```

降低 Diagnostics 服务器内存使用量

事务超时时间段是一种安全机制，用于防止 Diagnostics 服务器由于保留旧数据过长时间而使用过多内存。Diagnostics 服务器将保留其接收到的所有事务信息，直到收到事务结束通知 (ELT) 为止，该通知用于告知 Diagnostics 服务器事务已完成。每次 Diagnostics 服务器接收事务数据时都会重置事务超时期。

如果运行 Commander 模式的 Diagnostics 服务器的计算机发生过载（CPU 负载过重或每秒处理的事务过多），或者负载生成器或 Business Service Management 与 Diagnostics 命令服务器之间发生网络连接问题，又或者 Business Process Monitor 与 Business Service Management 之间发生网络连接问题，则 Diagnostics 服务器可能无法收到事务结束通知 ELT。如果在事务超时期限到期时还未收到 ELT，则 Diagnostics 服务器会认为不会收到 ELT，并继续处理事务数据，并释放事务数据所使用的内存。

correlation.txn.timeout 属性可设置事务超时期限的持续时间。如果在 Diagnostics 服务器中发生内存不足的情况，则可以缩短事务超时期限，以缩短 Diagnostics 服务器等待事务结束的时间。调整此属性值时应谨慎，因为多个探测器可能正在将数据发送到 Diagnostics 服务器，并且某个 Diagnostics 服务器中的活动事务可能处于空闲状态。将此属性值设置得过低会导致错误地报告事务。如果需要减少此属性值，请将它设置为测试中的最长事务时间加上 90 秒。

配置基于服务器请求名称的剪裁

通过基于服务器请求名称的剪裁，您可以配置 Diagnostics，使其过滤掉可导致 Diagnostics 服务器性能问题的服务器请求，而无须更改探测器所使用的配置或设备。

注意：基于服务器请求名称的剪裁不能取代为探测器配置的延迟剪裁和深度剪裁。

通过 **<Diagnostics 服务器安装目录 >\etc\trimming.properties** 文件中的 **trim.fragment** 属性，可以指定 Diagnostics 要剪裁的服务器请求碎片的名称。Diagnostics 可以剪裁真实用户和虚拟用户服务器请求的碎片。

默认情况下，会在 **trimming.properites** 中注释掉属性 **trim.fragment.1** 和 **trim.fragment.2**。要指定要剪裁的碎片，请取消对其中一个属性的注释，并按在 Diagnostics 视图中列出的碎片名称，键入要剪裁的碎片名称。如果需要剪裁两个以上的碎片，请创建更多 **trim.fragment** 属性。请确保增大末尾的数字，以确保每个属性名称都是唯一的。例如，下一个 **trim.fragment** 属性将会命名为 **trim.fragment.3**。

由这些属性设置剪裁的事件和碎片将会计入已丢失事件计数和已丢失碎片计数中。

在 HP Diagnostics 中实现复合应用程序搜寻自动化

复合应用程序搜寻 (CAM) 为应用程序服务器（探测器）分组以及连接到这些应用程序服务器的新组件的持续检测 （通过探测器对这些其他组件的后续调用实现）提供了一种便捷方式。

除在 UI 中配置应用程序外， Diagnostics 还为 CAM 提供脚本支持。它允许基于 UI 外部新添加的探测器动态创建新应用程序。

应用程序脚本

用于创建新应用程序的脚本存储在每个 Mediator 的 **etc/appDiscoveryRules.properties** 中。随 Diagnostics 附带的脚本包含一些示例。

通常，应用程序基于实体属性 （如探测器名称、探测器组或服务器请求名称）中提供的某些模式。**/groupby** 路径用于查询 Diagnostics 数据模型和选择实例。此查询路径可用于以下针对应用程序搜寻的脚本，也可用于自动设置阈值和警报。有关详细信息，请参阅 《Diagnostics Data Model and Query API Guide》。

以下示例描述了一种基于部分探测器名称创建新应用程序的一种简单方式。

```
# Example:
#
#  Put all probes with a particular naming pattern into
#  an application.
#
#  If you have a very consistent naming convention for probes, you
#  can auto-insert new probes into the appropriate application.
#
#  In the below example, we put any probe that has a name starting
#  with "cs_" into the "Sales" application.

/groupby[name\='Default\ Client']/probegroup/probe=\
   String probeName = probe.getName();\
   if( probeName.startsWith("cs_") ) {\
      uid=name="Sales";\
   }
```

/groupby 定义（**蓝色粗体文本**）定期查询此 Mediator 上的所有探测器，并根据返回的探测器名称执行脚本（*红色斜体文本* 中的 JavaScript）。在以上示例中，代码为以“**cs_**”开头的所有探测器创建了名为“**Sales**”的应用程序。

除名称以外，还可指定应用程序权限。此脚本包括指定应用程序权限的更多示例。

另外，还可自动包含所有相关探测器实体，如服务器请求和 SQL 语句。要执行此操作，请使用值“**applyAppFilterToProbeContents**”设置变量 **discoveryPolicies**：

```
/groupby[name\='Default\ Client']/probegroup/probe=\
   String probeName = probe.getName();\
   if( probeName.startsWith("cs_") ) {\
      uid=name="Sales";\
      discoveryPolicies="applyAppFilterToProbeContents"; \
   }
```

在环境之间移动复合应用程序

脚本方法提供了一种简单方式，可使应用程序在环境 （如从 QA 到生产环境）之间移动。但是，需要使用脚本创建所有应用程序定义。一个主脚本可用于所有 Mediator 上，即使探测器不向此 mediator 报告。

请注意，应使用可在生产和预生产环境之间使用的命名方案。可通过以下方式实现：

- 将探测器放入特定探测器组，这些组在生产和预生产环境之间保持不变
- 使用探测器命名规则，允许脚本创建应用程序名称，如以上示例所示。

如果探测器在相同探测器组中，且此组名在两个环境之间保持不变 （但探测器名称发生更改），请在脚本中使用 **probegroup.getName()** 访问此探测器组名：

```
/groupby[name\='Default\ Client']/probegroup/probe=\
   String probegroupName = probegroup.getName(); \
   String probeName = probe.getName();\
   if( probegroupName.startsWith("cs") ) {\
      uid=name="Sales";\
      discoveryPolicies="applyAppFilterToProbeContents"; \
   } \
   else if (probegroupName.startsWith("is") ) { \
      uid=name="Information Systems";\
      discoveryPolicies="applyAppFilterToProbeContents"; \
   }
```

此脚本为常规脚本，可在环境之间进行交换。

准备高可用性 Diagnostics 服务器

如果 Diagnostics 部署要求 Diagnostics 服务器具备高可用性，可以为每个 Diagnostics 服务器创建备用 Diagnostics 服务器。这样便可在出现硬件故障或其他 Diagnostics 服务器主机问题时，使用备用 Diagnostics 服务器。

创建备用 Diagnostics 服务器

通过在备用计算机上安装 Diagnostics 服务器，然后将主 Diagnostics 服务器数据定期复制到备用 Diagnostics 服务器中，您可以为每个 Diagnostics 服务器创建备用 Diagnostics 服务器。

要配置备用 Diagnostics 服务器，请执行以下操作：

1 在备用计算机上安装 Diagnostics 服务器。确保要在备用服务器上安装的 Diagnostics 服务器与主服务器上的 Diagnostics 服务器版本相同。

2 在备用 Diagnostics 服务器的主机中使用以下命令，定期将主服务器远程备份到备用服务器中：

```
% cd /opt/MercuryDiagnosticsServer/
% ./bin/remote-backup.sh -h <primary_server_host> -o .
```

使用被复制的 Diagnostics 服务器的主机名替换以上命令中的 **<primary_server_host>**。

这些命令会将 Diagnostic 数据、配置文件、自定义视图、警报和注释以增量方式复制到备用 Diagnostics 服务器中。您可以使用 cron 作业或 Windows 中的计划任务来计划定期备份。

注意：wget 实用程序将通过 HTTP 下载备份。对于 Windows，必须在 Diagnostics 服务器的主机上安装 **cygwin**。您可以从 http://www.cygwin.com/ 获取 **cygwin** 的副本。

故障转移到备用 Diagnostics 服务器

如果主 Diagnostics 服务器的主机发生故障，可将备用 Diagnostics 服务器配置为以主 Diagnostics 服务器运行。

要使备用 Diagnostics 服务器充当主 Diagnostics 服务器运行，请执行以下操作：

1. 更改备用 Diagnostics 服务器的主机名，使其与发生故障的主 Diagnostics 服务器主机的主机名一致。这样，探测器即可在 Diagnostics 服务器启动时重新连接到该 Diagnostics 服务器。
2. 将备用 Diagnostics 服务器作为 Windows 服务启动，或者使用 **bin/server.sh** 或 **bin\server.cmd** 脚本。探测器将重新连接到 Diagnostics 服务器。当探测器与 Diagnostics 服务器的连接断开时，它将会每隔约 30 秒尝试重新连接。
3. 此时，备用 Diagnostics 服务器已成为主 Diagnostics 服务器。按 “创建备用 Diagnostics 服务器”（第 445 页）中所述，配置新的备用 Diagnostics 服务器。

注意：发生故障的 Diagnostics 服务器主机恢复后，请不要将它作为主 Diagnostics 服务器，因为它已丢失了在使用新主 Diagnostics 服务器期间从探测器收集的所有数据。

针对 HP ServiceGuard 配置 Diagnostics （HA 解决方案）

您可以针对 HP ServiceGuard 配置 Diagnostics，以作为 HA （高可用性）解决方案。本节概述了将 HP Diagnostics （7.50 和更高版本）配置为在 HP ServiceGuard (Linux) 下运行的必要步骤。

注意：本节内容假定您熟悉 Diagnostics 和 HP ServiceGuard。

本节中描述的配置步骤也可用于其他 HA 解决方案（例如 Microsoft 群集服务）。

应当在有足够空间存储 Diagnostics 时序数据库 (TSDB) 和其他配置项 （例如用户权限、自定义控制面板屏幕等）的共享磁盘中安装 Diagnostics 服务器。

两个 Diagnostics 服务器 （活动和备用）都需要通过 NTP 或 Business Service Management 进行时间同步。建议不要将 SYSTEM 用作时间同步机制，因为两个 Diagnostics 服务器所使用的 “时钟”必须相同。

Diagnostics 服务器会将主机名用作归档和存储目录下的子目录的前缀。这需要通过指定 **-Dmediator.id=cluster -Dserver.id=<cluster>**（可以用任何其他唯一名称替换 <cluster>），在启动服务器的 Java 命令行上进行覆盖。

示例：<installdir>/bin/server.sh

```
$JAVA1_5_HOME/bin/java -Dserver.id=cluster -Dmediator.id=cluster -server
-Xmx512m
$SERVER_BCP $JAVAOPTS -Dsun.net.client.defaultReadTimeout=70000
-Dsun.net.client.defaultConnectTimeout=30000
-classpath $SERVER_HOME/lib/mediator.jar$PATHSEP$SERVER_HOME/lib/
loading.jar$PATHSEP$SERVER_HOME/lib/common.jar$PATHSEP$SERVER_HOME/
lib/mercury_picocontainer-1.1.jar  com.mercury.opal.mediator.util.DiagnosticsServer
```

注意：所有命令行组成部分都必须位于同一行中。

ServiceGuard 程序包需要应用程序的启动和停止命令。 Diagnostics 的启动命令是 **< 服务器安装目录 >/bin/server.sh** 脚本。

停止命令需要使用一个新脚本，此脚本也在 **< 服务器安装目录 >/bin** 中，并具有以下内容：

< 安装目录 >/bin/stop.sh

```
#!/bin/sh
PID=`ps -ef | grep -v grep | grep DiagnosticsServer | awk '{ print $2 }' `
kill $PID
sleep 10
```

请注意，应确保脚本具有执行权限 (chmod u+x stop.sh)。

在 ServiceGuard 程序包脚本中，添加以下行：

```
stop_command:
  <installdir>/bin/stop.sh

start_command:
  <installdir>/bin/server.sh &
```

请注意，应使用 Diagnostics 的服务器安装目录替换 <installdir>，并确保 server.sh 的末尾有一个 & 符号。

Diagnostics 服务器分配 （LoadRunner/Performance Center 运行）

默认情况下，为 LoadRunner 或 Performance Center 运行所选择的探测器会使用在 **< 探测器安装目录 >/etc/dynamic.properties** 中指定的 Diagnostics 服务器。

可在为运行启动探测器时覆盖该配置。要执行此操作，请在 Diagnostics Commander 服务器上修改映射文件。这样，即可覆盖探测器的 Diagnostics 服务器分配。

在 LoadRunner/Performance Center 与 Business Service Management 的组合环境中运行 Diagnostics 时，这将十分有用。可能有些探测器在 LoadRunner/Performance Center 中运行时所使用的 Diagnostic 服务器与在向 Business Service Management 报告数据时所使用的 Diagnostic 服务器不同。

使用此机制可能比编辑探测器配置文件更方便。

注意：当探测器未运行时，它将使用在 **< 探测器安装目录 >etc/dynamic.properties** 文件中指定的 Diagnostics 服务器。

要覆盖探测器的 Diagnostics 服务器分配，请在 Commander 模式的 Diagnostics 服务器的主机中，修改 **<Diagnostics 服务器安装目录 >\etc** 目录下的 **server_assignment.properties** 文件。

server_assignment.properties 文件的格式为：

<ProbeID> = <Server.id>

- 请使用探测器的 ID 替换 **<ProbeID>**。
- 请使用 Diagnostics 服务器的 ID 替换 **<Server.id>**。

每次 LoadRunner/Performance Center 开始运行时，均会动态读取 **server_assignment.properties** 文件。无需重新启动 Commander 模式的 Diagnostics 服务器，对此文件的更改就会生效。

为 LoadRunner 配置 Diagnostics 服务器 脱机分析文件大小

对于每个运行的 LoadRunner 场景或 Performance Center 测试，Mediator 模式 Diagnostics 服务器都会生成一个 LoadRunner 脱机分析所需的文件，该文件包含在场景中捕获的 Java 数据。此文件可能会变得非常大。请确保 Mediator 模式 Diagnostics 服务器的主机中（用于在场景运行时临时存储文件）和 LoadRunner 控制器主机中 （用于在场景结束后存储文件）均有足够的磁盘空间用于保存 LoadRunner 脱机文件。

估算 LoadRunner 脱机文件的大小

对该脱机文件大小的估算很大程度上取决于数据和捕获数据的速率。

要估算 LoadRunner 脱机文件的大小，请执行以下操作：

1 当 Load Runner 场景启动后，运行负载测试 5 分钟，并监控 Mediator 模式的 Diagnostics 服务器所创建的脱机文件大小。

将 Mediator 模式下 Diagnostics 服务器主机中的脱机文件保存在 **<Diagnostics 服务器安装目录 >/data/< 最新目录 >** 中。脱机文件的扩展名为 **.inuse**。

2 5 分钟后，记录脱机文件的大小。

3 使用上一步中的脱机文件大小乘以 12，推测 1 小时后脱机文件的大小。

4 然后，使用上一步骤中计算出的 1 小时的文件大小乘以预期的实际负载测试运行小时数，即可确定在负载测试结束时的预期脱机文件大小。

5 确定 Mediator 模式下 Diagnostics 服务器的主机和控制器主机上是否有足够磁盘空间用于容纳预期的脱机文件大小。

减小 LoadRunner 脱机文件大小

如果担心脱机文件的大小，可以通过增加 Mediator 模式下 Diagnostics 服务器的脱机聚合时间段，来减小文件大小。这样，即可降低脱机数据的详细程度级别并减小脱机文件大小。

这些属性的默认设置是 **5s**（5 秒），此默认设置会使 Mediator 模式的 Diagnostics 服务器将所有数据聚合到 5 秒的时间片中。增大这些属性的值可使脱机文件变小，这是因为聚合时间段越长，需要存储的数据点就越少。例如，将脱机聚合时间段属性增大到 45s 可将文件大小减小 50-75%。

注意：脱机聚合时间段的调整对脱机文件大小的影响很大程度上取决于应用程序的行为和负载测试的具体情况。

可通过使用以下步骤在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中，修改 Mediator 模式下 Diagnostics 服务器的脱机聚合时间段属性 **bucket.lr.offline.duration** 和 **bucket.lr.offline.sr.duration**。

要通过增加 Mediator 模式下 Diagnostics 服务器的脱机聚合时间段来减小脱机文件大小，请执行以下操作：

1. 确保 Mediator 模式下的 Diagnostics 服务器未参与任何活动的 LoadRunner/Performance Center。这是必要的，因为必须重新启动 Mediator 模式下的 Diagnostics 服务器，才能使以下步骤中所述的属性更改生效。
2. 导航到以下 URL，访问 Mediator 配置页面：

   http://<Diagnostics 服务器主机名 >:8081/configuration/Aggregation?level=60
3. 通过增大 “Load Runner/Performance Center 脱机虚拟用户聚合时间段”属性的设置，来增大脱机虚拟用户聚合时间段。此属性值必须是 5 的倍数，例如 45s。
4. 通过增大 “Load Runner/Performance Center 脱机服务器请求聚合时间段”属性值，来增大脱机服务器请求聚合时间段。此属性值必须是 5 的倍数，例如 45s。
5. 通过单击页面底部的 “提交”，使用修改后的属性值更新 Mediator 模式的 Diagnostics 服务器。

   此时页面顶部会出现一条消息，指出已保存更改，并提醒您重新启动 Mediator 模式下的 Diagnostics 服务器。此时，还会显示 “重新启动 Mediator”按钮。

有关从配置页面更新属性值的详细信息以及用于显示命令按钮的屏幕图像，请参阅 “更改服务器配置”（第 751 页）。

要使配置更改生效，请单击 “重新启动 Mediator”，以重新启动 Mediator 模式下的 Diagnostics 服务器。

配置 Business Service Management 样本队列大小和 Web 服务 CI 频率

以下配置适用于 Business Service Management 集成。

配置 Business Service Management 样本队列大小

默认情况下， Business Service Management 样本队列大小设置为 100。一次创建多于 100 个的样本时，会丢失一些样本，导致 Application Management for SOA 中丢失数据。您可在日志中看到以下消息：“BAC sample being dropped since too many are waiting for delivery”。

可以通过设置服务器属性 **dispatcher.server.wdedelivery.max.queue.size** 以配置 WDEDelivery 队列大小，来增大样本队列大小。

Web 服务 CI 的频率

Diagnostics 会使用默认频率自动创建 Web 服务 CI 并将其添加到运行时服务模型。

您可以更改用于将 Web 服务 CI 添加到运行时服务模型的进程的计时。Web 服务 CI 填充进程具有在 **server.properties** 中定义的以下配置属性：

- **bac.webservice.CI.create.runfrequency** – 两次填充操作之间的间隔秒数 （默认值 = 300，即 5 分钟）
- **bac.webservice.CI.create.query.granularity** – 用于确定要创建的 Web 服务 CI 的 Diagnostics 查询粒度 （默认值 = 1d）

使用 Diagnostics 服务器配置 Diagnostics 配置页面

可以通过 Diagnostics 服务器配置页面设置某些属性值，这些属性值可控制 Diagnostics 服务器与其他 Diagnostics 组件之间的通信方式，以及它处理从探测器接收的数据的方式。

注意：为了确保输入有效的属性值，请使用这些页面更新 Diagnostics 服务器属性，而不要直接编辑属性文件。

有关使用 Diagnostics 服务器配置页面查看和修改 Diagnostics 的信息，请参阅附录 A，“Diagnostics 管理 UI”。

在生产过程中优化 Diagnostics 服务器以处理 更多探测器

单个 Diagnostics 服务器进程可以处理的探测器数在很大程度上取决于每 5 分钟的时间间隔内的唯一服务器请求数，以及每个服务器请求中的方法数和层数。以下优化过程能够增加每个服务器进程可处理的探测器数。

- 默认情况下，Diagnostics 服务器每隔 5 秒从每个探测器获取趋势一次，每隔 45 秒从每个探测器获取树一次。如果单个 Diagnostics 服务器进程要处理的探测器超过 25 个，则可对其进行优化，以降低获取趋势和树的频率。在生产过程中，建议的最佳设置是每隔 30 秒获取趋势一次，每隔 120 秒获取树一次。可以在 **<Diagnostics 服务器安装目录 >\Server\etc\server.properties** 中配置这些值，如下所示：

```
# The interval at which to pull trends from probes
probe.trends.pull.interval = 30s

# The interval at which to pull trees from probes
probe.trees.pull.interval = 120s
```

- 服务器进程的最大堆大小由服务器启动脚本中的 **-Xmx** 参数确定。最大堆大小的默认设置为 512 MB。可根据探测器中的负载增大最大堆大小。第 1 章，“准备安装 HP Diagnostics”。中提供了基于要处理的探测器数的最大堆大小建议值。
- 如果服务器要处理的探测器超过 30 个，则强烈建议在生产过程中为 Diagnostics 服务器使用 1 Gbps 的链接。
- 如果单个服务器进程要处理的探测器超过 75 个，请增加 jetty 线程数。用于确定线程数的一般经验法则是，将探测器数乘以 2 再加 40。默认值是 200。可通过在 **<Diagnostics 服务器安装目录 >\Server\etc\webserver.properties** 中修改 **jetty.threads.max** 属性，来增加 jetty 线程数。例如：

```
jetty.threads.max=300
```

12

高级 Java 代理与应用程序服务器配置

本节讨论 Diagnostics Java 代理和应用程序服务器环境的高级配置。应由深入了解该产品的经验丰富的用户执行高级配置。请谨慎修改任何组件属性。

本章包括：

- 配置 SOAP 错误有效负载数据 （第 491 页）
- 配置 REST 服务 （第 493 页）
- JMS 临时队列 / 主题的自定义分组 （第 493 页）
- 配置 SQL 查询分析 （第 493 页）
- 配置为服务器请求显示的应用程序名称 （第 494 页）
- 在 Java Profiler UI 中维护探测器设置 （第 495 页）
- 为 JUnit 测试生成性能报告 （第 503 页）

高级配置概述

以下要点列出了在配置环境时需要参考的探测器配置信息源。

- 如果要设置探测器，以阻止其他人在 Profiler 模式下使用，请参阅 “禁用 Java Diagnostics Profiler”（第 457 页）。
- 要将日志消息发布到较低级别消息的探测器日志中，请按照 “控制探测器日志记录”（第 458 页）中描述的内容调整日志级别。
- 如果在同一主机上安装了多个代理，请确保每个代理的日志消息储存在不同的文件中，如 “更改探测器的日志目录”（第 459 页）中所述。
- 要检查通常会从 Diagnostics 报告的度量数据中剪裁的处理性能，请降低剪裁级别或完全关闭剪裁功能，如“控制代理上的自动方法剪裁”（第 464 页）中所述。
- 如果代理和 Diagnostics 命令服务器之间存在代理服务器，则必须正确地设置属性，以将 Diagnostics 命令服务器的 URL 告知代理，具体请参阅 “为代理服务器配置代理”（第 467 页）。
- 如果在 HP 软件即服务 (SaaS) 环境中安装了 Java 代理，请禁用代理和 Diagnostics Mediator 服务器之间的反向 http (rhttp) 通信，具体请参阅 “在 VMware 上运行的探测器的时间同步”（第 468 页）。

- 如果要在虚拟环境中运行，请参阅 “在 VMware 上运行的探测器的时间同步”（第 468 页）。
- 如果需要限制异常数据的数量，请参阅 “限制异常树数据”（第 468 页）。
- 如果要使用某些需要执行属性文件更改操作的收集选项，请参阅本节中的其他主题，例如 “配置用户 ID”（第 480 页）。

禁用 Java Diagnostics Profiler

您可以在 Java 代理 上禁用 Diagnostics Profiler for Java，以避免其被意外访问。如果禁用了 Java Diagnostics Profiler，则不能通过以下 Java Diagnostics Profiler URL 访问用户界面：http://< 探测器主机 >:< 探测器端口 >/profiler。

要禁用 Java Diagnostics Profiler，请在 <探测器安装目录>/etc/probe.properties 中将 **disable.profiler** 属性设置为 true。

disable.profiler 的默认值为 **false**。如果要在禁用 Java Diagnostics Profiler 后再次将其启用，请将 disable.profiler 属性的值由 **true** 更改为 **false**。

控制探测器日志记录

可以使用探测器属性控制探测器记录的消息级别，还可以更改发布日志消息的位置。

控制日志消息级别

属性文件 **<探测器安装目录>/etc/logging.properties** 中的 **lowest_printing_level** 属性可控制由探测器输出到标准输出中的消息级别。此属性的默认设置是 **OFF**。该设置几乎将阻止向控制台输出任何代理消息。

通过更改 **lowest_printing_level** 属性值，可以动态地调整记录级别。保存属性文件后，对消息记录级别的更改就会生效。

lowest_printing_level 属性的有效值包括：

属性值	描述
OFF	不记录消息。
DEBUG	记录所有消息。
INFO	记录 “信息”、“严重”和 “警告”级别消息。
WARN	记录 “警告”和 “严重”级别消息。
SEVERE	记录 “严重”级别消息。

更改探测器的日志目录

探测器的默认日志目录位置为 **<探测器安装目录>/log**。如果在同一主机中有多个探测器，可以使用 **log.dir** 属性更改每个探测器的日志目录位置。可以使用两种方法设置该属性：

- 可以在属性文件 **<探测器安装目录>/etc/probe.property** 中设置 **log.dir** 属性的值。
- 可以在应用程序服务器的启动命令行中将 **log.dir** 属性的值指定为 JAVA 系统属性，如以下示例所示：

```
-Dprobe.log.dir=/path/to/log
```

有关在启动命令行中指定 log.dir 属性的详细信息，请参阅 “为代理服务器配置代理”（第 467 页）。

设置探测器的主机名

探测器的主机名使用 Diagnostics Commander 服务器注册探测器。Diagnostics Commander 服务器使用探测器的主机名与探测器进行通信，并将其与探测器监控的服务器系统度量数据一起显示在 Diagnostics 视图中。

正常情况下，探测器可以检测到其主机的计算机名。在某些情况下，服务器的配置可能存在问题，导致探测器无法检测到正确的主机名。如果安装了防火墙或 NAT，或者代理主机被配置为多域设备，则代理可能无法正确地检测到其主机。

如果探测器无法检测到其主机名，可以指示探测器通过基于套接字连接的反向 DNS 查找来获取主机名，也可以使用探测器属性来指定主机名。

指示探测器使用反向 DNS 查找

如果探测器的主机配置阻止探测器检测主机名，可以通过设置 **server.host.name.lookup** 属性，指示探测器使用反向 DNS 查找来检测主机名。此属性位于 **< 探测器安装目录 >/etc/dispatcher.properties** 文件中。

server.host.name.lookup 属性的默认值为 “false”。该值指示探测器不使用反向 DNS 进行查找。可将此属性设置为 “true”，以指示探测器使用反向 DNS 进行查找。

手动指定探测器主机名

registered_hostname 属性支持您手动设置探测器的主机名，并禁止探测器执行自动查找。

要为探测器设置默认的主机名，请将 **registered_hostname** 属性（该属性位于属性文件 **< 探测器安装目录 >/etc/dispatcher.properties** 中）设置为计算机名称或 IP 地址。

设置 **registered_hostname** 属性后，将禁止自动查找主机名称。

注意：只有在使用 LoadRunner、Performance Center 或 Diagnostics Standalone 的测试环境中，才可能需要因 NAT 或防火墙问题而设置 **registered_hostname** 属性。

如果在使用 Business Service Management 或 Diagnostics Standalone 的生产环境中设置 **registered_hostname**，则您指定的名称将在系统运行状况中显示为主机名。

指定其他探测器 IP 地址

通过 **probe.host.ip.address.override** 属性（位于属性文件 **<探测器安装目录>/etc/dispatcher.properties** 中），可以覆盖探测器的 IP 地址。如果希望探测器提供具有其他 IP 地址（例如允许服务器通过通道与探测器进行通信的虚拟 IP）的服务器，则可以使用此属性。

设置活动产品模式

通常，系统将基于您在安装程序中输入的选项来设置“Java 代理”模式。但是，您也可以按照本节中的描述，手动设置此模式。

可在不同模式下设置 Java 代理以执行以下操作：

- 监控从开发到预生产测试再到生产等各环境中的应用程序
- 与其他 HP 软件产品配合使用
- 用作不向服务器或其他 HP 软件产品报告数据的 Diagnostics Java Profiler

Java 代理所使用的模式由属性文件 **<探测器安装目录>/etc/probe.properties** 中的 **active.products** 属性值确定。

active.products 属性中的模式值还用于针对许可证容量确定使用情况（请参阅“基于当前连接的探测器许可证信息”（第 83 页））。对于 Diagnostics，有两种类型的 LTU（要使用的许可证）：

- AM - 在企业模式下使用产品时，通常用于生产环境。
- AD - 在预生产加载测试环境中使用产品时，探测器位于 LoadRunner 或 Performance Center 运行中。

可在最初安装 Java 代理 时设置 **active.products** 属性值。

要更改 **active.products** 属性的值，可以编辑属性文件并重新启动应用程序服务器。或者可以重新运行 Java 代理安装模块并使用 “更改”选项将模式设置为 Diagnostics Profiler 模式 (PRO)、应用程序管理 / 企业模式 （用于 Diagnostics (Enterprise)）和 / 或 TransactionVision (TV) 或 Diagnostics 模式 （用于 LoadRunner/Performance Center (AD)）。

注意：要在企业模式下使用独立的 Diagnostics Profiler for Java 或与其他 HP 软件产品集成，请联系 HP 软件客户支持以购买 HP Diagnostics。

如果要在所使用的 HP 软件产品的用户界面中查看 Diagnostics 数据，则必须执行额外一些配置步骤。有关与 Business Service Management、 LoadRunner 或 Performance Center 集成的详细信息，请参阅 “设置与其他 HP 软件产品的集成”（第 691 页）中的部分。以下各节将说明如何配置 **active.products** 属性的每个产品模式。

PRO 产品模式 – Diagnostics Profiler for Java

设置 PRO 模式后，代理会收集一些性能度量，并在可通过代理主机上的 URL 访问的独立 Diagnostics Java Profiler 用户界面中显示这些度量。

如果将 Java 代理 作为试用版 Java Diagnostics Profiler 的一部分运行，则对代理的限制也会限制它可处理的负载量。

如果要将 Java 代理 作为完整 Diagnostics Enterprise 产品的一部分运行，或将它与其他 HP 软件产品配合使用，则 Profiler 启用时将没有负载限制。

PRO 模式不可用于针对许可证容量确定使用情况。

企业产品模式

在企业模式下配置后，代理可以与 Business Service Management、LoadRunner、Performance Center 等 HP 软件产品配合使用，也可以作为完整的 Diagnostics Enterprise 产品使用。即便如此，您还必须执行其他配置以启用这些集成（有关详细信息，请参阅 “设置与其他 HP 软件产品的集成”（第 691 页）的部分）。

在企业模式下，也会将数据发送到 Diagnostics Java Profiler。

在企业模式下，您还必须使用 Diagnostics 服务器注册代理 （请参阅 “使用 Diagnostics 服务器注册代理”（第 148 页））。

Java 代理默认使用企业模式 （如果没有指定 AD 或 AM 模式）。在企业模式下，将针对 AM 许可证容量计算代理。

AM 产品模式

在 AM 模式下， Java 代理将会捕获所有插桩数据。可以设置 AM 模式，避免 Business Service Management 生产部署中的代理被意外包含在 LoadRunner 或 Performance Center 运行内。在 AM 模式下，代理不会被列为 LoadRunner 或 Performance Center 的可用代理。

将始终针对 AM 许可证容量计算 AM 模式下的代理。

AM 模式会取代除 AD 模式之外的所有其他模式。

AD 产品模式

在 AD 模式下，Java 代理仅在从 LoadRunner 或 Performance Center 运行时捕获数据，并将结果存储在用于该运行的特定 Diagnostics 数据库中，例如 Default Client:21。

当代理处于 AD 模式下时，除非探测器参与了 LoadRunner/Performance Center 运行，否则代理将不会使用资源或将任何数据发送到服务器。

有关如何设置 LoadRunner 集成的信息，请参阅第 23 章，“设置 HP LoadRunner 与 HP Diagnostics 集成”；有关如何设置 Performance Center 集成的信息，请参阅第 24 章，“设置 Performance Center 以使 Diagnostics”。

使用此模式可避免 QA 环境中的代理使用其他资源，以及在未运行负载测试时，仍然继续向 Diagnostics 服务器报告数据。

在 AD 模式内运行探测器的另一个好处在于，仅在 LoadRunner 或 Performance Center 运行中时，才针对 AD 许可证容量计算 AD 模式下的探测器。例如，如果在 LoadRunner/Performanace Center AD 模式下安装了 20 个探测器，但一次测试中只有 5 个探测器，则只会针对 AD 许可证容量计算 5 个探测器。

TV 产品模式

此模式会将事件发送到 Transaction Vision，且可以与其他模式同时使用。TV 模式不可用于针对 HP Diagnostics 许可证容量确定使用情况。

控制代理上的自动方法剪裁

代理的默认配置包括用于控制方法剪裁操作的设置。可以根据方法的执行时间来控制剪裁，这称为 “延迟”；也可以根据方法调用的 “堆栈深度”来控制剪裁。默认配置将指示探测器同时根据延迟和深度来进行剪裁。

您可以降低剪裁级别，也可以完全关闭剪裁。可以使用 **< 探测器安装目录 >/etc/capture.properties** 中的 **minimum.method.latency** 和 **maximum.stack.depth** 属性来控制剪裁。

控制延迟剪裁

完成时延迟大于或等于 **minimum.method.latency** 属性值的方法将被捕获，而完成时延迟小于该值的方法将被剪裁，以避免关注度不高的方法产生开销。

注意：在以下情况中，不会在延迟时间小于剪裁属性时剪裁延迟：

- 方法是调用树的根。
- 方法抛出了异常。

如果必须捕获所有方法的信息，请降低 **minimum.method.latency** 属性的值，或将其设置为零。

设置 **minimum.method.latency** 属性时，请注意以下事项：

- **minimum.method.latency** 属性值越小，对应用程序性能造成不利影响的可能性就越大。
- 根据您所使用的平台，以及是否使用本机时间戳 (**use.native.timestamps** = false)，建议将该值的增量指定为大于或等于 10 毫秒。

控制深度剪裁

在小于或等于 **maximum.stack.depth** 属性值的堆栈深度处调用的方法将被捕获。在大于此深度的堆栈深度处调用的方法将被剪裁，以避免关注度不高的方法产生开销。

以下是一个示例：

如果 **maximum.stack.depth** 为 3 and /login.do calls a() calls b() calls c()，则只会捕获 /login.do、 a 和 b。

请注意，将 **maximum.stack.depth** 设置为较低的值可以显著减少捕获所导致的开销。

配置 URI 截断、映射和剪裁

任何 HTTP/S 服务器请求 URI 在被探测器报告之前均可以进行转换。可能的转换基于由 **dynamic.properties** 中 **uri.pattern.replace** 属性控制的正则表达式匹配和替换。属性的值是要在每个 URI 上尝试执行的模式替换操作的逗号分隔列表。

当发现过多服务器请求，并且希望使用一个能够聚合这些 URI 的简单服务器请求 URI 来替换众多服务器请求 URI 时，此转换十分有用。

使用 s/pattern/replace/ 语法截断或映射 URI。要执行多个操作，请使用逗号分隔的列表。操作将按顺序执行。

例如，要在某个字符串之前截断，请先匹配该字符串及其后的任意字符，然后将其替换为空。在此例中，“$”匹配行尾。

```
s/string.*$//
```

URI 截断和映射下的 dynamic.properties 文件中的注释提供详细信息和更多示例。

重要信息：过度使用此功能会影响性能。

如果服务器请求太多，则还可以使用 **capture.properties** 文件中的 **maximum.uri.pathsegments** 属性将服务器请求剪裁成 n 段路径。

默认值为 -1，即禁用该属性。对于 SaaS 环境中报告的探测器（在 Java 代理安装期间选择的 SaaS），maximum.uri.pathsegments 将自动设置为 2，以确保发送到 HP 托管的服务器的服务器请求数据量不会太大。

例如，设置 2 将确保不会超过两个路径段。因此，
http://localhost:8080/path1/path2/path3 将剪裁成
http://localhost:8080/path1/path2/。

可以使用 uri.pattern.replace，然后设置 maximum.uri.pathsegments 以剪裁成特定数目的路径段。或者仅适用一个属性或其他方法。

为代理服务器配置代理

重要信息：本节仅适用于将 Java 代理与 Diagnostics 服务器配合使用的情况。

可使用两个属性为 Java 代理指定 Diagnostics 命令服务器的 URL。所设置的属性取决于是否存在代理服务器。

- **dispatcher.properties** 中的 **registrar.url**

  在安装代理时，将设置 **< 探测器安装目录 >\etc\dispatcher.properties** 中的 **registrar.url** 属性。如果代理和 Diagnostics 命令服务器的 URL 之间存在直接连接，则代理将使用该属性的值。

- **webserver.properties** 中的 **registrar.url**

  如果存在代理服务器，则必须在 **< 探测器安装目录 >\etc\webserver.properties** 文件中设置 **registrar.url** 属性，以指明 Diagnostics 命令服务器的 URL。

在 VMware 上运行的探测器的时间同步

对于在 VMware 宾客系统中运行的探测器， VMware 宾客系统和基础 VMware 主机之间的时间必须同步。如果未正确同步时间，则 Diagnostics UI 可能不会准确显示在 VMware 宾客系统中运行的探测器的度量，或者不显示任何度量。

应当根据 VMware 产品白皮书 (http://www.vmware.com/pdf/vmware_timekeeping.pdf) 中有关时间同步的“Synchronizing Hosts and Virtual Machines with Real Time”部分中的建议来同步时间。必须在装有 Diagnostics 探测器的每个 VMware 宾客操作系统中安装 VMware 工具。必须打开 VMWare 工具中的时间同步选项。

只有当宾客操作系统时间最初被设置为早于 VMware 主机时间时，VMware 工具中的此选项才可用。有关如何安装 VMware 工具的说明，请参阅 VMware ESX Server 的 “Basic System Administration”文档。如果要使用任何非 VMware 时间同步软件（如网络时间协议），则应当在 VMware ESX Server 服务控制台中运行该软件。

当 attempt.vmware.timestamp.adjustments 探测器属性设置为 true 时，如果在 VMware 宾客系统上运行探测器时遇到负延迟问题，则应在探测器的 **etc/capture.properties** 文件中设置以下属性：

```
use.vmware.timestamp.workaround=true
```

当 use.vmware.timestamp.workaround 设置为 true 后，探测器将使用备用调用获取 VMware 主机时间戳，以解决负延迟问题。

限制异常树数据

代理将收集异常信息，并使用该信息来构建异常实例树。异常实例树可为 Diagnostics UI 中出现的异常信息 （例如堆栈跟踪）提供数据。

默认情况下，受监控的应用程序中发生的所有异常都是异常实例树的候选项。但是，由于无关的异常会使数据显示、数据收集和传输操作过载，因此，收集所有异常信息通常是不可取的。因此，您可以限制要为其收集数据的异常类型。例如，通过筛选基于服务器的应用程序错误（如 **javax.naming.AuthenticationException**），可使异常树包含更多特定于应用程序的错误。

可以通过限制特定异常类型或限制异常类型的数量来控制所收集的异常树数据。

限制特定异常类型

通过在 **< 探测器安装目录 >\etc\dispatcher.properties** 文件中设置 **exception.types.to.exclude** 和 **exception.types.to.include** 属性，可以控制要在收集操作中排除和包含的特定异常类型：

- **exception.types.to.exclude**

  设置此属性可忽略一个或多个指定的异常类型。每个指定类型的子类型也会被忽略，但由 **exception.types.to.include** 属性指定的子类型除外。

- **exception.types.to.include**

  设置此属性可指定要包含哪些已被指定为排除的异常或其子类型 （如果有）。还将包含指定为要排除的任何异常类型的子类型。

这两种属性都采用完全限定的异常类型名称的列表，该列表以逗号分隔。对 **dispatcher.properties** 文件的更改将立即生效，而无需重新启动应用程序。

限制异常类型的数量

您可以通过在 **server.properties** 中设置 **exception.instance.tree.count** 属性，指定不同类型的异常数量，从而限制要收集的异常树数据。默认情况下此属性设置为 4，意味着只会使用在探测器数据收集循环中遇到的前 4 个异常类型来构建异常树。可以增大或减小此设置。

示例

以下示例会忽略 ClassNotFoundException 类型的异常及其全部子类型。

```
...
exception.types.to.exclude=javax.naming.AuthenticationException
```

以下示例会排除 java.lang.IOException 类的某些子类型，如下图所示：

```
...
exception.types.to.exclude=java.io.IOException,java.io.InvalidClassException
exception.types.to.include=java.io.ObjectStreamException
```

下图显示了 java.io 类层次结构中包含的和不包含的异常类型：

Diagnostics 探测器管理页面

可以使用 Diagnostics 探测器管理页面来配置 Java 代理 和 Profiler 设置。可以直接从浏览器中访问 Diagnostics 探测器管理页面。

访问 Diagnostics 探测器管理页面

在浏览器中打开 Diagnostics 探测器管理页面。

要访问 Diagnostics 探测器管理页面，请执行以下操作：

1 在浏览器中导航到 http://< 探测器主机 >:< 探测器端口 >。

会将探测器分配到从 **35000** 开始的第一个可用端口。

此时将打开管理页面。

2 选择与要执行的活动对应的菜单选项。

- **打开 Diagnostics Profiler。**打开 Java Diagnostics Profiler。
- **高级选项。**打开组件页面。有关详细信息，请参阅 “Diagnostics 探测器组件页面”（第 472 页）。
- **管理授权和身份验证。**根据探测器配置方式的不同，将通过此选项访问不同的页面。

- 如果探测器被配置为可与 Diagnostics 服务器配合使用，则可以通过探测器连接的 Diagnostics 命令服务器来管理探测器 (Profiler) 授权和身份验证设置。单击此选项时，将会重定向到该 Diagnostics 命令服务器。有关详细信息，请参阅附录 B，“用户身份验证和授权”。
- 如果探测器被配置为仅作为 Profiler 使用，并且未与任何 Diagnostics 服务器连接，则此选项会打开用户管理页面，您可以在该页面中创建、编辑、删除用户以及更改用户权限。有关详细信息，请参阅 “Diagnostics Java Profiler 的身份验证和授权”（第 474 页）。

Diagnostics 探测器组件页面

可以在组件页面中打开 Java Diagnostics Profiler，然后访问用户管理页面。

要访问组件页面，请执行以下操作：

1. 按 “访问 Diagnostics 探测器管理页面”（第 471 页）中所述打开 Diagnostics 探测器管理页面。
2. 单击 “高级选项”。
3. 如果出现提示，请输入用户名和密码。

   此时将打开组件页面。

4 单击下列选项之一：

- **query。**供开发者内部使用。
- **inst**。包含各种插桩选项。有关探测器插桩的详细信息，请参阅“Java 应用程序的自定义插桩”（第 297 页）。
- **security。**根据探测器配置方式的不同，将通过此选项访问不同的页面。
  - 如果探测器被配置为可与 Diagnostics 服务器配合使用，则可以通过探测器连接的 Diagnostics 命令服务器来管理探测器 (Profiler) 授权和身份验证设置。单击此选项时，将会重定向到该 Diagnostics 命令服务器。有关详细信息，请参阅 “用户身份验证和授权”（第 755 页）。
  - 如果探测器被配置为仅作为 Profiler 使用，并且未与任何 Diagnostics 服务器连接，则此选项会打开用户管理页面，您可以在该页面中创建、编辑、删除用户以及更改用户权限。有关详细信息，请参阅 “Diagnostics Java Profiler 的身份验证和授权”（第 474 页）。
- **scheduler。**可用于查看和控制计划的定期后台任务。对于 ServerCommunication 计划程序或 sharedInfrequentEventScheduler，可以查看其中的任务的状态和数量。可以为每个任务选择 RUN NOW 或 DELETE 等操作。
- **infrequentLogger。**查看非常用日志记录表中的条目的当前状态。
- **files。**安装目录浏览器，用于上载和下载属性文件、日志文件等。

Diagnostics Java Profiler 的身份验证和授权

将 Java 代理仅安装为 Profiler（未连接到任何 Diagnostics 服务器）时，可以在 Diagnostics 探测器用户管理页面中管理 Profiler 的身份验证和授权。

注意：如果 Java 代理被配置为与 Diagnostics 服务器配合使用，则可以通过探测器连接的 Diagnostics 命令服务器来管理探测器 (Profiler) 的身份验证和授权设置。有关详细信息，请参阅 “用户身份验证和授权”（第 755 页）。

要管理独立 Java Diagnostics Profiler 的用户的身份验证和授权，请执行以下操作：

1 访问 Diagnostics 探测器管理页面

在浏览器中导航到 http://< 探测器主机 >:< 探测器端口 >。会将探测器分配到从 35000 开始的第一个可用端口。

此时将打开 Diagnostics 探测器管理页面。

2 选择 “管理授权和身份验证”，以打开用户管理页面。

在用户管理页面中，可以创建新用户、为用户分配权限、更改现有用户的密码，以及删除用户。

要创建新用户，请执行以下操作：

1 单击 “创建用户”，在 “新建用户名” 框中输入用户名称，然后单击 “OK”。此时，新用户将出现在用户名列表中。

2 在代表新用户的行中，在 “密码” 框中键入密码，然后在 “确认密码” 框中重新键入密码以进行确认。

3 在“< 当前用户 > 的密码”框中，键入当前登录用户的密码，然后单击“保存更改”。

要为用户分配权限，请执行以下操作：

1 转至代表相关用户的行，然后选中代表不同权限的复选框。

可以将以下权限级别分配给 Java Diagnostics Profiler 用户：

权限	描述
查看	用户可以在 UI 中查看 Profiler 数据。
执行	用户可以执行垃圾收集，并清除由 Profiler 保存的性能数据。
更改	用户可以执行具有潜在风险的操作，例如获取堆转储或更改插桩。

注意：权限级别 “rhttpout” 和 “系统” 仅供内部使用。

每个权限级别均独立存在。各个级别之间没有权限继承关系。必须向用户授予执行功能所需的所有权限级别。

2 在“<当前用户>的密码”框中，键入当前登录用户的密码，然后单击“保存更改”。

要更改现有用户的密码，请执行以下操作：

1 转到代表相关用户的行中，在 “密码” 框中键入密码，然后在 “确认密码” 框中重新键入密码以进行确认。

2 在“<当前用户>的密码”框中，键入当前登录用户的密码，然后单击“保存更改”。

要删除用户，请执行以下操作：

1 在 “< 当前用户 > 的密码”框中，键入当前登录用户的密码。

2 单击对应于要删除的用户的 “删除用户”按钮。

此时将打开一个消息框，询问您是否要删除所选用户。

3 单击 “OK”删除用户。

配置 CPU 时间度量收集

CPU 时间度量出现在事务视图、探测器视图、调用配置文件视图和 Portal 组件的 “详细信息”窗格中。可以启用、禁用和配置 CPU 时间度量收集。CPU 时间度量包括 “CPU (平均值)”和 “CPU (合计)”。如果禁用 CPU 时间度量收集，或者没有为方法配置 CPU 时间度量收集，则这些度量将显示为 “暂缺”。

CPU 时间度量依赖于 CPU 时间戳。通常以下平台可支持 CPU 时间戳：Windows、 Solaris、 AIX、 HP-UX 和 Linux 内核 2.6.10 或更高版本 （例如 RedHat 5.x 和 SUSE 10.x）。

注意：对 CPU 时间戳的支持不仅会随操作系统的不同而发生变化，也受到平台体系结构 （例如， SPARC 和 x86）的影响。
有关特定平台版本和体系结构是否支持 CPU 时间度量的最新信息，请参阅位于 http://support.openview.hp.com/sc/support_matrices.jsp 的 Diagnostics Support Matrix。

重要信息：在 VMware 中， CPU 时间度量来自于访客操作系统，并受 VMware 虚拟计时器的影响。请参阅位于 http://www.vmware.com/pdf/vmware_timekeeping.pdf 的有关时间同步的 VMware 白皮书，以及“在 VMware 上运行的探测器的时间同步”（第 468 页）。

默认情况下，已为服务器请求启用 CPU 时间度量收集操作。但是您可以禁用 CPU 时间度量收集，并在属性文件或 Java Diagnostics Profiler UI 中配置 CUP 时间度量收集。可以配置对以下 CPU 时间度量的收集：

- 仅针对服务器请求
- 服务器请求和 Portlet 方法
- 服务器请求和所有方法

对于 Java 代理， CPU 时间度量收集操作由以下两个属性控制：

- **< 探测器安装目录 >\etc\capture.properties** 中的 **use.cpu.timestamps** 属性。

  默认情况下，此属性设置为 **true**，将启用 CPU 时间度量收集。下面列出的第二个属性可控制对所有 CPU 时间戳的收集操作。如果将 use.cpu.timestamps 属性设置为 false，则不会为探测器报告的任何服务器请求或方法收集 CPU 时间度量。

- **< 探测器安装目录 >\etc\dynamic.properties** 中的 **cpu.timestamp.collection.method** 属性。

注意：配置 CPU 时间戳收集时请谨慎，因为这会增加 Diagnostics 的开销。开销增加的原因在于，在收集时间戳时需要为每个方法执行额外的调用。

可以将 **Cpu.timestamp.collection.method** 设置为以下值之一：

- **0 –** 不收集 CPU 时间戳。
- **1 –** 仅对服务器请求收集 CPU 时间戳。

  默认值为 1，表示可在服务器请求级别而不是事务级别报告 CPU 时间。但是，如果在属性文件中删除或注释掉此设置，则默认值为 0。
- **2 –** 对所有服务器请求和所有方法收集 CPU 时间戳。
- **3 –** 对所有服务器请求以及为 Portal 组件插桩的生命周期方法收集 CPU 时间戳。

此外，还可使用 Java Diagnostics Profiler 中的 “配置”选项卡设置 **cpu.timestamp.collection.method** 属性，如下所示：

1 在 **Profiler** UI 中，选择 “配置”选项卡。无需启动 Profiler 即可更改此探测器配置。

2 在 “配置”屏幕中，从下拉列表中选择 “采集 CPU 时间戳”选项。

CPU 时间戳收集方法	描述
从不	不采集 CPU 时间戳。
仅针对服务器请求	仅为服务器请求收集 CPU 时间戳。
针对服务器请求和 Portlet 方法	对所有服务器请求以及为 Portal 组件的插桩生命周期方法收集 CUP 时间戳。
针对服务器请求和所有方法	对所有服务器请求和所有方法收集 CPU 时间戳。

3 完成更改后，单击 “应用更改”。

注意：更改将立即生效，而无需重新启动应用程序 （或探测器）。

配置用户 ID

可以按特定 Web 服务的用户分组 Web 服务度量。然后，将为该用户聚合度量，并在“服务 (按用户 ID)”或“操作 (按用户 ID)”等 SOA 服务视图中显示度量。可通过多种方式定义用户 ID：

- SOAP 标头中的值
- SOAP 包络中的值
- SOAP 正文中的值
- HTTP 标头中的值
- JMS 队列名称 （或主题名称），用于 JMS Web 服务上的 SOAP
- JMS 消息属性，用于 JMS Web 服务上的 SOAP
- JMS 消息标头，用于 JMS Web 服务上的 SOAP
- 特定的 IP 地址
- IP 地址的范围

重要信息：定义基于 SOAP 标头、包络或正文的用户 ID 时，需要使用用于 Java 探测器的 Diagnostics SOAP 消息处理程序。对于某些应用程序服务器，已在 Diagnostics 中提供特殊插桩以自动加载 Diagnostics SOAP 消息处理程序。

但是，仍需对 WebSphere 5.1 JAX-RPC 和 Oracle 10g JAX-RPC 进行一些手动配置。有关详细信息，请参阅“加载 Diagnostics SOAP 消息处理程序”（第 237 页）。

并非所有应用程序服务器都可使用 Diagnostics SOAP 消息处理程序。并没有提供用于从 SOAP 负载中捕获 SOAP 错误或客户 ID 的自定义插桩。因此，不能在所有应用程序服务器的所有版本中使用此功能。有关 Diagnostics SOAP 消息处理程序支持的最新信息，请参阅 Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

如果需要确定使用特定服务的用户以及他们使用该服务的频率，则按用户 ID 聚合数据将十分有用。用户 ID 对 Business Service Management 也十分有用。Business Service Management 用户可以查看同一应用程序在不同用户使用时的性能，以比较其性能特征。

配置用户 ID 的操作是可选操作。默认情况下，将使用 IP 地址作为 SOAP over HTTP/S Web 服务的用户 ID；使用入站队列名称（或主题名称）作为 SOAP over JMS Web 服务的 SOAP 的用户 ID。

本节包括：

- “用户 ID 的基本配置过程”（第 482 页）
- “关于用户 ID 规则”（第 483 页）
- “Java 代理 的用户 ID 规则语法和示例”（第 485 页）

用户 ID 的基本配置过程

配置用户 ID 的基本过程如下：

1 （可选）。在 **consumer.properties** 文件中指定 ***dump-payload**，将完整的 SOAP 负载输出到 **consumer.log** 文件中。可使用此输出来计划如何创建用于配置用户 ID 的特定规则，以用于 SOAP 有效负载捕获。

在配置用户 ID 前，请先熟悉 SOAP 有效负载的相关数据，这样才能确定如何以最佳方式创建特定的诊断规则，以供 Diagnostics 查找用户 ID 的值。

只有在需要帮助以找到其中包含用户 ID 的元素时，才应使用“转储负载”选项。

在使用时，此选项应该是等号 (=) 右侧的唯一值，例如：
DumpTest;HTTP_WS;TraderService = *dump-payload

重要信息：请不要同时使用同一服务名称来提取值和转储有效负载。

例如，要使用此功能，请输入以下内容：

SoapTest1;HTTP_WS;TraderService = *dump-payload

这将导致输出与 TraderService 匹配的规则的 SOAP 负载。consumer.log 文件内容如下所示：

```
2009-01-15 14:42:13,653 INFO consumer [[ACTIVE] ExecuteThread: '0' for queue:
'weblogic.kernel.Default (self-tuning)'] [PAYLOAD:] <?xml version="1.0" encoding="UTF-8"
standalone="yes"?><soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:trad="http://
www.bea.com/examples/Trader"      xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/">
   <soapenv:Header>
    <CallerA>customerA</CallerA>
   </soapenv:Header>
   <soapenv:Body>
      <trad:buy soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <string xsi:type="xsd:string">hpq</string>
         <intVal xsi:type="xsd:int">11</intVal>
      </trad:buy>
   </soapenv:Body>
</soapenv:Envelope>
```

2 对于要按用户分组其度量数据的每个 Java 代理，请按照 “Java 代理 的用户 ID 规则语法和示例”（第 485 页）中所述的内容更新 **consumer.properties** 文件。

3 如果要跟踪的用户类型超过 5 个，请在 **dispatcher.properties** 文件中更新 **max.tracked.ids.per.probe** 设置。

4 查看 **probe/files/log** 目录中的 **< 探测器名称 >_id.properties** 文件。可能需要修改或完全删除 **< 探测器名称 >_id.properties** 文件，才能与先前步骤中对 **consumer.properties** 所做的更改保持一致。该文件还具有 max.tracked.ids.per.probe (dispatcher.properties) 设置，一旦达到所设置的限制，每个探测器和其他所有用户将被分类为 “其他”。

关于用户 ID 规则

用户 ID 分配由 **consumer.properties** 文件中的用户 ID 规则控制。

每个类别的用户 ID 都有其自身的规则：SOAP 规则、HTTP 标头规则、JMS Web 服务规则和 IP 规则。但并不会按照规则的定义方式应用规则。首先会应用 SOAP 标头规则，然后应用 HTTP 标头规则和 JMS 规则，最后应用 IP 规则。

重要信息：规则中的所有配置项均区分大小写。例如，如果输入 TraderService 的 < 模式名称 >，则 Web 服务必须包含一个大写的 T 和 S，模式才能与其匹配。

并不需要使用所有的规则类型。可能需要使用 SOAP 规则，而不需要 HTTP 规则和 IP 规则。如果与这三种规则都不符合，则会使用原始 IP 地址或队列名称作为用户 ID。

SOAP 规则允许从 SOAP 标头、 SOAP 包络或正文中的 XML 元素获得用户 ID。规则会指定一个正则表达式，用于匹配用户调用的 Web 服务的名称。有关使用正则表达式的帮助，请参阅 “使用正则表达式”（第 882 页）。

如果存在匹配，探测器还会尝试查找规则中指定的文本元素。如果未在 SOAP 标头中找到文本元素，则会跳过此规则，探测器将继续转到已定义的下一规则。

HTTP 头规则允许从 HTTP 请求中的 HTTP 头集合中的头获得用户 ID。

JMS Web 服务规则允许用户 ID 作为 JMS 队列名称 / 主题名称、JMS 消息属性或消息标头 （仅限 JMSReplyTo）。

IP 规则允许从 IP 地址到用户 ID 的映射中获得用户 ID。该规则用于定义要分配到用户 ID 的 IP 地址或地址范围。

Java 代理 的用户 ID 规则语法和示例

可通过在 **consumer.properties** 文件中指定规则，来控制用户 ID 的分配。

重要信息：规则中的所有配置项均区分大小写。例如，如果输入 TraderService 的 < 模式名称 >，则 Web 服务必须包含一个大写的 T 和 S，模式才能与其匹配。

SOAP 标头中的值

要根据 SOAP 标头中的值分配用户 ID，请使用以下格式：

< 规则名称 >;HTTP_WS;< 模式名称 > = soap-header;< 元素值 >

< 规则名称 > 是用于标识规则的字符串。该名称在 consumer.properties 文件中必须是唯一的。

< 模式名称 > 是用于匹配 Web 服务名称的正则表达式，您也可以使用确切的 Web 服务名称。

< 元素值 > 是 SOAP 包络中的元素，其值将用作用户 ID。

例如，下面的规则与服务名称为 TraderService 的 Web 服务相匹配，且使用 CallerA 元素的值作为用户 ID：

```
SoapRule1;HTTP_WS;TraderService = soap-header;CallerA
```

如果 TraderService Web 服务的调用者已定义 CallerA 的值，则会按 CallerA 的不同值对度量数据进行分组。以下 SOAP 标头摘录将映射到 TraderService 的此调用者的用户标识 “Customer2”：

```
SoapTest1;WS<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <env:Header>
      <CallerA>Customer2</CallerA>  <---- The consumer id returned would be
                                                "Customer2"
   </env:Header>
   <env:Body   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <m:sell    xmlns:m="http://www.bea.com/examples/Trader">
         <string     xsi:type="xsd:string">sample string</string>
         <intVal     xsi:type="xsd:int">100</intVal>
      </m:sell>
   </env:Body>
</env:Envelope>
```

默认情况下， Diagnostics 将在第一层元素 （SOAPenv:Header 下的元素）中查找 Caller A。可以配置 Diagnostics，使其在更深层的 XML 元素中查找用户 ID。**consumer.properties** 文件中的 **max.search.level.depth** 用于控制用户 ID 的搜索深度 （默认深度为 1 层）。例如，当 max.search.level.depth = 2 时，将会在以下内容中查找用户 ID：

```
<env:Header>
   <test:id>
      <test:CallerA>consumerA</test:CallerA>
   </test:id>
</env:Header>
```

SOAP 包络中的值

要根据 SOAP 包络中的值分配用户 ID，请使用下列格式：

< 规则名称 >;HTTP_WS;< 模式名称 > = soap-envelope;< 元素值 >

< 规则名称 > 是用于标识规则的字符串。该名称必须在 consumer.properties 文件中是唯一的。

< 模式名称 > 是用于匹配 Web 服务名称的正则表达式，您也可以使用确切的 Web 服务名称。

< 元素值 > 是 SOAP 包络中的元素，其值将用作用户 ID。

SOAP 正文中的值

要根据 SOAP 正文中的值分配用户 ID，请使用下列格式：

< 规则名称 >;HTTP_WS;< 模式名称 > = soap-body;< 元素值 >

< 规则名称 > 是用于标识规则的字符串。该名称必须在 consumer.properties 文件中是唯一的。

< 模式名称 > 是用于匹配 Web 服务名称的正则表达式，您也可以使用确切的 Web 服务名称。

< 元素值 > 是 SOAP 正文中的元素，其值将用作用户 ID。

HTTP 标头中的值

要根据 HTTP 标头中的值分配用户 ID，请使用以下格式：

< 规则名称 >;HTTP_WS;< 模式名称 > = attribute;< 标头值 >

< 规则名称 > 是用于标识规则的字符串。该名称在 consumer.properties 文件中必须是唯一的。

< 模式名称 > 是用于匹配 URI 中内容的正则表达式。

< 标头值 > 是 HTTP 标头，其值将用作用户 ID。

例如，下面的规则与 URI 为 “/webservice/.*” 的 Web 服务匹配，且使用 “User-Agent” 标头的值作为用户 ID：

```
WsRule1;HTTP_WS;/webservice/.* = attribute;User-Agent
```

如果 Web 服务调用者已定义 User-Agent 的值，则会按 User-Agent 的不同值对度量数据进行分组。以下 HTTP 标头摘录将映射到标头中的用户 ID：

```
GET /service/call HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 2000)
Host: ovrntt1
Caller: ovrntt1
Connection: Keep-Alive
```

JMS 队列名称

要根据 JMS 队列 / 主题名称匹配来分配用户 ID，请使用下列格式：

< 规则名称 >;JMS_WS;< 队列名称 >=< 用户 ID 字符串 >

< 规则名称 > 是用于标识规则的字符串。该名称在 consumer.properties 文件中必须是唯一的。

< 队列名称 > 是用于匹配 JMS/ 主题名称的正则表达式。

< 用户 ID 字符串 > 是用作用户 ID 的字符串。

例如，下面的规则与以 queue://sca_soapjms.* 开头的 JMS 队列名称相匹配，且使用字符串 “myJMSConsumer”作为用户 ID：

```
JMSTest3;JMS_WS;queue\://sca_soapjms.*=myJMSConsumer
```

必须在队列或主题之后，使用反斜杠 “\:”来转义 “:”。

匹配过程中使用的优先级由在 consumer.properties 文件中指定的顺序确定。JMS_WS 队列匹配的优先级高于 IP 匹配； JMS_WS 属性匹配的优先级高于 JMS_WS 标头匹配； JMS_WS 标头匹配的优先级高于 JMS_WS 队列名称匹配。

JMS 消息属性

要根据 JMS 队列 / 主题名称匹配来分配用户 ID，且使用 JMS 消息属性值作为用户 ID，请使用以下格式：

< 规则名称 >;JMS_WS;< 队列名称 >=jms-property;< 属性值 >

< 规则名称 > 是用于标识规则的字符串。该名称在 consumer.properties 文件中必须是唯一的。

< 队列名称 > 是用于匹配 JMS/ 主题名称的正则表达式。

< 属性值 > 是 JMS 属性，其值将用作用户 ID。

例如，下面的规则与以 queue://MedRec.* 开头的 JMS 队列名称相匹配，且使用 JMSXDeliveryCount 属性中的值作为用户 ID：

```
JMSTest1;JMS_WS;queue\://MedRec.*=jms-property;JMSXDeliveryCount
```

必须在队列或主题之后，使用反斜杠 “\:”来转义 “:”。

JMS 消息标头

要根据 JMS 队列 / 主题名称和 JMS 消息标头匹配来分配用户 ID，请使用以下格式：

< 规则名称 >;JMS_WS;< 队列名称 >=jms-header;< 标头值 >

< 规则名称 > 是用于标识规则的字符串。该名称在 consumer.properties 文件中必须是唯一的。

< 队列名称 > 是用于匹配 JMS/ 主题名称的正则表达式。

< 标头值 > 必须是 JMSReplyTo。

例如，下面的规则与以 queue://MedRec.* 开头的 JMS 队列名称相匹配，且使用 JMSReplyTo 标头中的值作为用户 ID：

```
JMSTest1;JMS_WS;queue\://MedRec.*=jms-header;JMSReplyTo
```

必须在队列或主题之后，使用反斜杠 “\:” 来转义 “:”。

特定的 IP 地址

要根据 IP 地址来分配用户 ID，请使用以下格式：

< 规则名称 >; IP; <IP 地址 > = < 用户 ID 字符串 >

例如，下面的规则与 IP 地址 123.456.567.8 相匹配，并使用名称 “CustomerA_IP” 作为用户 ID：

```
IPRule1;IP;123.456.567.8 = CustomerA_IP
```

IP 地址的范围

要根据 IP 地址的范围来分配用户 ID，请使用以下格式：

< 规则名称 >; IP; <IP 地址范围 > = < 用户 ID 字符串 >

其中，可以使用整数定义 <IP 地址范围 >，使用 “*” 指定通配符，使用 “-” 指定整数范围。

例如，下面的规则与第一个八位字节为 15 的所有 IP 地址相匹配，并使用名称 “mySuperCluster” 作为用户 ID：

```
IPRule2;IP;15.*.*.* = mySuperCluster
```

下面的规则与第一个八位字节是 15，且第二个八位字节是 200 至 300 之间数字的所有 IP 地址相匹配；它使用名称 “Customer_IP” 作为用户 ID：

```
IPRule3;IP;15.200-300.*.* = Customer_IP
```

配置 SOAP 错误有效负载数据

如果检测到 SOAP 错误，则会将 SOAP 有效负载包括在 SOAP 错误数据中。只有当存在 SOAP 错误时，才会捕获 SOAP 有效负载。

可以在 Diagnostics UI 中查看实例树中的负载信息。支持 JAX-WS 和 JAX-RPC Web 服务。

由于有效负载可能包含如信用卡号等敏感信息，默认情况下禁用对 SOAP 错误的有效负载捕获。

要启用对 SOAP 错误的有效负载捕获，请将 Java 代理的 **dispatcher.properties** 文件中的 **max.soap.payload.bytes** 设置为大于 0 的值（推荐使用 5000）。这是捕获的字节数，因此如果在 UI 中看到的有效负载表明此值太小，则可以增大该数值。默认情况下该值设置为 0 以禁用有效负载捕获。

捕获 SOAP 有效负载需要使用用于 Java 探测器的 Diagnostics SOAP 消息处理程序。对于某些应用程序服务器，已在 Diagnostics 中提供特殊插桩以自动加载 Diagnostics SOAP 消息处理程序。
需要对 WebSphere 5.1 JAX-RPC 和 Oracle 10g JAX-RPC 进行一些手动配置。有关详细信息，请参阅 “加载 Diagnostics SOAP 消息处理程序”（第 237 页）。

Diagnostics SOAP 消息处理程序并非对所有应用程序服务器都可用，也没有可用于从 SOAP 负载中捕获 SOAP 错误或用户 ID 的自定义插桩。因此，不能在所有应用程序服务器的所有版本中使用此功能。有关 Diagnostics SOAP 消息处理程序支持的最新信息，请参阅 Diagnostics Support Matrix，网址为：
http://support.openview.hp.com/sc/support_matrices.jsp。

对于 Java 代理，可以通过修改 **< 探测器安装目录 >\etc\dispatcher.properties** 文件来定义负载大小限制。大于指定大小的负载将被截断。

例如，以下条目可将 SOAP 负载长度由默认的 5000 增加到 10000：

```
max.soap.payload.bytes = 10000
```

将此属性设置为 0 可禁用此功能。

配置 REST 服务

可以配置 REST 样式的 Web 服务，使其在 Diagnostics UI 中显示为常规 Web 服务。有关配置详细信息，请参阅以下文件中的注释：**< 探测器安装目录 >\etc\rest.properties**。

当前仅支持 HTTP （不支持 JMS）。

JMS 临时队列 / 主题的自定义分组

为了在 Diagnostics 中进行报告， JMS 临时队列中的 SOAP 会被分到同一节点。Diagnostics 会将队列 / 主题名称与正则表达式列表进行匹配，以查找临时队列 / 主题名称。根据不同的类型，匹配的队列 / 主题名称将被替换为 **queue:< 探测器 ID>\TEMPORARY** 或 **topic:< 探测器 ID>\TEMPORARY**。

用于此匹配的正则表达式的列表位于 **< 探测器安装目录 >/etc/capture.properties** 文件中。可以在 **grouped.temporary.jms.names** 属性中自定义正则表达式列表。

配置 SQL 查询分析

如果存在大量使用文本的 SQL 查询，则会造成服务器符号表的崩溃。在这类情况下，可以在 Java 代理上配置 **dispatcher.properties** 文件中的 **sql.parsing.mode** 属性。可能的模式设置如下所示：

1 - 仅方法，无 SQL 查询。

2 - SQL 查询的主类别 （选择 / 更新 / 插入 / 删除 /...）。

3 -（默认）每个完整 SQL 查询一个度量，用于将类似语句聚合到一个度量 （忽略文本、关键字大小写 ...）。

4 - 每个完整 SQL 查询一个度量，仅聚合相同语句。

```
sql.parsing.mode = 3
```

dispatcher.properties 文件中的另一个属性可用来限制在临时数据库表中收集的不同 SQL 语句数目，允许您使用 SQL 语句正则表达式替代项折叠表名称。该属性为 **sql.pattern.replace**（有关详细信息，请参阅 **dispatcher.properties** 文件）。

配置为服务器请求显示的应用程序名称

Diagnostics UI 的“服务器请求”详细信息窗格中的“已部署到”值可为大多数应用程序服务器的服务器请求显示应用程序名称。在 Diagnostics 9.0 之前，仅为 WebLogic 应用程序服务器提供了此信息，因此，只有 WebLogic 探测器可以填充服务器请求中的应用程序名称标识符。

为了确保与服务器请求趋势线的向后兼容性，除 WebLogic 服务器请求之外，不会填充其他服务器请求的“应用程序名称”。

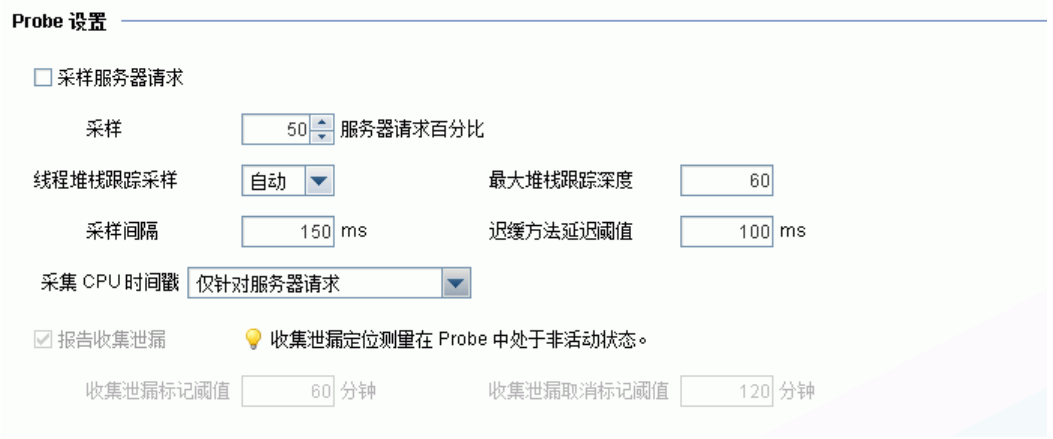
可以使用 **capture.properties** 文件中的 **keep.fragment.data.compatible** 属性对此进行配置。默认情况下， keep.fragment.data.compatible=**true**，这意味着除 WebLogic 服务器请求外，不会填充其他服充务器请求的“应用程序名称”。

如果需要 Diagnostics UI 显示每个服务器请求的 J2EE 应用程序服务器名称（在“服务器请求”视图的详细信息窗格中显示为“已部署到”），请将此属性设置为 **false**。

在 Java Profiler UI 中维护探测器设置

您可以使用 Java Diagnostics Profiler 中的 “配置” 选项卡来维护插桩点以及编辑探测器配置，而无需手动编辑 Java 代理捕获点文件或属性文件。不管配置文件是否已启动，都可以从 Java Diagnostics Profiler 访问 “配置” 选项卡。

Java Diagnostics Profiler 配置选项卡的 “探测器设置” 部分可用于配置探测器设置，以进行堆栈跟踪采样、收集 CPU 时间度量（使用时间戳）以及报告收集泄漏。



在 Java Diagnostics Profiler 的 “配置” 选项卡中单击 “应用更改” 后，您在 “配置” 选项卡的 “探测器设置” 部分所做的更新将应用到捕获点文件和属性文件。

注意：更改将立即生效，而无需重新启动应用程序 （或探测器）。

以下部分描述各 “探测器设置”部分：

- “配置线程堆栈跟踪采样”（第 496 页）
- “控制 CPU 时间戳收集”（第 501 页）
- “启用和配置收集泄漏报告”（第 502 页）

配置线程堆栈跟踪采样

如果启用了异步线程采样，则可以在 “调用配置文件”视图中查看在长时间运行的碎片中执行了哪些方法，即使在此段时间内并没有命中任何插桩方法。请参阅《HP Diagnostics User's Guide》中关于调用配置文件的章节，以查看显示了基于线程采样添加的额外节点的屏幕截图。

可使用多个属性持启用和配置线程堆栈采样。

dynamic.properties 中包含以下属性：

- **enable.stack.trace.sampling** – 启用异步线程堆栈跟踪采样；可能的值包括 false、auto （默认值）和 true。

  当动态属性 enable.stack.trace.sampling 设置为 auto 时，如果探测器在选定（已验证）平台和 JVM 上运行，则会启用堆栈跟踪采样。而对于其他 JVM，则必须将其明确地设置为 true。由于 JVM 可能会生成错误或中止运行，请谨慎操作。请参阅 Diagnostics 自述文件。

- **tardy.method.latency.threshold** – 在对插桩的方法进行堆栈跟踪采样之前，该方法在没有命中任何插桩点的情况下，必须运行的最短时间。此属性的主要目的是将堆栈跟踪收集限制为最受关注的案例，以此控制采样的开销。

- **stack.trace.sampling.rate** – 在执行下一次连续采样前必须等待的时间。

  如果 **stack.trace.sampling.rate** 的值太小，可能会导致频繁地采样并提供大量数据，但同时也会增加开销。

频繁采样所造成的开销主要是会增加服务器请求的延迟时间。探测器的 CPU 总体使用率也会随之上升，但这种影响小于对延迟的影响。对于具有多个 CPU 的系统，进程的 CPU 使用率实际上可能会下降 （并非好现象）。

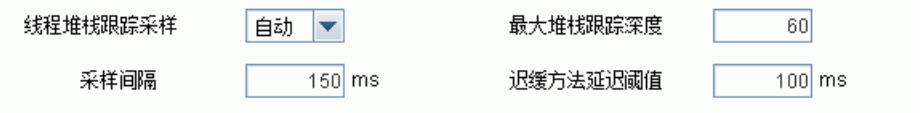
- **stack.trace.depth.max** – 对从 JVM 中获取的堆栈跟踪的深度限制。大多数情况下无需调整该值。

dispatcher.properties 中包含以下属性：

- **enable.stack.trace.aggregation** – 一个布尔属性，允许关联线程以合并在收集的多个连续堆栈跟踪中观察到的节点，除非有证据表明这些节点并不表示单个方法调用。将设置为 ture 时，它将减少所创建的额外调用树节点的数量，但这会造成一个假象，即对额外节点的调用数量是已知的，且这个数量很小。将设置为 false 时，它会为可以看见它的每个方法和堆栈跟踪创建一个节点，从而造成一种假象，即对节点的调用数量是已知的，且这个数量很大。而实际上，堆栈跟踪采样根本无法显示调用数量。
- **aggregated.stack.trace.validity.threshold** – 如果 enable.stack.trace.aggregation 属性设置为 true，则只会报告大于堆栈跟踪的 **aggregated.stack.trace.validity.threshold** 数的调用树节点。此设置可控制噪音消除和内存需求量 （尤其是在服务器端）。

所有这些属性都可以动态更改，因此无需重新启动应用程序。

可以使用 Diagnostics Java Profiler 中的“配置”选项卡远程更改前四个属性（在 dynamic.properties 中）。完成更改后，请记得单击 “应用更改”，以应用通过“配置”选项卡所做的所有更改。



线程采样配置示例

使用案例 1：某特定方法的平均延迟时间约为 170 毫秒，但有时该方法需要 1.4 秒才能完成执行。任何碎片的 “调用配置文件”中的大多数方法都只需 550 毫秒或更短时间就能执行。这是因为相关方法会对其调用对象进行多次调用，但您并不想对它们执行插桩。

因此，您应启用堆栈跟踪采样，以找出执行时间过长的原因。如果要最大程度地减少开销，请将 tardy.method.latency.threshold 设置为 600 毫秒。这可以确保不对大多数方法进行采样，因为大多数方法均可在此时间之前完成执行。但运行时间大于此值的所有方法（包括上面提到的长时间运行的方法）都会被采样。一旦方法的运行时间超过 600 毫秒 （或更长时间），而没有调用任何插桩的方法，则它们会被采样。

如果还将 **stack.trace.sampling.rate** 的值设置为 100 毫秒，则理论上会为每个执行时间为 1.4 秒 ((1400ñ600) / 100) 的方法调用进行 8 次采样。因为您知道方法会对其调用对象进行多次调用，所以还可以将 **aggregated.stack.trace.validity.threshold** 设置为 0。这可以确保即使所收集的每个堆栈跟踪都完全不同，也会全部报告它们。

如果您检查调用配置文件，查看长时间运行的服务器请求实例，则会发现堆栈跟踪采样显示的额外节点。

使用案例 2：准备一个要部署的自定义应用程序，并发现 Diagnostics Agent 附带的默认插桩无法有效工作，原因是许多调用配置文件只包含很少的方法，因此无法提供有关应用程序特定行为的信息。考虑到性能和内存消耗问题，您不愿意为属于该自定义应用程序的所有类和方法添加额外的插桩。

启用堆栈跟踪采样。假设一个典型的无足够调用树详细信息的服务器请求的运行时间约为 2 秒，则您选择 **stack.trace.sampling.rate** 为 200 毫秒。这可以对每个典型的服务器请求最多进行 10 次堆栈跟踪。但您并不想报告所有堆栈跟踪，因为堆栈跟踪中的某些可见方法的执行速度非常快，对服务器请求的总体延迟不会造成实质性的影响。因此，将 **aggregated.stack.trace.validity.threshold** 设置为 2，这样就能确保只报告在 3 个或更多的连续堆栈跟踪中可见的方法，或者是预期延迟时间为 600 毫秒或更长时间的方法。

查看其中包含通过采样获取的额外节点的调用配置文件后，就可以在充分了解信息的情况下，决定为部署中的探测器配置添加额外的插桩点。

关于堆栈跟踪线程采样的疑难解答

问题 1：为何在启用堆栈跟踪采样后，仍然无法在调用配置文件中看见新节点？

答案：详细查看以下清单，以了解是否与您的情况相符：

- 您是否在使用 Java 1.5 或更新版本？堆栈跟踪采样无法在较低版本的 Java 中运行。
- 调用配置文件中的最后一个可见方法是否是出站调用？不会对标记为出站的方法执行采样。（为了确认方法是否被标记为出站，请在 detailReport.txt 文件中查找方法，然后查看其对应的插桩点详细信息中是否包含关键字 “Outbound”。）
- 调用配置文件中的最后一个方法是否标记为 no-layer-recurse？不会对此类方法执行采样。（可以使用上一点中提到的步骤，检查方法是否标记为 no-layer-recurse。）
- 您是否尝试过减小 tardy.method.latency.threshold 或 minimum.method.latency 的值？调用配置文件中的最后一个可见方法所生成的调用已被剪裁，但这些调用禁止采样执行，因为对于调用者而言，不存在 tardy.method.latency.threshold 的非活动期。

- ❑ 您是否尝试过减小 aggregated.stack.trace.validity.threshold 的值，或查看 probe.log 文件中是否存在有关堆栈深度过浅的警告？可能所观察的堆栈跟踪变化太快，导致无法进行报告。
- ❑ 您是否尝试过减小 stack.trace.sampling.rate 的值？方法可能错过了采样机会。
- ❑ 您是否确定调用配置文件中最后一个可见方法的延迟不是因为运行垃圾收集器而导致的？包括堆栈跟踪采样代码在内的 Java 代码不会在垃圾收集期间运行。

问题 2：可以使用的最小 stack.trace.sampling.rate 值是多少？

答案：您可以使用任何正值，但请注意，如果采样频率超过平台的实际运行能力，则平台会拒绝采样。三个决定性的因素为：可用的 sleep() 最小粒度、计时器分辨率和收集一组样本实际所需的时间。

问题 3：可以使用的最大 stack.trace.sampling.rate 值是多少？

答案：该值没有限制。设置很大的值是否有用，完全取决于应用程序的服务器请求延迟时间。要想获得所有结果，请至少为您关注的每个服务器请求计划几个样本。即使这样，仍可能需要您调整其他采样参数。

控制 CPU 时间戳收集

CPU 时间戳可计算方法对 CPU 的独占使用时间。您可以在 Java Diagnostics Profiler 中的 “热点”选项卡中查看此信息。

重要信息：在 VMware 中， CPU 时间度量来自于访客操作系统，并受 VMware 虚拟计时器的影响。请参阅位于 http://www.vmware.com/pdf/vmware_timekeeping.pdf 的有关时间同步的 VMware 白皮书，以及“在 VMware 上运行的探测器的时间同步”（第 468 页）。

默认情况下，已为服务器请求启用 CPU 时间度量收集操作。

您可以在属性文件中配置 CPU 时间度量收集 （请参阅 “配置 CPU 时间度量收集”（第 477 页）），也可以按照下述方法使用 Java Diagnostics Profiler UI 进行收集。

1 在 **Profiler** UI 中，选择“配置”选项卡。无需启动 Profiler 即可更改此探测器配置。

2 在 “配置”屏幕中，从下拉列表中选择 “采集 CPU 时间戳”选项。

CPU 时间戳收集方法	描述
从不	不采集 CPU 时间戳。
仅针对服务器请求	仅为服务器请求收集 CPU 时间戳。
针对服务器请求和 Portlet 方法	对所有服务器请求和 Portal 组件的插桩生命周期方法收集 CUP 时间戳 (layertype=portlet)。
针对服务器请求和所有方法	对所有服务器请求和所有方法收集 CPU 时间戳。

3 完成对 “配置”选项卡的更改后，单击 “应用更改”。

注意：更改将立即生效，而无需重新启动应用程序 （或探测器）。

启用和配置收集泄漏报告

默认情况下，会为探测器启用数据收集和收集泄漏定位。默认情况下，探测器的 **etc/auto_detect.points** 文件中的 **[Collection Leak Pinpointing] keyword = clp** 设置为 true。

注意：如果要使用 Java 代理的回收泄漏定位 (CLP) 功能，则必须使用应用程序服务器的相应模式运行 JRE Instrumenter。

可以使用 Java Profiler “配置”选项卡为收集泄露报告设置以下配置项：

➤ **报告收集泄漏。**可以通过在报告功能对应的 UI 中取消选中复选框来禁用此功能。

- **收集泄漏标志阈值。**收集大小增长的时间段的阈值。如果收集的大小增长周期超过此阈值，则探测器会将此收集标记为内存泄漏。
- **收集泄漏取消标志阈值。**

也可以在探测器的 **dynamic.properties** 文件中设置这些值：**clp.diagnostics.reporting**、 **clp.diagnostics.growth.time** 和 **clp.diagnostics.nongrowth.time**。

为 JUnit 测试生成性能报告

在运行 JUnit 测试时，可以启用并配置 Java 代理，以便为所有单元测试生成性能报告。此功能有利于确定特定测试的性能（延迟 /CPU）是否随时间变化而更改。

单元测试完成后，Java 代理会为每个测试方法（表示为服务器请求）创建 CSV 文件。此 CSV 文件包含在每个 JVM 实例中执行的所有方法的完成列表，通常每个测试类一个。CSV 文件可在电子表格程序中打开，以分析和可视化性能特征（对于选择特定方法而言， Excel 中的 “筛选器”功能非常有用）。

以下为 CSV 文件示例：

```
Date,Server Request,Avg Latency,Count,Min Latency,Max Latency,Cpu
Time,Exceptions
Fri Sep 23 12:55:22 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1068.81,1,1068.81,1068.81,374.403,0
Fri Sep 23 12:55:40 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1064.845,1,1064.845,1064.845,405.60
2,0
Fri Sep 23 12:55:57 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1141.689,1,1141.689,1141.689,358.80
2,0
Fri Sep 23 12:56:27 PDT
2011,UT_SiSXmlDataReader.testDataSample(),1474.81,1,1474.81,1474.81,468.003,0
```

延迟时间单位为毫秒 (ms)。

默认情况下，每次测试执行的数据将附加到 CSV 文件。当测试作为连续集成周期（允许您捕获随时间变化的结果）的一部分运行时，此功能尤其有用。

要使用此功能，请通过指定以下 JVM 参数，在 JUnit 测试执行中启用 Java 代理:

JVM 参数	**描述**
-javaagent:<Java 代理主目录 >/DiagnosticsAgent/lib/probeagent.jar (UNIX) 或 -javaagent:<Java 代理主目录 >\DiagnosticsAgent\lib\probeagent.jar (Windows)	通过指定指向代理 JAR 文件的路径来启用代理。如果使用 JRE 1.4，则可以改为使用 **-Xbootclasspath/p:<Java 代理安装目录 >/DiagnosticsAgent/classes/boot**。
-Ddispatcher.ac.autostart=true	指示代理立即启动配置处理。
-Dcapture.exit_report=dir=perftest:append	指示代理在指定目录中生成性能报告并附加结果。（要覆盖文件，请使用 **override** 替换 **append**。）
-Ddispatcher.minimum.fragment.latency=1ms	仅收集延迟超过 1 毫秒的服务器请求 （如执行 JUnit 测试方法）。

下例显示对 ANT 的集成：

```
<junit dir="${build}" fork="yes" forkmode="perTest" printsummary="yes"
jvm="${env.JAVA_HOME}/bin/java">

    ...

    <jvmarg value="-javaagent:C:/MercuryDiagnostics/JavaAgent/DiagnosticsAgent/lib/
probeagent.jar"/>
    <jvmarg value="-Ddispatcher.ac.autostart=true"/>
    <jvmarg value="-Dcapture.exit_report=dir=<dir_name>:append"/>
    <jvmarg value="-Ddispatcher.minimum.fragment.latency=1ms"/>

    ...

</junit>
```

除了以上设置，还需要激活 JUnit 点（在 **<Java 代理主目录 >/DiagnosticsAgent/etc/auto_detect.points** 中设置 **active=true**）：

```
[JUnit]
class      = junit.framework.TestCase
method     = !test.*
signature  = !.*
deep_mode  = hard
layer      = JUnit
active     = true
```

注意：如果使用 JUnit 4.x 且单元测试类不是 junit.framework.TestCase 的子类，则需要在以上 JUnit 点中更改类定义以匹配单元测试类。

13

了解 .NET 代理配置文件

您可以通过修改以下 .NET 代理配置文件中的元素和属性来控制 .NET 代理的配置：**< 探测器安装目录 >/etc/probe_config.xml**。

本章包括：

了解 .NET 代理配置文件

本节主题介绍组成 .NET 代理配置文件 **<探测器安装目录>/etc/probe_config.xml** 的元素和属性。

可通过描述各个元素的用途、属性、父元素和子元素来定义这些元素。有关特定于 TransactionVision 的其他 .NET 代理配置元素的信息，请参阅《HP TransactionVision Deployment Guide》。

.NET 代理配置元素

<appdomain> 元素

目的

为含有多个应用程序域的进程生成 AppDomain 包含列表。如果没有为进程定义 appdomain 元素，则将包含进程的所有应用程序域。

属性

属性	有效值	默认值	描述
enabled	true false	true	确定是否应当启用 AppDomain。 由 process 元素的 enableallappdomains 属性覆盖。
name	字符串	无	.NET AppDomain 的名称。（限定的 IIS 路径，请参见以下示例。）
website	字符串	无	网站类型的 appdomain 的网站名称（仅供参考）。

元素

出现次数	零或更多次
父元素	process
子元素	bufferpool、 credentials、 diagnosticsserver、 mediator、 id、 ipaddress、 logging、 lwmd、 modes、 points、 profiler、 sample、 trim、 webserver、 symbols、 filter、 topology

示例

```
<appdomain enabled="true" name="1/ROOT/MSPetShop"/>
```

其中， 1/ROOT 为网站 ID， MsPetShop 为虚拟 DirName

```
<appdomain enabled="false" name="1/ROOT" website="Default Web Site">
        <points file="Default Web Site.points"/>
        <id probeid="Default Web Site" />
</appdomain>
```

<authentication> 元素

目的

列出经过验证的用户名和密码。

属性

属性	有效值	默认值	描述
username	字符串	admin	用户名帐户。
password	字符串	admin	必须使用 < **探测器安装目录** >**bin** 目录中的 Passgen 实用程序生成密码。

元素

出现次数	零到多次
父元素	profiler
子元素	无

示例

```
<profiler authenticate="true">
    <authentication username="Test" password="uU8X9zOtl6Twi7TkGAhQ="/>
</profiler>
```

<bufferpool> 元素

目的

配置缓冲池的行为。

属性

属性	有效值	默认值	描述
size	数字	65536	每个缓冲区的大小。
buffers	数字	512	池中的缓冲区数量。
sleep	数字	1000	两次刷新检查之间的毫秒数。
expires	数字	1000	缓冲到期之前的毫秒数。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<bufferpool size="65536" buffers="512" sleep="1000" expires="1000" />
```

<captureexceptions> 元素

用途

启用和控制对异常的堆栈跟踪捕获。

属性

属性	有效值	默认值	描述
enabled	true false	true	启用异常捕获。
max_per_request	数字	4	为一个服务器请求所捕获的最大异常数。
max_stack_size	数字	0（表示没有最大值）	所捕获的异常的最大调用堆栈大小。

元素

出现次数	1
父元素	probeconfig
子元素	include、 exclude

示例

```
<captureexceptions enabled="true" max_per_request="4">
```

<consumeridrules> 元素

用途

这是用于配置用户 ID 规则的根元素。

属性

属性	有效值	默认值	描述
enabled	true false	false	启用用户 ID 规则评估。

元素

出现次数	1
父元素	probeconfig
子元素	httpheaderules、 iprules、 soaprules

示例

```
<consumeridrules enabled="false">
```

<cputime> 元素

目的

控制 **cputime** 设置属性。

属性

属性	有效值	默认值	描述
mode	none、serverrequest、method	serverrequest	

元素

出现次数	1
父元素	probeconfig、 process 或 appdomain
子元素	无

示例

```
<cputime mode="serverrequest"/>
```

<credentials> 元素

目的

提供用于验证与 Diagnostics 服务器的通信的凭据。

属性

属性	有效值	默认值	描述
username	字符串	无	用于 Diagnostics 服务器验证的用户名。
password	字符串	无	用于 Diagnostics 服务器验证的密码。
authenticate	true、 false	true	启用和禁用身份验证。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<credentials username="test" password="diag" authenticate="true"/>
```

<demomode> 元素

目的

该元素用于配置演示模式。通过演示模式，可以更容易地展示 .NET 代理的功能和价值，因为它只要求定义较少的自定义点。打开演示模式后，将显示全部出站调用，而不受任何其他设备影响。

一旦识别出引发出站调用的相关调用后，则应关闭演示模式，并添加“自定义”插桩，确保引发出站调用的调用堆栈明确。

注意：建议在生产环境下关闭此模式。

演示模式主要用于在不插桩生成出站调用的方法时查找出站调用 （webserver、http、remoteing、msmq）。这是一种快速的方法，可找出如何在不插桩特定应用程序方法的前提下，连接应用程序。在生产情况下这种方法可能过于繁杂，但在缺乏上游插桩且不了解出站调用的生成源时，这种方法十分有用。可将其用于包括 ASP.NET 在内的所有种类的应用程序。

属性

属性	有效值	默认值	描述
enabled	true、 false	false	启用或禁用演示模式。

元素

出现次数	零或一次。
父元素	probeconfig
子元素	无

示例

```
<demomode enabled="false"/>
```

<depth> 元素

目的

配置深度裁剪。

属性

属性	有效值	默认值	描述
enabled	true false	true	启用深度裁剪。
depth	数字	25	设置深度裁剪的深度。

元素

出现次数	1
父元素	trim
子元素	无

示例

```
<trim>
    <depth enabled="true" depth="25"/>
</trim>
```

<diagnosticsserver> 元素

目的

包含与用于企业模式的 Diagnostics 服务器相关的连接和设置信息。

属性

属性	有效值	默认值	描述
url	注册器 URL。 http://< 主机 >: < 端口 >	无	连接到注册器的 URL。
delay	数字	2	在注册之前等待的秒数。
keepalive	数字	15	保持连接间隔的秒数。
proxy	代理服务器的 URL	无	注册器连接代理服务器。
proxyuser	代理服务器的用户 ID	无	代理服务器用户帐户。
proxypassword	代理服务器的密码	无	代理服务器用户帐户的密码。
registered_hostname	字符串	无	注册的主机名（用于防火墙遍历的外部名称）。
register_byip	true、 false	false	使用 IP 地址而不使用主机名注册。
timeskewcheckinterval	数字	60	等待从 Diagnostics 服务器获取时间偏差的秒数。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

以下是一个有关 <diagnosticsserver> 元素设置的常规示例。问号 (?) 表示需要替换成合适的值。

```
<diagnosticsserver url="http://localhost:2006/commander" delay="2"
keepalive="15" proxy="?" proxyuser="?" proxypassword="?"
registerhostname="?" register_byip="false"/>
```

有关使用 registered_hostname 属性覆盖默认的探测器主机计算机名称的步骤，请参阅 “覆盖默认探测器主机名”（第 612 页）。

<exceptiontype> 元素

目的

定义异常类型。

属性

属性	有效值	默认值	描述
name	字符串	无	异常的类名称。

元素

出现次数	零到多次
父元素	include、 exclude
子元素	无

示例

```
<exceptiontype name="System.DivideByZeroException"/>
```

<exclude> 元素 （当父元素是 captureexceptions 时）

用途

定义要排除的异常的列表。

属性

无

元素

出现次数	1
父元素	captureexceptions
子元素	exceptiontype

示例

```
<exclude>
    <exceptiontype name="System.DivideByZeroException"/>
</exclude>
```

<exclude> 元素 （当父元素是 lwmd 时）

目的

定义要从 .NET Profile “收集”选项卡和 Diagnostics 用户界面 “收集”视图的 “收集 (按增长率)”和 “收集 (按大小)”表中排除的收集类。

指定的收集类中可能包含用于实现 **ICollection** 的类。请注意，此设置不影响 LWMD 点的测量，只会影响 LWMD 数据的显示以及发送到 Diagnostics 服务器的 LWMD 数据量。

属性

无

元素

出现次数	零到多次
父元素	lwmd
子元素	无

示例

```
<lwmd enabled="true" sample="15s" autobaseline="1h" growth="10" size="10">
    <exclude>System.Collections.ArrayList</exclude>
    <exclude>System.Data.DataView</exclude>
</lwmd>
```

请注意， System.Data.DataView 可实现 System.Collections.ICollection。

<excludeassembly> 元素

用途

排除对程序集的测量。程序集是 .exe 或 .dll 文件。此元素能够从测量中排除敏感的程序集。例如，将产品用于模糊和加密敏感程序集中的代码时，如果测量敏感程序集，则会抛出异常。

将 <excludeassembly name=< 要排除的程序集名称 > 作为子元素添加到 process 元素中。

属性

属性	有效值	默认值	描述
name	字符串	无	要排除的程序集的名称（不含文件扩展名）。

元素

出现次数	零到多次
父元素	process
子元素	无

示例

```
<process enablealldomains="true" name="ASP.NET">
   <logging level="" />
   <points file="ASP.NET.points" />
   <points file="ADO.points" />
   <points file="WCF.points" />

   <excludeassembly name="Acme.Encryption" />

  <appdomain enabled="false" name="TestWebService">
     <points file=" TestWebService .points" />
   </appdomain>
</process>
```

<filter> 元素

目的

筛选出某些会使结果产生偏差或不代表受监控的进程的度量。

属性

属性	有效值	默认值	描述
firstserverrequest	true、 false	false	支持 / 禁止在应用程序启动后第一次运行特定服务器请求 (URL) 时跳过度量收集操作。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<filter firstserverrequest="false"/>
```

<httpclient> 元素

目的

此元素用于配置是否在 HTTP 出站调用标识中包含 URL。默认值为 true，并且应当保持为此值，除非出站 HTTP 调用具有许多不同的 URL。由于会产生大量出站调用 （为每个不同的 URL 创建一个出站调用），因此可能会降低 Diagnostics 服务器的性能。如果不需要 HTTP 出站调用的 URL，也可以将其关闭。这样，HTTP 出站调用的标识将只包含向其发送请求的服务器和端口号。

属性

属性	有效值	默认值	描述
showurl	true、 false	true	支持 / 禁止在客户端使用 HTTP 生成的出站调用标识中包含 URL。 如果有过多不同的 http 客户端调用，则可将其设置为 false，以避免服务器 / 代理端的符号表激增。 对于 REST 服务客户端应用程序，此值应设置为 false。

元素

出现次数	零到一次。
父元素	probeconfig、 process、 appdomain
子元素	无

示例

```
<httpclient showurl="true"/>
```

<gentvhttpeventforwcf> 元素

目的

设置此选项后，会为包含使用 IIS（基于 http）托管的任何绑定的 WCF 服务启用 TransactionVision 事件生成。某些 WCF 服务可能会使用不作为真正的 Web 服务受到支持的自定义或专用绑定，在此类情况下，将不会生成 TransactionVision Web 服务事件，除非启用此选项。

属性

属性	有效值	默认值	描述
enabled	true、 false	false	为具有使用 IIS（基于 http）托管的任何绑定的 WCF 启用 / 禁用 http 事件生成。如果启用，则会提供 TransactionVision Web 服务事件。

元素

出现次数	零到一次。
父元素	probeconfig、 process、 appdomain
子元素	无

示例

```
<gentvhttpeventforwcf enabled="true"/>
```

<httpheaderrule> 元素

目的

定义 HTTP 标头的用户 ID 规则。

属性

属性	有效值	默认值	描述
id	字符串	无	规则的 ID。
rule	字符串	无	用于与用户正向其发送 HTTP 请求的 URL 匹配的正则表达式。
consumeridfield	字符串	无	用作用户 ID 的标头名称。

元素

出现次数	零到多次
父元素	httpheaderrules
子元素	无

示例

```
<httpheaderrule id="httpHeader 1" rule="/Webservice/.*"
consumeridfield="Caller"/>
```

<httpheaderrules> 元素

目的

此元素包含所有 <httpheaderrule> 元素。

属性

无

元素

出现次数	1
父元素	consmeridrule
子元素	httpheaderule

示例

```
<httpheaderrules>
</httpheaderrules>
```

<id> 元素

目的

提供探测器 ID 和探测器组 ID。

属性

属性	有效值	默认值	描述
probeid	字符串，其中包含： 字母、数字、下划线、短划线、句点和在内部定义的 $() 变量值： $(APPDOMAIN)、 $(MACHINENAME)、 $(WEBSITENAME)、 $(PID)	$(APPDOMAIN).NET	由 LoadRunner/ Performance Center 和系统运行状况程序识别的探测器的名称。
probegroup	字符串	默认值	定义由 Diagnostics 服务器识别以报告系统度量和探测器度量的分组。

元素

出现次数	每个父元素 1 次
父元素	probeconfig、 process、 appdomain
子元素	无

示例

默认设置示例。

```
<id probeid="$(APPDOMAIN).NET" probegroup="Default"/>
```

示例

在 LoadRunner 8.1 环境中运行的向“myDiagSever”报告的探测器示例，其中，探测器名称由有效字符、在其中部署应用程序的网站的名称，以及在其中部署应用程序的计算机名称组成。

```
<id probeid="LR_81_$(WEBSITENAME)_$(MACHINENAME).NET"
probegroup="LR_81_myDiagServer"/>
```

<include> 元素 （当父元素是 captureexceptions 时）

目的

定义要包含的异常的列表。

属性

无

元素

出现次数	1
父元素	captureexceptions
子元素	exceptiontype

示例

```
<include>
    <exceptiontype name="System.DivideByZeroException"/>
</include>
```

<include> 元素 （当父元素是 lwmd 时）

目的

定义要包含的收集，以排除其他收集。

属性

无

元素

出现次数	零到多次
父元素	lwmd
子元素	无

示例

```
<include>System.Collections.Hashtable</include>
<include>System.Collections.ArrayList</include>
```

<nstrumentation> 元素

目的

包含 Instrumenter 的日志记录配置。

属性

无。

元素

出现次数	每个父元素 1 次
父元素	probeconfig、 process
子元素	logging

示例

```
<instrumentation>
    <logging level="property lwmd" />
</instrumentation>
```

<iprule> 元素

目的

定义 IP 地址的用户 ID 规则。

属性

属性	有效值	默认值	描述
id	字符串	无	启用用户 ID 规则评估。
rule	字符串	无	定义要分配到用户 ID 的 IP 地址或地址范围。
consumerid	字符串	无	使用的用户 ID （如果与规则匹配）。

元素

出现次数	零到多次
父元素	iprules
子元素	无

示例

```
<iprule id="IpTest1" rule="43.*.1-20.*" consumerid="HP"/>
```

<iprules> 元素

目的

此元素包含所有 <iprule> 元素。

属性

无

元素

出现次数	1
父元素	consumeridrules
子元素	iprule

示例

```
<iprules>
</iprules>
```

<latency> 元素

目的

配置延迟裁剪。

属性

属性	有效值	默认值	描述
enabled	true false	true	启用延迟裁剪。
throttle	true false	true	启用延迟裁剪阻断。
min	数字	2	最小延迟阈值。
max	数字	100	最大延迟阈值。
increment	数字	2	阈值增量。
increment threshold	数字	75	在增加阻断之前的缓冲区使用百分比。
decrement threshold	数字	50	在减小阻断之前的缓冲区使用百分比。

元素

出现次数	1
父元素	trim
子元素	无

示例

```
<trim>
    <latency enabled="true" throttle="true" min="2" max="100" increment="2"
incrementthreshold="75" decrementthreshold="50"/>
</trim>
```

<logdirmgr> 元素

目的

包含日志目录管理器的配置。 Logdirmgr 会监控日志目录以确保它不会无限制地增长，同时还会按照 scaninterval 的指示定期扫描日志。如果日志大小已超过 maxdirsize 所指定的大小，则 logdirmgr 会删除最早的文件，直到日志大小小于 maxdirsize 所指定的大小为止。

重要信息：必须向运行 .NET 进程的帐户 （对于 IIS，即 AppPool 帐户）提供日志文件夹的**删除**权限。默认情况下， NETWORK SERVICE 帐户或 APP Poll 标识帐户 （即默认的 Application Pool 帐户）不具备此权限。

属性

属性	有效值	默认值	描述
enabled	true false	true	
maxdirsize	数字	1024 MB	要用于限制日志目录大小的最大大小。
scaninterval	数字	30m	管理器扫描日志以检查增长率和大小情况的频率。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

```
<logdirmgr enabled="true" maxdirsize="1024 MB" scaninterval="30m"/>
```

<logging> 元素 （当父元素是 instrumentation 时）

目的

设置 .NET 代理测量过程的日志记录级别。

属性

属性	有效值	默认值	描述
level	off assert break severe warning info _____ debug points eh sig chi cil classmap ilasm symbols deepmode load all checksum property remoting lwmd http	"" 相当于 “info”	日志记录的级别。
threadids	true false	true	是否在日志中包括线程 ID。

注意：通常，不应使用“info”下面的有效值。这些 Diagnostic 设置会生成非常大的日志文件。

元素

出现次数	零到多次
父元素	instrumentation
子元素	无

示例

```
<instrumentation>
    <logging level="warning" />
</instrumentation>
```

<logging> 元素（当父元素是 appdomain、probeconfig 或 process 时）

目的

设置 .NET 代理进程的日志记录级别以监控和报告应用程序性能。

属性

属性	有效值	默认值	描述
level	off severe warning info _____ debug events property webserver http symbols probemetrics registrar threadpool authentication bufferpool rum bacforsoa vmware exceptions| tvdebug	"" 相当于 “info”	
max	数字	10	探测器日志文件的最大大小。日志达到此大小后，将不再记录任何日志。

注意：通常，不应使用“info”下面的有效值。这些 Diagnostic 设置会生成非常大的日志文件。

元素

出现次数	
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<logging max="10" level="INFO"/>
```

<lwmd> 元素

目的

配置轻型内存诊断 (LWMD) 功能。

属性

属性	有效值	默认值	描述
enabled	true false	false	为 lwmd 捕获启用采样。
sample	字符串	1m	采样间隔 （h- 小时 /m- 分钟 /s- 秒）。
autobaseline	字符串	1h	自动基线间隔。
manualbaseline	字符串	无	手动基线时间。
growth	数字	15	增长率跟踪的收集数。
size	数字	15	大小跟踪的收集数。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	exclude、 include

示例

```
<lwmd enabled="false" sample="1m" autobaseline="1h" manualbaseline= "?"
growth="15" size="15"/>
```

<mediator> 元素

目的

指定在企业模式中时，向其发送事件的 Mediator 模式 Diagnostics 服务器。

属性

属性	有效值	默认值	描述
host	主机名	无	Mediator 的名称。
port	数字	2612	Mediator 端口。
ssl	true/false	false	当 Diagnostics 服务器 URL 以 http 开头时，默认值为 false。当 Diagnostics URL 以 https 开头时，默认值为 true。
metrichost	字符串		要向其发送度量数据的主机。
metricport	数字	2006	探测器向其发送探测器度量数据（如堆使用情况和可用性）的端口。
block	true/false	false	在建立 Mediator 连接之前阻止通信。
ipaddress			连接到事件服务器时使用的本地 IP 地址。

属性	有效值	默认值	描述
localportstart	数字	4000	用于到 Diagnostics Mediator 服务器的 tcp 事件通道连接的端口范围起始值。应仅在指定了 ipaddress 时使用。
localportend	数字	5000	用于到 Diagnostics Mediator 服务器的 tcp 事件通道连接的端口范围结束值。应仅在指定了 ipaddress 时使用。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<mediator host="localhost" port="2612" ssl="false" metricport="2006"
block="false" ipaddress="16.255.18.99" localportstart="4000"
localportend="5000"/>
```

<metrics> 元素

目的

此元素包含所有 <metric> 元素。

属性

无

元素

出现次数	每个父元素 1 次
父元素	appdomain、 process
子元素	metric

示例

```
<metrics>
  <metric name="% Time in GC" group="Memory" units="percent"
category=".NET CLR Memory" counter="% Time in GC"/>
</metrics>
```

<metric> 元素

目的

指定希望 Diagnostics .NET 从 perfmon 中收集的其他探测器度量值。有关更多信息，请参阅 “收集其他探测器度量或修改探测器度量”（第 622 页）。

属性

属性	有效值	默认值	描述
name	字符串		要在 Diagnostics UI 中显示的度量的名称。
group	字符串		要在 Diagnostics UI 中显示的度量的组 （类别）。
units	microseconds、milliseconds、seconds、minutes、 hours、days、 bytes、kilobytes、megabytes、gigabytes、count、 percent、fraction_percent、load、 status		Perfmon 度量的度量单位。
category	字符串		在 perfmon 中指定的性能计数器类别。
counter	字符串		在 perfmon 中指定的性能计数器

注意：计数器的实例自动分配为该计数器的进程实例或 ASP.NET 应用程序计数器的应用程序域实例。如果计数器没有进程或未收集到应用程序域实例，则应当定义系统度量。

元素

出现次数	每个父元素 1 次或多次
父元素	metrics
子元素	无

示例

```
<metrics>
  <metric name="% Time in GC" group="Memory" units="percent"
category=".NET CLR Memory" counter="% Time in GC"/>
</metrics>
```

<modes> 元素

目的

指定 .NET 代理应当在哪个（些）产品模式下运行。有关使用不同模式的详细信息，请参阅 “控制可与代理一起使用的 HP 软件产品”（第 593 页）。

<modes> 元素还可用于针对 HP Diagnostics 许可证容量确定使用情况（有关许可证的详细信息，请参见 “基于当前连接的探测器许可证信息”（第 83 页））。

<modes> 元素的值将在您安装代理时进行初始设置。

可在不同模式中设置 .NET 代理以执行以下操作：

- 从开发到预生产测试，再到生产，监控应用程序。
- 配合使用其他 HP 软件产品。
- 用作不向服务器或其他 HP 软件产品报告数据的 Diagnostics Java Profiler。

属性

属性	有效值	默认值	描述
enterprise	true false	取决于在安装过程中选择的模式。 ➤ 如果 pro 值为 false，则为 true ➤ 如果 pro 值为 true，则为 false	设置在企业模式中运行的代理（探测器使用 Diagnostics 服务器）。 企业模式类似于 *ad*、*am* 和 *pro* 模式的组合。它将捕获 LoadRunner 运行的数据，以及 LoadRunner 运行以外的数据。 .NET 代理默认使用企业模式（如果没有指定 AD 或 AM 模式）。在企业模式下，将针对 AM 许可证容量计算代理。
ent	true false	取决于在安装过程中选择的模式。 如果 pro 值为 false，则为 true 如果 pro 值为 true，则为 false	这是 enterprise 属性的缩写。

属性	有效值	默认值	描述
ad	true false	false	*ad* 模式会取代所有其他模式。如果设置了 *ad* 模式和任何其他模式，则会将模式设置为 *ad*。 在 *ad* 模式中， .NET 代理只会从 LoadRunner 中捕获运行，并将结果放置在用于该运行的特定数据库中 （例如 Default21）。 当探测器在 LoadRunner 或 Performance Center 测试运行中运行时，只会针对 AD 许可证容量计算 AD 模式下的代理。当不在测试运行中运行时，代理将不针对许可证容量计数。 例如，如果在 LoadRunner/ Performanace Center AD 模式下安装了 20 个探测器，但一次运行中只有五个探测器，则只会针对 AD 许可证容量计算 5 个探测器。

属性	有效值	默认值	描述
am	true false	➤ false	*am* 模式会取代除 *ad* 外的所有其他模式。在 *am* 模式中，.NET 代理将忽略运行。如果 LoadRunner 在执行应用程序，则可以在正常 Diagnostics 数据库中看见数据。 将始终针对 AM 许可证容量计算 AM 模式下的代理。

属性	有效值	默认值	描述
pro	true false	取决于在安装过程中选择的模式。 ➤ 如果 enterprise 值为 false，则为 true ➤ 如果 enterprise 值为 true，则为 false	设置在 Profiler 模式中运行的代理。 此模式会将数据发送到 profiler，且可以与其他模式同时使用。将不针对许可证容量计算 Pro 模式下的代理。
tv	true false	false	启用 TransactionVision 事件捕获功能。有关如何设置传输选项及其他 TV 选项的细信息，请参阅 “关于针对 TransactionVision 的 .NET 代理配置”（第 279 页）。 此模式会将事件发送到 TransactionVision，此模式可与其他模式结合运行，但在 TV 模式下，将不针对 Diagnostics 许可证容量计数代理。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	无

示例

```
<modes enterprise="false" ad="false" am="false" pro="true"/>
```

<points> 元素

目的

指定用于测量的捕获点文件。

属性

属性	有效值	默认值	描述
file	字符串	无	测量捕获点文件的名称。

元素

出现次数	零或更多次
父元素	appdomain、 process
子元素	无

示例

```
<points file="ASP.NET.points"/>
```

<probeconfig> 元素

目的

为 .NET 代理配置提供一个包含根的元素。

属性

无。

元素

出现次数	1
父元素	无
子元素	appdomain、 bufferpool、 captureexceptions、 consumeridrules、 credentials、 diagnosticsserver、 eventserverhost、id、instrumentation、ipaddress、 logging、 lwmd、 mediator、 modes、 points、 process、 profiler、 rum、 sample、 soappayload、 trim、 webserver、 topology、 vmware、 xvm

示例

```
<probeconfig>
</probeconfig>
```

<process> 元素

目的

提供待监控进程的包含筛选器列表。

如果没有定义进程元素，则不会监控任何进程。

属性

属性	有效值	默认值	描述
enablealldomains	true false	true	如果设置为 true，则进程中所有 appdomain 上的 enable 属性将被覆盖，以便启用所有属性。
name	字符串	无	这些设置所适用的 .NET 进程的名称。

以下是 <process> 元素的 enablealldomains 属性的规则：

- enablealldomains = false：如果在 <appdomain> 列表中没有域，则不启用任何域。
- enablealldomains = false：如果在 <appdomain> 列表中有域且 <appdomain> 的 “enable” 属性设置为 true 或未定义该属性，则应启用域。
- enablealldomains = true：如果在 <appdomain> 列表中有域，则应当仅启用列表中的域，而不考虑其 “enable” 属性为何。
- enablealldomains = true：如果在 <appdomain> 列表中没有域，则应启用所有域。
- 未定义 enablealldomains 属性：与 enablealldomains = true 情况相同。

元素

出现次数	零或更多次
父元素	probeconfig
子元素	appdomain、 bufferpool、 credentials、 diagnosticsserver、 mediator、 id、 instrumentation、 ipaddress、 logging、 lwmd、 modes、 points、 profiler、 sample、 trim、 webserver、 filter、 symbols、 topology

示例

```
<process enablealldomains="true" name="ASP.NET">
```

<profiler> 元素

目的

包含有关 Profiler 功能的设置。

属性

属性	有效值	默认值	描述
authenticate	true、 false	无	启用 / 禁用传入 Profiler 连接请求的身份验证。
register	true、 false	false	通知探测器进行注册，即使它处于“仅 Profiler”模式也是如此。
samples	数字	60	通知 Profiler 要为 lwmd/ 堆趋势保留的样本数量。
best	数字	1	要保留的最快实例树的数量。
worst	数字	3	要保留的最慢实例树的数量。
inactivitytimeout	字符串	10m	用户停止与 Profiler 交互之后 Profiler 继续运行的时长。
disableremoteaccess	true、 false	false	禁用对 Profiler 的远程访问，从而不提供用户名 / 密码，但仍然能够通过 telnet/RemoteDeskTop 访问计算机并在本地运行 Profiler。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	authentication

示例

```
<profiler authenticate="true" register="false" samples="60" best="1" worst="3"
inactivitytimeout="10m">
    <authentication username="admin" password="admin"/>
</profiler>
```

<rum> 元素

目的

控制实时用户监控的设置。

属性

属性	有效值	默认值	描述
enable	true false	true	启用或禁用 RUM 集成功能。
responseheader	字符串	X-HP-CAM-COLOR	其值包含 Diagnostics 到 RUM 集成信息的 http 标头的名称。
encryptedkey	字符串		必须使用 <探测器安装目录>\bin 目录中的 Passgen 实用程序生成加密密钥。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

```
<rum enabled="true" responseheader="X-HP-CAM-COLOR"
encryptedkey="OBF:3pe941vx43903wre40303xxz3q6r42ob43n93wre3io03xjs4
0h940pc3wir3q233jur3zir3yi03zir3vc03wre3xpi3r8o3olr44na3zor3v6m3vc03zir4
4u03ohb3rdi3xjs3wx03v6m3zor3yc63zor3jqz3q6r3wd740vi40b53xpi3ike3wx04
3gp42ur3q233y3r3zwy3wx0432i42293p9p"/>
```

要创建加密密钥，请按如下方式使用 PassGen 实用程序：

```
cd < 安装目录 >/bin
PassGen /system encryptionKey
```

其中， **encryptionKey** 是由字母数字字符组成的字符串，最大长度为 128 个字符。 encryptedkey 将在 stdout 上显示。。

Passgen 示例：

```
PassGen /system TheLazyFoxJumpedHigh
```

返回：

```
OBF:3q6r3xxz3y3r3xjs3wx03yc63n0r3lbr3vc03wd745893wre44u0413j3kn93zw
y40vi432i44fr3m453m894493439040pc40303kjd419r44na3wx0451h3wir3v6m3
lfr3mwj3yi03wre3xpi3xxz3y3r3q23
```

<sample> 元素

目的

设置采样类型和速率。

属性

属性	有效值	默认值	描述
method	百分比、计数、周期	百分比	设置采样方法： ➤ 如果为百分比，则必须在 0 - 100 之间 ➤ 如果为计数，则必须 >1 ➤ 如果为周期，则必须是标准 Diagnostics 时间字符串（3m 相当于 3 分钟， 4s 相当于 4 秒，以此类推）
rate	数字	0	设置百分比类型的采样速率。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process、 ws
子元素	无

示例

<xvm><ws><sample method="percent" rate="50"/></ ws ></xvm>
采样为随机的百分率。

<xvm>< ws ><sample method="count" rate="50"/></ ws ></xvm>
采样为每速率计数一次。

<xvm>< ws ><sample method="period" rate="60000"/></ ws ></xvm>

<soapcapture> 元素

目的

配置是否捕获 SOAP 请求和响应。

属性

属性	有效值	默认值	描述
enabled	true false	true	启用或禁用 SOAP 请求和响应捕获。如果禁用，则会对以下各项产生影响： ➤ 针对 SOAP 错误的 SOAP 请求捕获 ➤ 针对 TV 模式的 SOAP 请求和响应捕获 ➤ 通过 SOAP 规则分配的 ConsumerID。
maxsize	数字	0	此为可选属性，它指定所捕获的 SOAP 请求或响应的最大大小，以字符为单位。 0 表示无限制。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

```
<soapcapture enabled="true" maxsize="0" />
```

<soappayload> 元素

目的

此元素已过时，并由 <soaprequestforsoapfault> 元素所替代。

配置 “SOAP 错误的 SOAP 有效载荷捕获”功能，该功能可提供与 SOAP 错误相关的 SOAP 有效荷载。此处的 SOAP 有效荷载将定义为整个 SOAP 信封。

属性

属性	有效值	默认值	描述
enabled	true false	true	启用或禁用 SOAP 有效荷载捕获功能。
maxsize	数字	5000	此为可选属性，它指定任何有效负载捕获的最大大小，以字符为单位。如果不存在，则使用默认值。如果存在，但在设置过程中出错，则也使用默认值。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

```
<soappayload enabled="true" maxsize="5000" />
```

<soaprequestforsoapfault> 元素

目的

配置针对 SOAP 错误的 SOAP 请求捕获（包括有效负载）。有效负载可能包含如信用卡号等敏感信息；因此，默认情况下会禁用此元素。

注意：如果禁用了 <soapcapture> 元素，则它将覆盖 <soaprequestforsoapfault> 设置。请参阅 <soapcapture> 元素的文档。

属性

属性	有效值	默认值	描述
enabled	true false	false	启用或禁用“SOAP 错误的 SOAP 请求捕获”功能。默认情况下为禁用状态。
maxsize	数字	5000	此为可选属性，它指定 SOAP 请求捕获的最大大小，以字符为单位。如果不存在，则使用默认值。如果存在，但在设置过程中出错，则也使用默认值。

元素

出现次数	每个父元素 1 次
父元素	probeconfig
子元素	无

示例

```
<soaprequestforsoapfault enabled="true" maxsize="5000" />
```

<soaprule> 元素

目的

定义 SOAP 标头的用户 ID 规则。

属性

属性	有效值	默认值	描述
id	字符串	无	规则的 ID。
rule	字符串	无	用于匹配用户所调用的 Web 服务名称的正则表达式。
consumeridfield	字符串	无	SOAP 标头中的元素，用于获取用作用户 ID 的值。

元素

出现次数	零到多次
父元素	soaprules
子元素	无

示例

```
<soaprule id="SOAP1" rule="TestService2" consumeridfield="Caller"/>
```

<soaprules> 元素

目的

此元素包含所有 <soaprule> 元素。

属性

无。

元素

出现次数	1
父元素	consumeridrules
子元素	soaprules

示例

```
<soaprules>
</soaprules>
```

<sqlparsing> 元素

目的

此元素用于指示解析 SQL 查询的模式。如果存在大量使用文本的 SQL 查询，则会造成服务器符号表的崩溃，因此默认情况下，设置为模式 3 以避免此类问题。

属性

属性	有效值	默认值	描述
mode	1, 2, 3, 4	3	模式，指示解析 SQL 查询的方式。 1 - 仅限方法，无 SQL 查询 2 - SQL 查询的主类别 （选择 / 更新 / 插入 / 删除 /...） 3 -（默认）每个完整 SQL 查询一个度量，用于将类似语句聚合到一个度量（忽略文本、关键字大小写 ...） 4 - 每个完整 SQL 查询一个度量，仅聚合相同语句
keywordsfile	字符串	无	可选，允许您指定包含关键字的文件，这些关键字是您希望代理在 SQL 语句内查找，并在 Diagnostics 存储或显示时以大写突出显示。这有助于确保类似的查询将识别为相同的查询，而不受大小写影响。

元素

出现次数	1
父元素	probeconfig
子元素	

示例

```
<sqlparsing mode="4" keywordsfile="C:\myfolder\mykeyword.txt"/>
```

<symbols> 元素

目的

限制可捕获的唯一 URI 和 SQL 字符串的数量，以控制所消耗的内存大小。

属性

属性	有效值	默认值	描述
maxuri	数字	1000	设置可捕获的唯一 URI 的数量上限。
maxuriname	字符串	超出限制的唯一 URI 的最大数量	
maxsql	数字	1000	设置可捕获的唯一 URI 的数量上限。
maxsqlname	字符串	超出限制的唯一 SQL 的最大数量	

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	urireplacepattern

示例

```
<symbols maxuri="1000" maxuriname="Maximum number of unique URIs
exceeded" maxsql="1000" maxsqlname="Maximum number of unique SQLs
exceeded"/>
```

<timeskew> 元素

目的

用于配置 HP TransactionVision。计算时间服务器和运行 .NET 代理的主机之间的时间差异。可以配置检查时间服务器的频率。

属性

属性	有效值	默认值	描述
historysize	数字	24	（启动时读取）时间偏差样本数，以存储和进行比较，从而获得最佳样本。
checkinterval	数字	300,000 毫秒	（动态）为计算偏差时间而检查时间服务器前等待的时间，单位是毫秒。
latencythreshold	数字	100 毫秒	（动态）来自时间服务器的回复可以取为有效时间偏差值的最大时间，单位是毫秒。
retrythreshold	数字	8	（动态）请求到时间服务器失败时尝试的次数。

元素

出现次数	1 （一）
父元素	tv
子元素	无

示例

```
<timeskew historysize="24" checkinterval="300000" latencythreshold="100"
retrythreshold="8"/>
```

<topology> 元素

目的

控制是否收集拓扑信息并将其发送到 Diagnostics 服务器。

属性

属性	有效值	默认值	描述
enable	true false	true	启用采集拓扑信息并将其传递到 Diagnostics 服务器。

元素

出现次数	1
父元素	<probeconfig>、 <process> 或 <appdomain>
子元素	无

示例

```
<topology enable="true">
```

<transport> 元素

目的

配置 TransactionVision 使用的事件通道。

属性

属性	有效值	默认值	描述
type	mqseries sonicmq	sonicmq	代理使用的事件传输提供程序。
connectionString	见下文。		事件传输提供程序的连接信息。

type=sonicmq 时的 conectionString 语法

```
broker = <broker>; port = <port>; user = <user>; password =<password>;
configurationQueue = <configurationQueue>
```

其中：	为：
broker	其上运行 Sonic 代理的主机的名称。通常是分析器主机名。
port	代理通信时所使用的端口。默认为 21111。
user	用户 ID （SonicMQ 安装可能需要此 ID 进行连接）。默认情况下无需用户名。
password	密码 （SonicMQ 安装可能需要此密码进行连接） 。因通过 PassGen 实用程序创建，形式模糊。默认情况下无需密码。有关 **PassGen** 的详细信息，请参阅 《BSM 应用程序管理用户指南》中的 “管理实用程序”。
configurationQueue	含有 .NET TransactionVision Agent 配置消息的队列的名称。

type=mqseries 时的 conectionString 语法

host= < 主机 >; queuemanager=< 队列管理器 >; port= < 端口 >; channel=, 通道 >
configurationQueue = < 配置队列 >

其中：	为：
host	含有 TransactionVision 配置队列的主机。
queuemanager	队列管理器的名称。
port	队列管理器通信时所使用的 MQSeries 端口。
channel	将用于通信的 MQSeries 通道。
configurationQueue	含有 .NET TransactionVision Agent 配置消息的队列的名称。

元素

出现次数	1 （一）
父元素	tv
子元素	无

示例

对于 SonicMQ：

```
<transport type="sonicmq" connectionstring="broker=brokerHost;
    port=21111; user=; password=;
    configurationqueue=TVISION.CONFIGURATION.QUEUE"/>
```

对于 MQ 系列：

```
<transport type="mqseries" connectionstring="host=mqHost;
    queuemanager=; port=1414; channel=TRADING.CHL;
    configurationqueue=TVISION.CONFIGURATION.QUEUE"/>
```

<trim> 元素

目的

配置裁剪功能，以减少探测器和 Diagnostics 服务器之间传输的数据量。

Profiler 用户界面将忽略所有已配置的裁剪设置（如深度裁剪和延迟裁剪），因为 Profiler 不需要向 Diagnostics 服务器发送任何数据。

属性

无。

元素

出现次数	每个父元素 1 次
父元素	appdomain、 probeconfig、 process
子元素	depth、 latency

示例

```
<trim>
</trim>
```

<tv> 元素

目的

配置 .NET 代理以配合 TransactionVision 使用。

属性

属性	有效值	默认值	描述
eventthreads	数字	3	（启动时读取）代理产生的向分析器发送事件的线程数。
eventthreadsleep	数字	100	（动态）发送消息（事件包）后事件线程休眠的时间，单位是毫秒。
eventmemorythreshold	数字	25,000,000	（动态）内部缓冲区 (Q) 消耗的内存，之后代理将在应用程序线程上尝试和发送消息。
configthreadsleep	数字	10,000	（动态）浏览配置队列后事件线程休眠的时间，单位是毫秒。

元素

出现次数	1 （一）
父元素	ProbeConfig
子元素	transport、 timeskew

示例

```
<tv eventthreads="3" eventthreadsleep="80"
eventmemorythreshold="25000000" configthreadsleep="10000" >
    <timeskew historysize="24" checkinterval="300000" latencythreshold="100"
        retrythreshold="8"/>
        <transport type="sonicmq"
        connectionstring="broker=myhost.mydomain.com;
        port=21111; user=; password=;
        configurationqueue=TVISION.CONFIGURATION.QUEUE"/>
</tv>
```

<urireplacepattern> 元素

目的

用于通过用单个可聚合众多服务器请求的简化服务器请求 URI 替换众多服务器请求来降低服务器请求的数量。使用正规表达式模式匹配。请参阅 “配置 URI 截断和映射”（第 603 页）。

属性

属性	有效值	默认值	描述
enabled	true false	false	启用 URI 模式替换。
pattern value	s/string/string/	如果启用，则会为您定义两种默认模式。	模式值的语法为 s/search_pattern/replace_pattern/。 如果在模式中使用了 /，则应该使用字符 # （而不是 /）作为分隔符。 模式适用于所有服务器请求，且将按照在 probe_config.xml 内指定的顺序应用模式。

元素

出现次数	1
父元素	probeconfig、 symbols
子元素	无

示例

```
<symbols maxuri="" maxsql="">
        <urireplacepattern enabled="true">
         <pattern value="s/TestService1/CommonService/"/>

        <pattern value="s/TestService2/CommonService/"/>
   </urireplacepattern>
 </symbols>
```

<vmware> 元素

目的

在 VMware 环境中运行时，能够调整时间戳，以使其更加准确。

属性

属性	有效值	默认值	描述
attempttime stampadjustments	true false	false	在 VMware 环境中启用时间戳调整。
useworkaround	true false	false	当 attempttimestampadjustments 属性设置为 true，且在 VMware 客户机上运行 .NET 代理，则如果遇到负延迟问题，请将此属性设置为 true。将此属性设置为 true 后，.NET 代理将使用备用调用获取 VMware 主机时间戳以解决负延迟问题。
disableperfcounters	true false	false	如果 .NET 代理造成 IIS 工作程序进程在 VMWare 环境内崩溃，则将此选项设置为 True。这是一种在特定 VMWare 环境内遇到与访问平台 Perfmon 计数器相关的 Microsoft-VMWare 环境问题时的解决方法。

元素

出现次数	1
父元素	probeconfig
子元素	无

示例

```
<vmware attempttimestampadjustments="false"/>
```

<webserver> 元素

目的

指定本地 Web 服务器属性，以便与探测器通信。

属性

属性	有效值	默认值	描述
start	数字	35000	Web 服务器的起始端口。
end	数字	35100	Web 服务器的结束端口。
ipaddress	IP 地址		运行 Web 服务器的本地 IP 地址。

示例

```
<webserver start="35000" end="35100" ipaddress="16.255.18.99"/>
```

<ws> 元素

目的

控制 Web 服务关联采样。

属性

无。

元素

出现次数	1
父元素	<xvm>
子元素	<sample>

示例

```
<xvm><ws><sample method="percent" rate="50"/></ ws ></xvm>
```

<xvm> 元素

目的

控制跨 VM 设置。

属性

无。

元素

出现次数	1
父元素	probeconfig、 process 或 appdomain
子元素	<ws>

示例

```
<xvm></xvm>
```

14

高级 .NET 代理配置

本章提供有关 .NET 代理的高级配置说明。应由深入了解该产品的经验丰富的用户执行高级配置。请谨慎修改任何 Diagnostics 组件的属性。

本章包括：

➤ 配置 SOAP 错误数据 （第 621 页）

➤ 收集其他探测器度量或修改探测器度量 （第 622 页）

在 VMware 上运行的 .NET 代理的时间同步

对于在 VMware 主机中运行的 .NET 代理，需要满足其他一些时间同步要求。对于在 VMware 宾客系统中运行的代理，VMware 宾客系统和基础 VMware 主机之间的时间必须同步。如果未正确同步时间，则 Diagnostics UI 可能不会准确显示在 VMware 宾客系统中运行的探测器的度量，或者不显示任何度量。

应当根据 VMware 产品白皮书 (http://www.vmware.com/pdf/vmware_timekeeping.pdf) 中有关时间同步的“Synchronizing Hosts and Virtual Machines with Real Time”部分中的建议来同步时间。总之，必须在每个装有 Diagnostics 探测器的 VMware 宾客操作系统中安装 VMware 工具，并且必须在 VMWare 工具中打开时间同步选项。请注意，只有当宾客操作系统时间最初被设置为早于 VMware 主机时间时，VMware 工具中的此选项才可用。有关如何安装 VMware 工具的说明，请参阅 VMware ESX Server 的“Basic System Administration”文档。此外，如果要使用任何非 VMware 时间同步软件 （如网络时间协议），则应当在 VMware ESX Server 服务控制台中运行该软件。

自定义 ASP.NET 应用程序的插桩

安装 .NET 代理时，会创建 **ASP.NET.points** 文件，其中包含将由代理应用到受监控服务器上的所有 ASP.NET 处理的标准插桩。

对于已通过特定于应用程序的类和方法插桩的业务逻辑，必须创建特定于应用程序的插桩点，才能捕获该业务逻辑的性能度量。特定于应用程序的插桩点必须存储在自定义捕获点文件中，该文件可以与使用 **< 探测器安装目录 >/etc/probe_config.xml** 文件中的属性的应用程序关联。如果在安装期间或重新扫描 IIS 期间自动检测到应用程序，则会同时为应用程序自动创建自定义捕获点文件。

注意：如果不知道要监控的应用程序中的类和方法，则可以使用随 .NET 代理一起安装的 Reflector 工具来分析应用程序中的 .dll 文件，并搜寻类和方法。有关如何使用 Reflector 的说明，请参阅 “搜寻应用程序中的类和方法”（第 590 页）。

如果需要使 .NET 代理了解您要将自定义捕获点文件中的插桩点应用到应用程序，必须更新 **probe_config.xml** 文件中 **appdomain** 元素的 **points** 属性。

要将自定义捕获点文件与应用程序关联，请执行以下操作：

1 创建其中包含应用程序特定类的插桩的捕获点文件。要创建捕获点文件，请复制 **< 探测器安装目录 >/etc** 目录中的现有捕获点文件。

注意：如果在安装期间或重新扫描 IIS 期间自动检测到应用程序，则该应用程序已经具有捕获点文件，该捕获点文件的部分或全部点文件条目已经被注释掉。

---

2 通过添加插桩点来自定义捕获点文件，以便代理能够捕获应用程序的自定义业务逻辑。

以下示例演示了如何修改捕获点文件以便代理能够捕获 IBuySpy 自定义代码：

```
[IBuySpy Callee]
class = !IBuySpy.*
method = !.*
signature =
scope =
ignoreScope =
layer = Custom.IBuySpy
```

有关插桩的详细信息，请参阅第 10 章 “.NET 应用程序的自定义插桩”。

3 在 **probe_config.xml** 文件中更新 .NET 代理探测器的配置，以确保修改后的捕获点文件能被正确引用。

在 ASP.NET **<process>** 标记中，添加应用程序的 **<appdomain>** 标记。在 **file** 属性和 **enabled** 属性中包含 **<points>** 标记。有关更多详细信息，请参见“不同 IIS 路径下具有相同名称的虚拟目录 (AppDomain)”（第 588 页）。

```
<appdomain name="1/ROOT/your_app_name" website="Default Web Site"
enabled="true">
      <points file="DefaultWebsite-your_app.capture points"/>
</appdomain>
```

以下示例演示了此步骤。这里，已经为 MSPetsShop 应用程序创建了一个自定义捕获点文件，并将该文件命名为 **MSPetShop.points**。同时，已经将应用程序的 **<appdomain>** 标记和捕获点文件添加到 **probe_config.xml** 文件中的 ASP.NET **<process>** 标记中。请注意， IIS 路径包含在 appdomain 标记中。

```
<?xml version="1.0" encoding="utf-8"?>

<probeconfig>
 <id probeid="" probegroup="Umatilla"/>

 <credentials username="" password=""/>
 <profiler authenticate=""><authentication username="" password=""/></profiler>

 <diagnosticsserver url="http://issaquah:2006"/>
 <mediator host="issaquah" port="2612"/>
 <webserver start="35000" end="35100"/>
 <modes am="true"/>

 <instrumentation><logging level="" threadids="no"/></instrumentation>

 <lwmd enabled="true" sample="1m" autobaseline="1h" growth="10" size="10"/>

 <process name="ASP.NET", enablealldomains="false">
      <logging level=""/>
      <points file="ASP.NET.points"/>
      <appdomain name="1/ROOT/MSPetShop" website="Default Web Site"
enabled="true">
           <points file="DefaultWebsite-MSPetShop.points"/>
      </appdomain>
 </process>
</probeconfig>
```

4 按 “搜寻和标准插桩”（第 281 页）中的说明重新启动 IIS。

不同 IIS 路径下具有相同名称的虚拟目录 (AppDomain)

您可以区分同一 IIS 服务器上具有相同名称的两个或多个 AppDomain。如果有 3 个虚拟目录 (AppDomain) 具有同一个名称 CallChain，则考虑以下配置。

在 **probe_config.xml** 文件中，可以通过包含 IIS 配置路径来区分 AppDomain。

上例中的 3 个 CallChain 应用程序的配置如下所示：

```
<appdomain enabled="false" name="1/ROOT/CallChain/CallChain" website="Default
Web Site">
     <points file="Default Web Site-CallChain-CallChain.points" />
</appdomain>
<appdomain enabled="false" name="1/ROOT/CallChain" website="Default Web Site">
    <points file="Default Web Site-CallChain.points" />
</appdomain>
<appdomain enabled="false" name="2/ROOT/CallChain" website="WebSite2">
    <points file="WebSite2-CallChain.points" />
</appdomain>
```

生成的探测器使用 IIS 路径进行区分，在 Enterprise UI 中显示为：
1ROOTCallChain.NET, 1ROOTCallChainCallChain.NET, 2ROOTCallChain.NET

与 9.01 之前版本的向后兼容性

为了满足向后兼容性， 9.01 或更高版本的代理将能够读取和处理 ASP.NET AppDomain 中低于 9.01 版的探测器配置。“早期”格式如下例所示：

```
<appdomain name="CallChain">
    <points file="CallChain.points" />
</appdomain>
```

如果使用早期格式，则代理的行为将返回到先前版本的行为。

- 同时，将启用或禁用所有带有名称 “CallChain”（在本例中）的 AppDomain。
- 服务器上的所有 CallChain 探测器实例将整合到一个探测器中。
- 探测器和服务器请求的趋势线将沿着先前版本继续使用。

如果不需要向后兼容性 （如趋势线），则建议您不要使用早期格式的配置。

对于使用早期格式配置的 AppDomain，如果需要新行为，则应从 probe_config.xml 文件中删除“旧”格式条目。然后从探测器系统的开始菜单中运行 “重新扫描 ASP.NET 应用程序” 。此操作会添加新格式的 AppDomain 条目，允许您区分同一 IIS 服务器上具有相同名称的不同探测器。

升级安装将保留早期版本的 AppDomain 配置，并修改 probe_config.xml 为所有未列出的 AppDomain 添加新格式配置。

搜寻应用程序中的类和方法

要监控不熟悉的应用程序的性能，请使用随 .NET 代理安装的 Reflector 自动搜寻工具，以查找应用程序中要添加到由探测器使用的插桩中的类和方法。 Reflector 可执行文件位于 **< 探测器安装目录 >\bin\reflector.exe** 中。

要使用 Reflector 搜寻类和方法，请执行以下操作：

1. 找到要监控的应用程序的安装目录。
2. 在应用程序安装目录下找到 .dll 文件所在的文件夹。
3. 打开命令提示符，并将目录更改为用于存储应用程序的 .dll 文件的文件夹。
4. 通过在命令提示符处执行以下命令，对当前目录下的所有 .dll 文件和 .exe 文件运行 Reflector：

```
<probe_install_dir>\bin\Reflector.exe
```

您可以通过向命令中添加其他参数，来将 Reflector 限制到特定的 .dll 和 .exe 文件。以下示例演示了以另外一种方式输入上述示例中命令的方法：

```
<probe_install_dir>\bin\Reflector.exe *.dll *.exe
```

此命令明确地指示 Reflector 检查目标目录下的所有 .dll 和 .exe 文件。

要将 Reflector 限制到特定文件，可以输入如下命令：

```
<probe_install_dir>\bin\Reflector.exe WorkHorse.dll Utility.dll
```

此命令明确地指示 Reflector 仅检查两个指定的 .dll 文件。

如果有一个名为 PetShop 的应用程序，并且其 .dll 文件位于 bin 文件夹中，则您可执行如下例所示的命令：

```
C:\>cd "c:\Program Files\Microsoft\PetShop\Web\bin"

C:\Program Files\Microsoft\PetShop\Web\bin>
C:\MercuryDiagnostics\".NET Probe"\bin\Reflector.exe
```

5 Reflector 将显示在指定的 .dll 文件中找到的程序集、名称空间、类和方法的报告。

```
-------------------------------------------------------------------------------
Assemblies:
-------------------------------------------------------------------------------
C:\Program Files\Microsoft\PetShop\Web\bin\PetShop.BLL.dll
C:\Program Files\Microsoft\PetShop\Web\bin\PetShop.DAL.dll
C:\Program Files\Microsoft\PetShop\Web\bin\PetShop.Web.dll


-------------------------------------------------------------------------------
Namespaces:
-------------------------------------------------------------------------------
(8  classes)  PetShop.BLL
(6  classes)  PetShop.DALFactory
(17 classes)  PetShop.Web
(11 classes)  PetShop.Web.Controls
(2  classes)  PetShop.Web.ProcessFlow
(1  classes)  PetShop.Web.WebComponents


-------------------------------------------------------------------------------
(8 classes)  Namespace: PetShop.BLL
-------------------------------------------------------------------------------|
PetShop.BLL.Account (10 Methods)
    Equals              System.Boolean(System.Object)
    Finalize            System.Void()
    GetAddress          PetShop.Model.AddressInfo(System.String)
    GetHashCode         System.Int32()
    GetType             System.Type()
    Insert              System.Void(PetShop.Model.AccountInfo)
    MemberwiseClone     System.Object()
    SignIn              PetShop.Model.AccountInfo(System.String,System.String)
    ToString            System.String()
    Update              System.Void(PetShop.Model.AccountInfo)

PetShop.BLL.Cart (17 Methods)
    Add                   System.Void(System.String)
    Equals                System.Boolean(System.Object)
    Finalize              System.Void()
    get_Count             System.Int32()
    get_Item              PetShop.Model.CartItemInfo(System.Int32)
    get_Total             System.Decimal()
    GetCartItems          System.Collections.ArrayList()
    GetEnumerator         System.Collections.IEnumerator()
    GetHashCode           System.Int32()
    GetInStock            System.Int32(System.String)
    GetOrderLineItems     System.Collections.ArrayList()
    GetType               System.Type()
    MemberwiseClone       System.Object()
```

注意：可以将 Reflector 的输出重定向到某个文件中，如下例所示：

< 探测器安装目录 >\bin\Reflector.exe sys*.dll >< 报告名称 >.txt

随后， Reflector 的输出会重定向到所指定的文件。

可使用该报告中的信息自定义应用程序的插桩，如“自定义 ASP.NET 应用程序的插桩”（第 584 页）中所述。

控制可与代理一起使用的 HP 软件产品

可在以下各种模式中设置 .NET 代理：

- 从开发到预生产测试，再到生产，监控应用程序。
- 配合使用其他 HP 软件产品。
- 用作不向服务器或其他 HP 软件产品报告数据的 Diagnostics Java Profiler。

.NET 代理工作的模式由 **< 探测器安装目录 >/etc/probe_config.xml** 文件中设置的 **<modes>** 元素确定。

<modes> 元素还可用于针对许可证容量确定使用情况 （请参见“基于当前连接的探测器许可证信息”（第 83 页））。对于 Diagnostics，有两种类型的 LTU（要使用的许可证）：

- AM - 在企业模式下使用产品时，通常用于生产环境。
- AD - 在预生产加载测试环境中使用产品时，探测器位于 LoadRunner 或 Performance Center 运行中。

<modes> 元素的值将在您安装 .NET 代理时进行初始设置。请参阅第 8 章“安装 .NET 代理”

要更改 <modes> 元素的值，可以编辑 probe_config.xml 文件。或者，可以重新运行 .NET 代理安装程序并使用 “更改” 选项将模式设置为 Diagnostics Profiler 模式 (PRO)、应用程序管理 / 企业模式 （用于 Diagnostics (Enterprise)）和 / 或 TransactionVision (TV) 或 Diagnostics 模式 （用于 LoadRunner/Performance Center (AD)）。

注意：要在企业模式下使用独立的 Diagnostics Profiler for .NET 或与其他 HP 软件产品集成，请联系 HP 软件客户支持以购买 HP Diagnostics。

如果要在所使用的 HP 软件产品的用户界面中查看 Diagnostics 数据，则必须执行额外一些配置步骤。有关与 Business Service Management、 LoadRunner 或 Performance Center 集成的详细信息，请参见 “设置与其他 HP 软件产品的集成”（第 691 页）中的部分。

以下各节将说明如何配置 **<modes>** 元素的每个产品模式 （另请参见 “<modes> 元素”（第 548 页））。

PRO 模式 - Diagnostics Profiler for .NET

设置 PRO 模式后，代理会收集一些性能度量，并在可通过代理主机上的 URL 访问的独立 Diagnostics Profiler for .NET 用户界面中显示这些度量。

在此模式下，即使没有使用 Profiler UI，Profiler 也会一直收集数据，且可以与其他模式同时使用。

PRO 模式不可用于针对许可证容量确定使用情况。

企业模式

在企业模式中配置后，代理可以与 Business Service Management、LoadRunner、Performance Center 等 HP 软件产品配合使用，也可以作为完整的 Diagnostics Enterprise 产品使用。它不仅会捕获单独数据库中的 LoadRunner/Performance Center 运行的数据，也捕获 LoadRunner/Performance Center 运行外部的数据。

AD 模式和 AM 模式都会覆盖此模式。

在企业模式下，也会将数据发送到 Diagnostics .NET Profiler。如果在设置企业模式时还设置了 PRO 模式，则即使未使用 Profiler UI， .NET 代理也会持续收集数据。如果未设置 PRO 模式，则只有启动 Profiler UI 后，代理才会开始收集数据。

.NET 代理默认使用企业模式（如果没有指定 AD 或 AM 模式）。在企业模式下，将针对 AM 许可证容量计算代理。

AM 模式

在 AM 模式下， .NET 代理将会捕获所有插桩数据。可以设置 AM 模式，以避免 Business Service Management 生产部署中的代理被意外包含在 LoadRunner 或 Performance Center 运行内。在 AM 模式下，代理不会被列为 LoadRunner 或 Performance Center 的可用代理。

将始终针对 AM 许可证容量计算 AM 模式下的代理。

AM 模式会取代除 AD 模式之外的所有其他模式。

AD 模式

在 AD 模式下，.NET 代理仅在从 LoadRunner/Performance Center 运行时捕获数据，并将结果存储在用于该运行的特定 Diagnostics 数据库中，例如 Default Client:21。

当代理处于此模式下时，除非探测器参与了 LoadRunner/Performance Center 运行，否则代理将不会使用资源或将任何数据发送到服务器。

AD 模式会取代所有其他模式。因此，如果同时设置了 AD 模式和任何其他模式，则模式将被设置为 AD。

有关如何设置 LoadRunner 集成的信息，请参阅第 23 章，“设置 HP LoadRunner 与 HP Diagnostics 集成”；有关如何设置 Performance Center 集成的信息，请参阅第 24 章，“设置 Performance Center 以使 Diagnostics”。

使用此模式可避免 QA 环境中的代理使用其他资源，以及在未运行负载测试时，仍然继续向 Diagnostics 控制台数据集报告数据。

在 AD 模式下运行探测器的另一个好处在于，当探测器在 LoadRunner 或 Performance Center 测试运行中运行时，只会针对 AD 容量计算 AD 模式下的探测器。例如，如果在 LoadRunner/Performanace Center AD 模式下安装了 20 个代理，但一次测试中只有 5 个探测器，则只会针对 AD 许可证容量计算 5 个探测器。

TV 模式

此模式会将事件发送到 Transaction Vision，且可以与其他模式同时使用。TV 模式不可用于针对 HP Diagnostics 许可证容量确定使用情况。

关于 AD 模式和企业模式的注意事项

.NET 代理通过 Diagnostic Mediator 得知 LoadRunner/Performance Center 运行。

如果 LoadRunner/Performance Center 开始测试已插桩、但当前未运行的应用程序（例如，首次点击的 Web 应用程序），则当应用程序开始执行时，Diagnostics Agent 将不会得知此次运行。这是因为代理没有足够的时间进行初始化并开始监听负责此通知的 Mediator。

要解决此问题，需要在开始 LoadRunner/Performance Center 运行之前调用 Web 应用程序，“准备好”（初始化）.NET 代理。这将会初始化 Web 应用程序的进程（工作进程）和探测器，以便其准备好从 Mediator 接受运行信息。

配置对基于 MSMQ 的通信的支持

要配置 .NET 代理以支持基于 MSMQ 的通信，请在 AppDomain 的作用域中包含 **msmq.points** 文件，如从 **< 探测器安装目录 >/etc/probe_config** 文件中摘录的以下示例所示：

```
<process name="SimplestQueuingSender">
<points file="msmq.points"/>
<modes enterprise="true"/>
</process>
```

配置延迟剪裁和阻断

当 .NET 代理确定由于 Diagnostics 服务器无法跟上探测器所捕获的数据量而将要用尽资源时，它会使用延迟剪裁过程自动减少探测器所捕获的方法数。默认情况下会启用延迟剪裁功能，以便能够根据需要调整探测器的工作负载。

启用延迟剪裁时，.NET 代理会通过忽略其总延迟时间小于特定最小延迟阈值的方法，来剪裁探测器所捕获的方法数。剪裁背后的思路是：与其让探测器瘫痪或停止运行，不如放弃捕获延迟时间较少且不太重要的方法。剪裁能让探测器继续运行，以便其捕获延迟时间较长并且更受关注的方法。

注意：由于线程和缓冲，可将被剪裁的方法的部分信息传输到 Diagnostics 服务器。Diagnostics 服务器在检测到它仅收到方法的部分信息时，将发出警告消息。应忽略这些警告消息，除非您希望捕获所有方法的信息。

注意：

- 延迟剪裁和阻断不会在 Profiler 用户界面上体现出来。
- 可以配置 Diagnostics 服务器，使其应用探测器数据的其他剪裁方法，这将会影响 Diagnostics 用户界面所显示的数据的粒度。

禁用延迟剪裁

默认情况下会为 .NET 代理启用剪裁。要禁用剪裁，必须更改配置。

要禁用延迟剪裁，请执行以下操作：

将 **latency** 标记添加到 **< 探测器安装目录 >/etc/probe_config.xml** 配置文件中，如下例所示：

```
<trim>
 <latency enabled="false" />
</trim>
```

用于启用延迟剪裁的延迟元素属性是 **enabled**。**enabled** 设置为 true 时，会启用延迟剪裁。**enabled** 属性设置为 false 时，会禁用延迟剪裁。该属性的默认值为 true。

有关 **latency** 元素的属性和元素的描述，请参阅第 13 章 “了解 .NET 代理配置文件”

启用延迟剪裁

默认情况下会为 .NET 代理启用剪裁。如果在安装之后禁用了剪裁，则必须更改配置，才能再次启用剪裁。

要启用延迟剪裁，请执行以下操作：

在 **< 探测器安装目录 >/etc/probe_config.xml** 配置文件中，更改 **latency** 元素的 enabled 属性值，如下例所示：

```
<trim>
 <latency enabled="true" />
</trim>
```

用于启用延迟剪裁的延迟元素属性是 **enabled**。**enabled** 设置为 **true** 时，会启用延迟剪裁； **enabled** 属性设置为 **false** 时，会禁用延迟剪裁。该属性的默认值为 **true**。

有关 **latency** 元素的属性和元素的描述，请参阅第 13 章 “了解 .NET 代理配置文件”

设置延迟剪裁阈值

默认情况下，所设置的延迟剪裁阈值可以剪裁掉其延迟小于 2 毫秒的方法，而不会剪裁延迟大于 100 毫秒的方法。

您可以通过调整 **min** 属性值来设置最小剪裁阈值，通过调整 **max** 属性值来设置最大剪裁阈值。可在 **< 探测器安装目录 >/etc/probe_config.xml** 配置文件中的 **latency** 元素中指定上述属性。

```
<trim>
 <latency enabled="true" min="50" max="100" />
</trim>
```

用于控制剪裁阈值的延迟元素属性包括：

- **min**

  设置最小延迟阈值。当启用延迟剪裁时，其延迟小于或等于此属性值的方法将被剪裁掉。如果不指定该属性的值，则会使用默认值 2 ms。

  min 属性值越小，对应用程序性能造成不利影响的可能性就越大。较小的值意味着将捕获更多的低延迟方法，从而减少被剪裁的方法。

  如果必须捕获所有方法的信息，请将 **latency enabled** 设置为 false 以禁用延迟剪裁。

- **max**

  设置最大延迟阈值。当启用延迟剪裁时，其延迟大于或等于此属性值的方法将不会被剪裁掉。如果不指定值，则该属性的默认值为 100ms。

有关对 **latency** 元素的属性及元素的描述，请参阅第 13 章“了解 .NET 代理配置文件”

配置延迟剪裁阻断

默认情况下会阻断延迟剪裁。启用阻断时，会根据 Diagnostics 服务器处理后备服务耗用的探测器资源百分比自动调整所执行的剪裁量。

未启用阻断时，将总是会剪裁掉其延迟低于最小方法延迟阈值的方法。

如果探测器使用的资源百分比增加到大于已设置的阻断增量阈值，则有效剪裁阈值将会增大，以剪裁掉具有更长延迟时间的方法。如果使用的探测器资源百分比再次增加到大于阈值，则有效剪裁阈值将会再次增大，以剪裁掉具有更长延迟时间的方法。如果使用的探测器资源百分比降低到小于阻断减量阈值，则有效剪裁阈值将会减小，以再次捕获具有较短延迟时间的方法。

有效剪裁阈值不能增加到大于最大方法延迟阈值，也不能减小到小于最小方法延迟阈值。

下面是 **probe_config.xml** 配置文件中包含阻断属性的 **latency** 元素的一个示例：

```
<trim>
 <latency enabled="true" min="50" max="100"
 throttle="true" incrementthreshold="75"
 decrementthreshold="50" increment="2"/>
</trim>
```

用于控制阻断的 **latency** 元素属性包括：

- **throttle**

  该属性设置为 true 时，会启用阻断；该属性值设置为 false 时，会禁用阻断。该属性的默认值为 true。

- **increment**

  设置当所使用的探测器资源百分比超过 **incrementthreshold** 时，增加的有效剪裁阈值量。还可设置当百分比低于 **decrementthreshold** 时，减少的有效剪裁阈值量。该属性的默认值为 2 ms。

- **incrementthreshold**

  当探测器资源使用百分比达到或超过该属性值时，会触发阻断，以增大有效剪裁阈值。该属性的默认值是 75%。

➤ **decrementthreshold**

当探测器资源使用百分比降低到等于或低于该属性值时，会触发阻断，以减小有效剪裁阈值。该属性的默认值为 50%。

有关 **latency** 元素的属性和元素的描述，请参阅第 13 章，“了解 .NET 代理配置文件”。

配置深度剪裁

通过深度剪裁过程，.NET 代理可以自动减少捕获的方法数。当 Diagnostics 服务器无法满足探测器所捕获的数据量时，探测器可以使用深度剪裁防止资源用尽。默认情况下会启用深度剪裁。

注意：深度剪裁不会在 Profiler 用户界面中体现出来。

启用深度剪裁时， .NET 代理会忽略在大于最大堆栈深度阈值的堆栈深度处调用的方法，从而剪裁所捕获的方法数，而在小于或等于堆栈深度阈值的堆栈深度处调用的方法都将被捕获。剪裁背后的思路是：放弃捕获调用堆栈深处不重要的方法，以便探测器能够继续运行，并捕获调用堆栈中处于更高位置、更重要的方法。

例如，如果堆栈深度阈值为 3，则会生成以下方法调用：

```
/login.do calls a() calls b() calls c()
```

其中，仅 /login.do、 a 和 b 方法会被捕获，而方法 c 将会被剪裁掉。

下面是 **probe_config.xml** 配置文件中包含剪裁属性的 **depth** 元素的一个示例：

```
<trim>
 <depth enabled="true" depth="10" />
</trim>
```

用于控制剪裁的 **depth** 元素属性包括：

- **enabled**

  该属性设置为 true 时，会启用深度剪裁；该属性值设置为 false 时，会禁用深度剪裁。该属性的默认值为 true。

- **depth**

  设置用于深度剪裁的阈值。启用深度剪裁时，会剪裁调用深度等于或低于该属性值的方法。该属性的默认值为 25。

  将 **depth** 设置为较低的值可显著减少捕获过程的开销。有关 **depth** 元素的属性和元素的描述，请参阅第 13 章，“了解 .NET 代理配置文件”。

配置 URI 截断和映射

任何 HTTP/S 服务器请求 URI 在被探测器报告之前均可以进行转换。此转换基于由 **probe_config.xml** 配置文件中 **urireplacepattern** 元素控制的正则表达式匹配和替换。默认情况下关闭。

当您看到太多服务器请求，想要使用一个能够聚合这些 URI 的简单服务器请求 URI 来替换众多服务器请求 URI 时，此转换十分有用。

重要信息：过度使用此功能会影响性能。

示例显示如下：

```
<symbols maxuri="" maxsql="">
        <urireplacepattern enabled="true">
         <pattern value="s/TestService1/CommonService/"/>

        <pattern value="s/TestService2/CommonService/"/>
  </urireplacepattern>
 </symbols>
```

模式值使用的语法为 s/search_pattern/replace_pattern/。

search_pattern 和 replace_pattern 应当在 / 之内。如果在模式中使用了 /，则应该使用字符 # （而不是 /）作为分隔符。

模式适用于所有服务器请求，且将按照 probe_config.xml 文件中的指定顺序应用到 URI。

如果启用了 urireplacepattern，则默认会配置两个默认模式。

第一个默认模式将用于剪裁含有 ; 或 /! 的服务器请求。在这些标记之后的所有内容将从服务器请求中删除。

使用的模式为：s#(;|/?\\!).*$##"

第二个默认模式将使用固定标记 ("/Static Content") 替换图像、PDF 和 DOC 加载。

使用的模式为：

s#(?<word1>^.*)(/.*\\.js|css|jpg|gif|png|pdf|html|doc|docx)#${word1}/Static Content#

这两个模式均可以自定义。

配置 .NET 代理以实现轻量级内存诊断

轻量级内存诊断 (LWMD) 功能是指捕获和分析与 Collection 相关的使用情况数据的能力。具体地讲， Collection 是指可实现 **System.Collections.ICollection** 或 **System.Collections.Generic.ICollection** 接口的类，例如 ArrayList、HashTable、DataView 等。最常见的 .NET 内存泄露都发生在没有得到正确维护的 Collection 中。

安装 .NET 代理后，.NET 代理探测器的默认配置会关闭 LWMD。要启用 LWMD 功能，您必须对 **probe_config.xml** 文件进行两项修改：

- 必须启用 <lwmd> 元素 （请参阅 “<lwmd> 元素”（第 542 页））。
- 必须添加对 **Lwmd.points** 文件的一个或多个引用，具体如下所述。

注意：启用探测器捕获收集度量时，可能会导致应用程序在主机上产生额外的开销。

要捕获进程或 appdomain 的收集度量，请执行以下操作：

将 **Lwmd.points** 文件的 **points** 标记添加到 **probe_config.xml** 配置文件中的 **process** 标记或一个或多个 **appdomain** 标记中。

安装 .NET 代理时，会将 **Lwmd.points** 文件与 **ASP.NET.points** 和 **ADO.points** 文件一起安装在 **< 探测器安装目录 >/etc/** 目录下。 **Lwmd.points** 文件包含启用收集度量捕获所需的插桩说明。

要为在进程下运行的所有已启用 AppDomain 启用 LWMD 插桩，请将 points 标记添加到 **probe_config.xml** 配置文件中的 process 标记中。例如，要为所有已启用的 ASP.NET appdomain 启用 LWMD 插桩，请执行以下添加操作：

```
<process name="ASP.NET", <enablealldomains="false">
    <points file="ASP.NET.points" />
    <points file="ADO.points" />
    <points file="Lwmd.points"/>
    <appdomain name="1/ROOT/your_app_name" website="Default Web Site"
enabled="true">
        <points file="DefaultWebsite-your_app.capture points" />
    </appdomain>
</process>
```

要为在进程下运行的已启用的特定 AppDomain 启用 LWMD 插桩，请将 points 标记添加到 **probe_config.xml** 配置文件中的 appdomain 标记中。您可以将 points 标记添加到一个或多个 appdomain 标记中。例如，要为 ASP.NET 进程下运行的“your_app_name” appdomain 启用 LWMD 插桩，请执行以下添加操作：

```
    <process name="ASP.NET", <enablealldomains="false">
    <points file="ASP.NET.points" />
    <points file="ADO.points" />
    <appdomain name="1/ROOT/your_app_name" website="Default Web Site"
enabled="true">
        <points file="DefaultWebsite-your_app.capture points" />
        <points file="Lwmd.points"/>
    </appdomain>
</process>
```

要禁用 LWMD，请执行以下操作：

要禁用 LWMD 功能，您必须对 **probe_config.xml** 文件进行两项修改：

- 禁用 <lwmd> 元素 （请参阅 “<lwmd> 元素”（第 542 页））。
- 从所有 process 标记和相应的 appdomain 标记中，删除 **Lwmd.points** 文件的 points 标记。

配置文件中没有 LWMD points 标记时，探测器将无法查找 Lwmd.points 文件中包含的 LWMD 插桩说明，因此，探测器将不会插桩 Collection 使用情况。

要控制 LWMD 插桩，请执行以下操作：

安装 .NET 代理后， Lwmd.points 文件的默认配置包含在更广泛的程序集、appdomain、名称空间和类中插桩 Collection 使用情况的说明。您可以修改应用程序的点文件，缩小要检查的 Collections 的范围。LWMD 插桩将作为调用方插桩实现。有关该插桩如何工作的说明，请参阅 “调用方插桩”（第 398 页）。

注意：推荐的最佳做法是缩小 LWMD 插桩的范围。

要缩小需要检查的 Collections 范围，请执行以下步骤：

1. 从 process 标记和相应的 appdomain 标记中，删除 Lwmd.points 文件的 points 标记。这将会删除指定了较大插桩范围的 LWMD 设置。
2. 将 LWMD 部分添加到 process 或 appDomain 的点文件中。例如，将以下内容复制并粘贴到您的 **your_app.points** 文件中以完成此操作：

```
[LWMD]
keyWord = lwmd
scope =
ignoreScope =
```

3 设置 LWMD 部分中的 scope 和 ignoreScope 参数，以缩小要检查的 Collection 的范围。示例：

```
[LWMD]
keyWord = lwmd
scope = !my_namespace\..*
ignoreScope = !my_namespace.my_class1\..*
```

上述示例将插桩通过 my_namespace 名称空间构造的所有 Collection，但通过 my_namespace.my_class1 类中的任意方法构造的 Collection 除外。

对于 LWMD 插桩，存在一个未公开的内部默认 ignoreScope 值，会始终将它和您输入的任何值包含在内。默认值包含与 .NET 基础结构相关的名称空间和类，如果插桩这些内容将会对应用程序造成不利影响，例如 !System.*、 !Microsoft.* 等。

限制异常堆栈跟踪数据

代理会为抛出服务器请求的异常收集异常数据，并将这些信息显示在 Diagnostics UI 中。可以选择在收集的异常数据中包含堆栈跟踪。

然而，由于无关的异常堆栈跟踪会增加显示、数据收集和传输操作的负载，因此，收集所有异常的堆栈跟踪数据通常是不可取的。因此，您可以限制为其收集堆栈跟踪数据的异常类型。例如，筛选基于应用程序服务器的错误（如 **System.Security.Authentication.AuthenticationException**）可将堆栈跟踪用于更多特定于应用程序的错误。

可通过下列三种方式控制收集的堆栈跟踪数据：限制特定的异常类型、限制为其收集堆栈跟踪数据的异常数目以及限制堆栈跟踪数据的大小。

注意：您可以通过在 **probe_config.xml** 文件中将 **captureexceptions enabled** 设置为“false”，来禁用所有堆栈跟踪收集。默认情况下会启用堆栈跟踪收集。

本节包括：

- 限制特定异常类型
- 限制每个服务器请求的异常数
- 限制堆栈跟踪的大小
- 示例

限制特定异常类型

通过在 **probe_config.xml** 文件中设置 **exclude** 和 **include** 属性，可限制为其收集堆栈跟踪数据的异常，如下例所示：

```
<exclude>
      <exceptiontype name="System.ArithmeticException"/>
</exclude>
<include>
      <exceptiontype name="System.DivideByZeroException"/>
</include>
```

对于指定要排除或包含的异常类型的子类型，除非在包含或排除列表中另行明确指示，否则它们也将被排除或包含。

下图显示了根据前面的示例要包含和排除的异常类型：

对 **probe-config.xml** 文件的更改会立即生效，而不必重新启动应用程序。

限制每个服务器请求的异常数

默认情况下，.NET 代理探测器仅收集在服务器请求期间遇到的头 4 个异常的堆栈跟踪数据。如果应用程序包含较多需要查看堆栈跟踪信息的异常，则可以在 **probe_config.xml** 文件中增大 **max_per_request** 属性的值。与收集的所有度量一样，收集数据量的增加会给 Diagnostics 服务器带来更高的负载。

限制堆栈跟踪的大小

默认情况下，捕获的堆栈跟踪数据可以是任意大小。您可以限制堆栈跟踪字符串的大小，以提高“异常”选项卡的可读性。在 **probe_config.xml** 文件中，将 **max_stack_size** 属性的值设置为最大堆栈跟踪字符串。与收集的所有数据一样，收集数据量的增加会给 Diagnostics 服务器带来更高的负载。默认情况下，该属性设置为 0，表示堆栈跟踪大小不受限制。

示例

以下设置会启用最大堆栈跟踪字符串大小为 2048 的异常堆栈跟踪。

```
<captureexceptions enabled="true" max_per_request="4" max_stack_size="2048">
  <exclude>
      <exceptiontype name="System.ArithmeticException"/>
  </exclude>
  <include>
      <exceptiontype name="System.DivideByZeroException"/>
 </include>
</captureexceptions>
```

禁用日志记录

您可以通过更改 **probe_config.xml** 文件中 ASP.NET 进程部分的 **logging level** 标记禁用应用程序日志记录，如下例所示：

```
<process name="ASP.NET">
      <logging level="off"/>
</process>
```

您可以通过更改插桩部分的 **logging level** 标记禁用插桩日志记录，如下例所示：

```
<instrumentation>
      <logging level="off" />
</instrumentation>
```

覆盖默认探测器主机名

通过 **registered_hostname** 属性，您可以覆盖探测器用于在 Diagnostics 命令服务器中注册的默认主机名。在防火墙或 NAT 已开启或者探测器主机已配置为多域插桩的情况下，除非您覆盖默认主机名，否则 Diagnostics 命令服务器可能无法与探测器进行通信。

要覆盖探测器的默认主机名，请执行以下三个步骤。

1 首先，在 **probe_config.xml** 文件的 .NET 代理 **<diagnosticsserver>** 元素中，将 **registered_hostname** 属性设置为允许 Diagnostics 命令服务器与探测器通信的备用计算机名或 IP 地址。

例如：

```
<diagnosticsserver url="http://localhost:2006/commander"
registered_hostname=" my_host_name "/>
```

2 其次，使用 .NET 度量代理注册主机的备用计算机名或 IP 地址。要执行此操作，请在 **metrics.config** 文件中创建 **metrics.agent.registered_hostname** 条目。您可以将该条目添加在 metrics.systemgroup 条目下。

例如：

```
metrics.systemgroup = Default
metrics.agent.registered_hostname = my_host_name
```

3 最后，必须重新启动 .NET 代理和 .NET 度量代理，才能使此更改生效。

注意：

- 您需要设置 **registered_hostname** 属性，以便正确处理 IIS 主机头技术的使用。
- 因为 NAT 或防火墙而设置 **registered_hostname** 属性，这一问题仅会在使用 LoadRunner、 Performance Center 或 Diagnostics Standalone 的测试环境中发生。
- 但是，如果您要在使用 Business Service Management 或 Diagnostics Standalone 的生产环境中设置 registered_hostnamein，则所指定的名称会在系统运行状况中显示为主机名。

列出在主机上运行的探测器

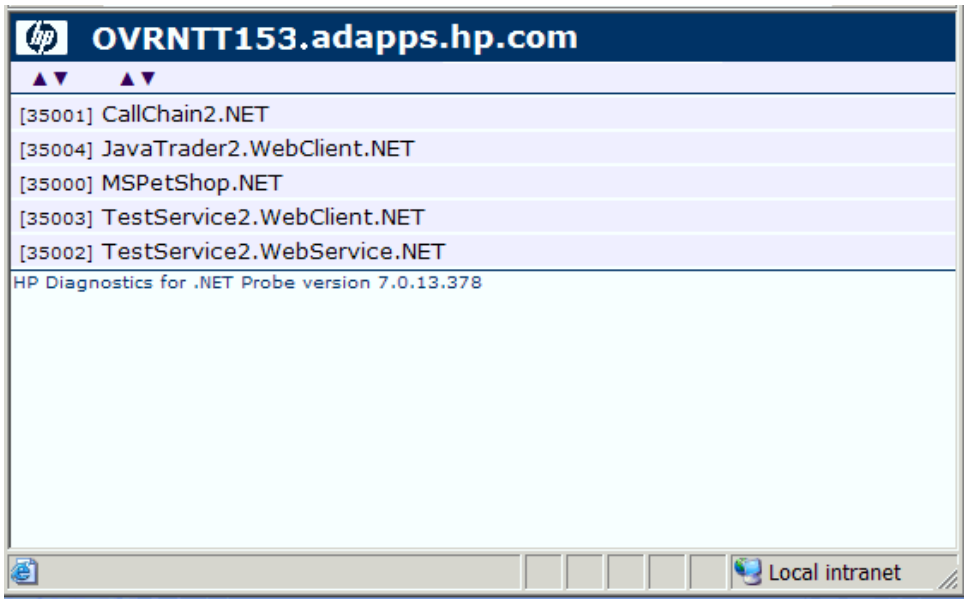
当多个探测器在单个主机上运行时，您无法得知每个探测器所使用的端口，因为端口是根据启动应用程序 （和探测器）时可用的端口随机分配的。启动或停止应用程序时，为给定应用程序的探测器分配的端口可能会发生更改。

可以通过访问下面的 URL，确定要在主机上运行的探测器及其使用的端口：

```
http://<probe_host>:<port>
```

对于端口值，输入端口号 35000 或 35001。输入任何一个均可。

探测器和端口的列表如下例所示：



.NET Profiler 的身份验证和授权

可以在 **<探测器安装目录>/etc/probe_config.xml** 文件中管理 Profiler 用户的身份验证和授权。

注意：如果将 .NET 代理配置为可与 Diagnostics 服务器配合使用，则可以通过探测器所连接的 Diagnostics 命令服务器管理探测器 (Profiler) 的授权和身份验证设置。有关详细信息，请参阅 “用户身份验证和授权”（第 755 页）。

通过 Diagnostics 服务器访问探测器时，默认用户名和密码均为 admin。

如果将 .NET 代理仅安装为 Profiler，则默认情况下，用户不需要输入用户名和密码便可访问 Profiler。

但是，可以将 Profiler 配置为需要进行用户身份验证。如果将 Profiler 配置为需要进行用户身份验证，则可以定义用于访问 Profiler 所需的密码。

要将 Profiler 配置为需要进行用户身份验证，请执行以下操作：

➤ 转到 **<探测器安装目录>/etc/probe_config.xml** 文件并将 **profiler authenticate** 的值设置为 **true**。

```
<profiler authenticate="true">
      <authentication username="Test" password="uU8X9zOtl6Twi7TkGAhQ="/>
</profiler>>
```

如果不设置用户名和密码，则默认的用户名和密码均为 admin。

要为 .NET Diagnostics Profiler 用户创建新的用户名和密码，请执行以下操作：

1 使用 **<探测器安装目录>/bin** 目录下的 **PassGen.exe** 实用程序生成新用户名和密码。输入用于加密的用户名和密码。为用户生成的加密密码为 FIPS-2 兼容。

2 在 **探测器安装目录>/etc/probe_config.xml** 文件中的 **<profiler authenticate="true">** 行后，按以下格式为每个新用户输入用户名和密码：

```
<profiler authenticate="true">
      <authentication username="" password=""/>
</profiler>
```

- ➤ 对于 **authentication username**，请输入运行 PassGen 实用程序时所选的用户名。
- ➤ 对于 **password**，请输入 **PassGen.exe** 实用程序返回的编码字符串。

注意：如果定义了用于访问 Profiler 的新用户名和密码，则可以不再使用默认的用户名和密码 （admin、 admin）。确切地说，必须使用已定义的新用户名之一。

配置用户 ID

可以按 Web 服务的特定用户分组 Web 服务度量。然后，将聚合该用户的度量并在 “服务 (按用户 ID)” 和 “操作 (按用户 ID)” 视图中显示。

如果需要确定使用特定服务的用户以及他们使用该服务的频率，则按用户 ID 聚合数据将十分有用。用户 ID 对 Business Service Management 也十分有用。 BSM 用户可以通过比较各自的性能特征，查看同一应用程序的性能。

配置用户 ID 的操作是可选操作。默认情况下，会将受监控 Web 服务的用户 ID 报告为 Web 服务用户的 IP 地址。

可通过以下三种方式定义用户 ID：

- 在 SOAP 请求中出现的值
- 定义为出现在 HTTP 标头中的值
- 特定 IP 地址或一个 IP 地址范围

配置用户 ID 的基本过程

配置用户 ID 的基本过程如下：

1. 对于每个要按用户对度量进行分组的 .NET 探测器，请按 “.NET 代理的用户 ID 规则语法和示例”（第 617 页）中所述，更新 probe_config.xml 文件。
2. 如果要配置 5 种以上的用户类型，请在 **server.properties** 文件中更新 **max.tracked.ids.per.probe** 设置。

关于用户 ID 规则

用户 ID 的分配由 **probe_config.xml** 文件中的用户 ID 规则控制。

每个类别的用户 ID 都有其自身的规则：SOAP 规则、 HTTP 头规则和 IP 规则。无论规则的定义顺序如何，都会按特定的顺序应用规则。首先会应用 SOAP 标头规则，其次应用 HTTP 标规则，最后应用 IP 规则。

并不需要使用所有的规则类型。可以有 SOAP 规则，而没有 HTTP 规则和 IP 规则。如果与这三种规则都不符合，则使用原始的 IP 地址作为用户 ID。

SOAP 规则允许从 SOAP 标头、信封或正文中的 XML 元素获得用户 ID。规则会指定一个正则表达式，用于匹配用户调用的 Web 服务的名称。有关使用正则表达式的帮助，请参阅 “使用正则表达式”（第 882 页）。

如果存在与 Web 服务名称匹配的对象，则代理 / 探测器将尝试在 SOAP 规则定义的相应 SOAP 位置的 consumeridfield 字段中查找已定义的元素。如果未找到元素，则会跳过此规则，代理 / 探测器将继续转到已定义的下一规则。

HTTP 头规则允许从 HTTP 请求中的 HTTP 头集合中的头获得用户 ID。

IP 规则允许从 IP 地址到用户 ID 的映射中获得用户 ID。该规则用于定义要分配到用户 ID 的 IP 地址或地址范围。

.NET 代理的用户 ID 规则语法和示例

规则的语法和示例特定于用户 ID 的定义方式。

SOAP 规则

SOAP 规则允许从 SOAP 标头、信封或正文中的 XML 元素获得用户 ID。

下面显示了根据 SOAP 标头中的值配置用户 ID 的示例：

```
<consumeridrules enabled="true">
  <soaprules>
     <soaprule id="SOAP1" rule="TestService" location="soap-header"
consumeridfield="Caller"/>
  </soaprules>
</consumeridrules>
```

id= 属性可以是要用来标识规则的任何名称； .NET 探测器不会使用此属性。

必须为 soaprule 定义 rule= 属性。该规则是一个正则表达式，用于与用户调用的 Web 服务名称进行匹配，您也可以使用确切的 Web 服务名称。

location= 可以设置为“soap-header”、“soap-envelope”、“soap-body”。如果不指定位置，则默认使用“soap-header”。如果为任何 SOAP 规则配置了位置，则必须为所有 SOAP 规则配置位置，否则将会发生严重的错误，并且会禁用基于 SOAP 逻辑的用户 ID。

必须为 soaprule 定义 consumeridfield= 属性。 SOAP 标头、信封或正文中的元素，其值将用作用户 ID。

如果存在与 rule= 属性所指定模式匹配的对象，则 .NET 代理将尝试查找在 consumeridfield 中定义的元素的文本元素。consumeridfield 中的元素可以是限定的名称（由名称空间名称和本地部分组成），也可以是没有关联名称空间的非限定名称。如果在特定位置中未找到元素，则会跳过此规则，探测器将继续转到已定义的下一规则。

例如，下面的规则与名为 TestService 的 Web 服务相匹配，并且将 Caller 元素的值用作用户 ID：

```
<soaprule id="SOAP1" rule="TestService" location="soap-header"
consumeridfield="Caller"/>
```

只要 TestService Web 服务的调用者具有已定义的 Caller 值，就会按 Caller 的不同值对度量进行分组。以下是从 SOAP 标头摘录的一段内容，该标头将映射到此 TestService 调用者的用户 ID “Customer2”：

```
SoapTest1;WS<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <env:Header>
      <Caller>Customer2</Caller>  <-- The consumer id returned is
                                                  "Customer2"
   </env:Header>
   <env:Body   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <m:sell    xmlns:m="http://www.bea.com/examples/Trader">
         <string     xsi:type="xsd:string">sample string</string>
         <intVal     xsi:type="xsd:int">100</intVal>
      </m:sell>
   </env:Body>
</env:Envelope>
```

启用 SOAP 捕获

因为 SOAP 信封可能会非常大，所以提供了 <soapcapture> 元素，以便您控制用于捕获 SOAP 请求和响应的开销 （主要是内存开销）。

<soapcapture enabled="true">

<soapcapture> 元素用于控制是否捕获 SOAP 请求和响应。如果禁用该元素，则不捕获 SOAP 请求和响应。这意味着将不在 TransactionVision 事件中包含 SOAP 请求或响应，也不在 SOAP 错误中显示任何 SOAP 请求，并且无法根据 SOAP 标头、信封或正文配置用户 ID。

<soapcapture> 设置会覆盖 <soaprequestforsoapfault> 中的设置，后者用于控制 SOAP 错误的 SOAP 有效负载捕获。请参阅 “配置 SOAP 错误数据”（第 621 页）。

HTTP 头规则

HTTP 头规则允许从 HTTP 请求中的 HTTP 头集合中的头获得用户 ID。必须为 HTTP 规则元素定义 rule 和 consumeridfield 属性，也可以为用户定义 id 属性，以标识各个规则。

该规则是一个正则表达式，用于匹配用户要将 HTTP 请求发送到的目标 URL。如果存在匹配，.NET 探测器将尝试为 consumeridfield 中定义的标头名称查找 HTTP 头。如果在 HTTP 头集中找不到头名称，则会跳过此规则，探测器将继续转到已定义的下一规则。

httpheader 规则示例：

```
<consumeridrules enabled="true">
  <httpheaderrules>
      <httpheaderrule id="httpHeader 1" rule="/Webservice/.*
consumeridfield="Caller"/>
  </httpheaderrules>
</consumeridrules>
```

IP 地址规则

IP 规则允许通过从 IP 地址映射用户 ID 来获得用户 ID。必须为 IP 规则元素定义 rule 和 consumerid 属性，也可为用户定义 id 属性，以标识各个规则。

该规则用于定义要分配到用户 ID 的 IP 地址或地址范围。该规则可定义为单个 IP 地址，例如 19.225.17.125。该规则还可以定义范围，例如 19.255.17.125,19.255.17.255。

还可以在 IP 地址的 octet 中使用星号，以匹配该 octet 中的任何内容，例如 19.255.17.*。可以在 octet 中定义一个范围，以匹配该 octet 组中的值范围，例如 19.255.17.20-255。也可以使用这些方式的组合，例如 19.*.17.20-255、20.*.10-55.*。如果存在匹配项，则 .NET 探测器会将用户 ID 设置为在规则中定义的用户 ID。

示例：

```
<consumeridrules enabled="true">
  <iprules>
     <iprule id="IpTest1" rule="18.*.1-20.*" consumerid="Client1"/>
     <iprule id="IpTest2" rule="17.*.*.*" consumerid="Client2"/>
     <iprule id="IpTest3" rule="19.255.17.125,19.255.17.255" consumerid="Client3"/>
  </iprules>
</consumeridrules>
```

配置 SOAP 错误数据

如果检测到 SOAP 错误，则会将 SOAP 有效负载包括在 SOAP 错误数据中。只有当存在 SOAP 错误时，才会捕获 SOAP 有效负载。

可以在 Diagnostics UI 中查看 SOAP 错误实例树（调用配置文件）的有效负载信息。

由于有效负载可能包含如信用卡号等敏感信息，默认情况下禁用对 SOAP 错误的有效负载捕获。要启用 SOAP 错误的有效负载捕获，请在 .NET 探测器系统上的 **probe_config.xml** 文件中设置 **<soaprequestforsoapfault enabled="true"/>**。

您还可以使用 **<soaprequestforsoapfault>** 元素中的 **maxsize** 属性来定义有效负载的大小限制。例如，以下条目可将 SOAP 负载长度由默认的 5000 增加到 10000：

```
<soaprequestforsoapfault enabled="true" maxsize="10000"/>
```

<soapcapture> 元素将覆盖 <soaprequestforsoapfault> 元素。因此，如果禁用 <soapcapture>，则即使 <soaprequestforsoapfault> 设置为 True，<soaprequestforsoapfault> 也会被禁用。同时，无论 <soapcapture> 最大值被设置为多少，它都会覆盖 <soaprequestforsoapfault> 最大值。也就是说，如果 <soapcapture> 最大值被设置为 5000，而 <soaprequestforsoapfault> 最大值被设置为 10000，则有效荷载大小的最大值是 5000。

收集其他探测器度量或修改探测器度量

可以使用 **< 探测器安装目录 >\etc\probe_config.xml** 文件中的 <metrics> 和 <metric> 元素配置 .NET 代理，以根据 Perfmon 计数器收集其他探测器度量。有关详细信息，请参阅“<metric> 元素”（第 546 页）和“<metric> 元素”（第 546 页）。

还可以使用 <metric> 元素修改探测器度量。但是请注意以下特殊情况：

- 如果要将度量从一个度量类别移动到另一个类别，则必须更改度量的组属性和度量名称属性。这是因为现有度量名称已在 Diagnostics Mediator 上的旧组中注册，所以此关联无法更改。
- 如果要重新定义现有探测器度量，最好是创建一个全新的度量条目，而不是将其他 Perfmon 计数器分配到该度量。这可确保避免聚合截然不同的数据。

性能计数器的安全性

.NET 代理使用性能计数器来收集探测器度量。这要求受 .NET 代理监控的应用程序进程具有访问性能计数器的权限。由于每个进程都作为用户帐户运行，所以该用户帐户必须具有访问性能计数器的权限。为此，最简单的方法是将进程运行时所用的用户帐户添加到 “性能监控器用户”组中。

但是， Microsoft 已经在 Windows Vista SP2、 Windows Server 2008 SP2、Windows 7 和 Windows Server 2008 R2 中引入了 “虚拟帐户”概念 （有关详细信息，请参见 http://technet.microsoft.com/zh-cn/library/dd548356(WS.10).aspx）。这些操作系统已经在 IIS 中使用了 “虚拟帐户”概念，并且默认情况下， IIS 中的应用程序池会作为 **ApplicationPoolIdentity** 运行。因为此用户帐户是虚拟的帐户，所以需要通过特殊步骤将此用户帐户添加到性能监控器用户组中。

在 Windows 2008 R2 和 Windows 7 中，执行以下操作：

1. 打开“服务器管理器”工具。有很多种方式可以打开该工具，其中之一是通过“管理工具”打开。
2. 在左侧窗格中，找到 “配置”下的 “本地用户和组”。
3. 单击“+”将其展开。
4. 双击“组”。
5. 双击“Performance Monitor Users”组。
6. 单击“添加 ...”按钮。
7. 单击“位置 ...”按钮。
8. 选择本地计算机。
9. 单击“确定”按钮。
10. 确保对象类型包括 “内置安全主体”。
11. 在文本框中输入 “IIS APPPOOL\<name of the application pool>”，例如 IIS APPPOOL\My WebService App Pool，其中 My WebService App Pool 是应用程序池的名称。
12. 单击“确定”按钮。

在 Windows 2008 SP2 和 Windows Vista SP2 中，执行以下操作：

1. 打开命令提示符窗口。
2. 键入 **net localgroup "Performance Monitor Users" "IIS APPPOOL\ <name of application pool> /ADD**，其中 <name of application pool> 是应用程序池的名称。
3. 如果成功，则会显示 “命令已成功完成”。

第 V 部分

配置通过代理服务器和防火墙进行的通信

本部分包括：

- 针对 HTTP 代理配置 Diagnostics 服务器和代理
- 配置 Diagnostics 以在防火墙环境中工作

15

针对 HTTP 代理配置 Diagnostics 服务器和代理

本章提供用于在各个 Diagnostics 组件之间启用 HTTP 代理通信的配置步骤。这些配置说明适用于深入了解 Diagnostics 且经验丰富的管理员。请谨慎修改 Diagnostics 组件的任何配置设置。

本章包括：

启用 HTTP 代理通信 Diagnostics 服务器

下一节描述如何配置 Diagnostics Commander 服务器和 Diagnostics Mediator 服务器，以使它们通过 HTTP 代理相互通信。

要针对 HTTP 代理通信配置 Diagnostics 服务器，请执行以下操作：

1 在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中设置以下属性：

- 将 **proxy.host** 设置为代理服务器的主机名称。
- 将 **proxy.port** 设置为代理服务器的端口。
- 将 **proxy.protocol** 设置为要用于代理服务器的协议 (http)。
- 将 **proxy.user** 设置为用于验证代理服务器的用户。
- 将 **proxy.password** 设置为用于验证代理服务器的密码。
- 对于要通过代理服务器运行的 Diagnostics Commander 服务器，设置 **commander.url**，以使主机名是实际主机名而不是本地主机。

2 重新启动 Diagnostics 服务器。有关说明，请参阅 “启动和停止 Diagnostics 服务器”（第 68 页）。

为 Java 代理启用 HTTP 代理通信

下一节描述如何配置 Java 代理，以使其通过 HTTP 代理与 Diagnostics Commander 服务器通信。您还可以在 Java 代理安装和设置程序中选中 “使用代理服务器”复选框来配置代理通信。

要为 HTTP 代理通信配置 Java 代理，请执行以下操作：

1 在 < **探测器安装目录** >/etc/dispatcher.properties 中设置以下属性：

- 将 **proxy.enabled** 设置为 true，为 Java 代理启用代理通信，然后输入以下选项。
- 将 **proxy.host** 设置为代理服务器的主机名称。
- 将 **proxy.port** 设置为代理服务器的端口。
- 将 **proxy.protocol** 设置为要用于代理服务器的协议 (http)。
- 将 **proxy.user** 设置为用于验证代理服务器的用户。
- 将 **proxy.password** 设置为用于验证代理服务器的密码。

2 重新启动所插桩的应用程序 VM。

为 .NET 代理启用 HTTP 代理通信

下一节描述如何配置 .NET 代理以通过 HTTP 代理与 Diagnostics Commander 服务器通信：

要配置 .NET 代理以执行 HTTP 代理通信，请执行以下操作：

1 在 < **探测器安装目录** >/etc/probe_config.xml 文件中设置以下 **proxy** 属性，以指向 Diagnostics 服务器主机：

- 将 **uri** 设置为 Diagnostics 服务器的主机。
- 将 **proxy** 设置为代理 url。
- 将 **proxy.user** 设置为用于验证代理服务器的用户。
- 将 **proxy.password** 设置为用于验证代理服务器的密码。

下面的示例显示了 **probe_config.xml** 文件中的以上设置：

```
<diagnosticsserver url="http://<diagserver_host_name>:2006/registrar/"
proxy="http://proxy:8080" proxyuser= "<username>" proxypassword="<password>"/>
```

2 在 < **探测器安装目录** >/etc/metrcis.config 文件中设置以下 **proxy** 属性，以便将系统度量配置为使用代理：

- 将 **proxy.uri** 设置为代理服务器 url。
- 将 **proxy.user** 设置为用于验证代理服务器的用户。
- 将 **proxy.password** 设置为用于验证代理服务器的密码。

3 重新启动所插桩的应用程序进程。

16

配置 Diagnostics 以在防火墙环境中工作

本章提供有关配置 HP Diagnostics 以使其在有防火墙的环境下正常工作的基本信息。当防火墙将探测器与其他 Diagnostics 组件或者 LoadRunner、Performance Center 或 Business Service Management 组件分隔开时，就需要进行该配置。有关更多详细信息，请参阅 LoadRunner、 Performance Center 和 Business Service Management 文档。

本章包括：

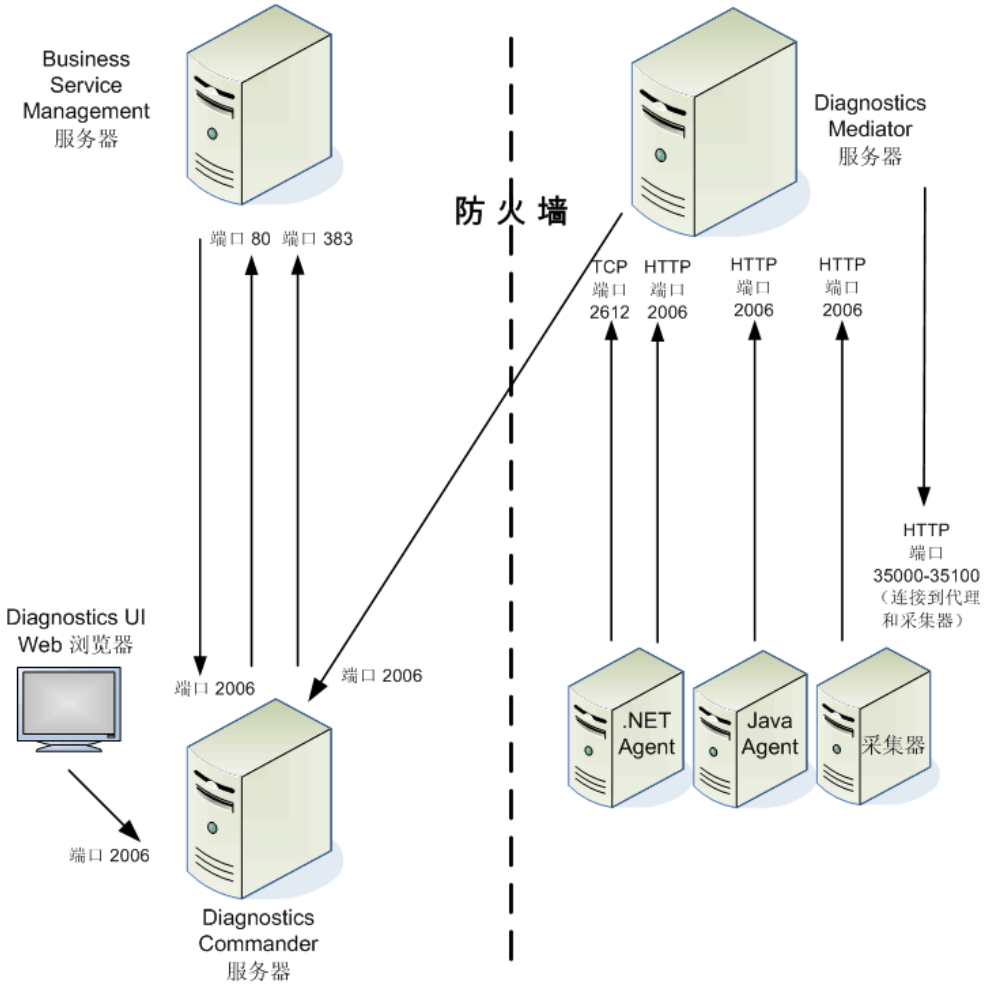
注意：应仅由深入了解 HP Diagnostics 且经验丰富的用户执行该配置。请谨慎修改 Diagnostics 组件的任何配置设置。

为 Diagnostics 配置防火墙的概述

根据组成 Diagnostics 集成环境的 HP 软件产品的不同，为 Diagnostics 配置防火墙的过程也不同。

Business Service Management

下图显示了一个典型的 Diagnostics 部署，其中，防火墙将探测器与其他 Diagnostics 和 Business Service Management 组件分隔开。

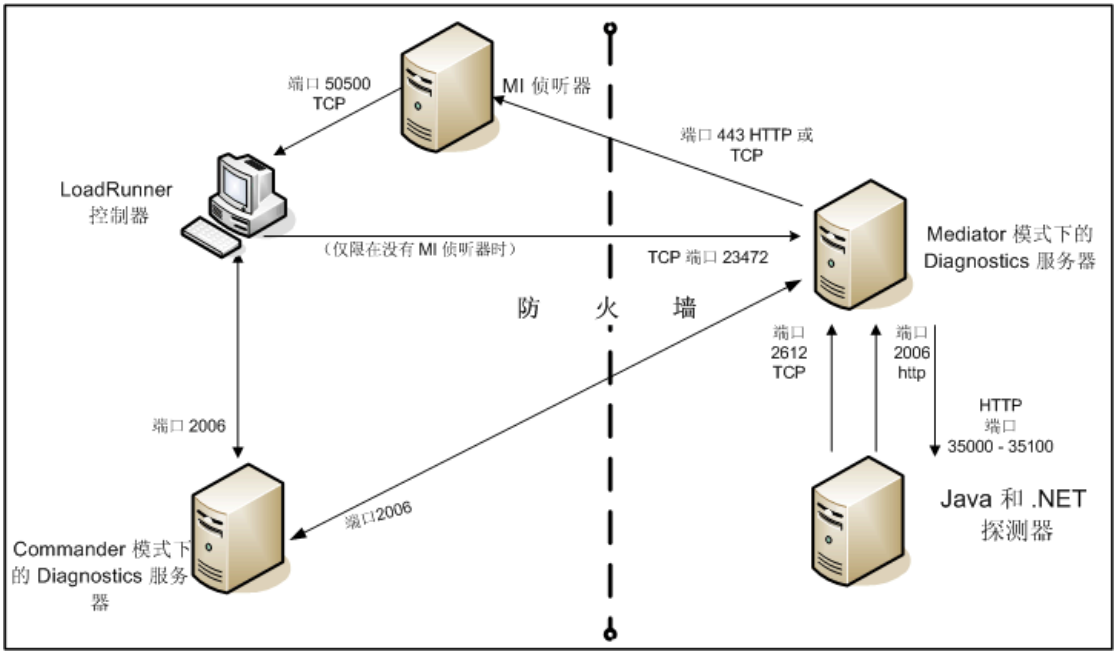


要配置防火墙以启用 Diagnostics 组件和 Business Service Management 之间的通信，请打开以下端口：

- 端口 2006，允许将 HTTP 请求从 Business Service Management 服务器发送到 Diagnostics Commander 服务器。
- 端口 80，允许将 HTTP 请求从 Diagnostics Commander 服务器发送到 Business Service Management 服务器。端口 383，允许从 Diagnostics Commander 服务器向 Business Service Management 进行 HI 更新。
- 端口 2006，允许将 HTTP 请求从 Diagnostics UI Web 浏览器客户端计算机发送到 Diagnostics Commander 服务器。请注意，您也可以从 Diagnostics UI 访问 Java 或 .NET Profiler。
- 端口 2006，允许将 HTTP 请求从 Diagnostics Mediator 服务器发送到 Diagnostics Commander 服务器。
- 端口 35000-35100，允许将 HTTP 请求从 Diagnostics Mediator 服务器发送到探测器。必须要允许其执行通信的实际端口取决于您在配置探测器时启用的端口号和所插桩的虚拟机数。有关设置探测器端口范围的信息，请参见“配置应用程序服务器上多个 Java 进程的监控”（第 231 页）。
- 端口 2612，允许将 TCP 请求从 .NET 代理发送到 Diagnostics Mediator 服务器。
- 端口 2006，允许将 HTTP 请求从 Java 代理、采集器、 .NET 度量采集器发送到 Diagnostics Mediator 服务器。

LoadRunner 和 Performance Center

下图显示了典型的 Diagnostics 拓扑，其中，防火墙将探测器与其他 Diagnostics 和 LoadRunner 组件分隔开。



注意：该图中以 LoadRunner 为例进行了说明，这些信息也适用于 Performance Center。

必须配置防火墙，以允许将 Diagnostics 组件进行相互通信。

要配置防火墙以启用 Diagnostics 组件之间的通信，请打开以下端口：

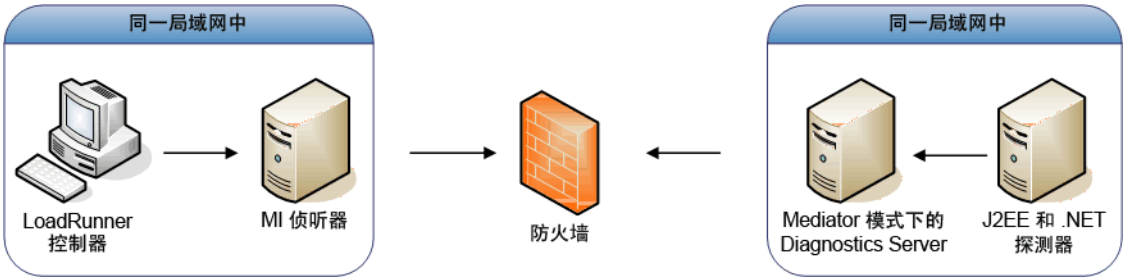
- 端口 2006，允许将 HTTP 请求从 Diagnostics Mediator 服务器发送到 Diagnostics Commander 服务器。
- 端口 2612，允许将 TCP 请求从探测器发送到 Diagnostics Mediator 服务器。
- 端口 2006，允许将 HTTP 请求从探测器发送到 Diagnostics Mediator 服务器。

注意：除了上面的拓扑以外，如果您使用 LoadRunner 分析工具查看脱机 J2EE 结果，请参见 “通过防火墙整理脱机分析文件”来正确配置控制器和 Mediator 模式下的 Diagnostics 服务器，以便于脱机文件检索。

通过防火墙整理脱机分析文件

在 LoadRunner/Performance Center 负载测试期间，接受探测器报告的 Diagnostics 服务器在其主机上会生成一个脱机分析文件。脱机分析文件由 LoadRunner/ Performance Center 在整理负载测试结果时检索。

如果参与负载测试的 LoadRunner/Performance Center 控制器与 Diagnostics 服务器之间存在防火墙，则必须将控制器和 Diagnostics 服务器配置为使用 MI 侦听器实用程序，以传输脱机分析文件。LoadRunner/Performance Center 附带了 MI 侦听器实用程序，应将它安装在防火墙内的计算机上，如下图所示。



要配置控制器以访问防火墙后的 Diagnostics 服务器，请参阅以下各节：

- 安装和配置 MI 侦听器。
- 配置 Diagnostics Mediator 服务器以使用防火墙。
- 配置 LoadRunner 和 Performance Center 以使用 Diagnostics 防火墙。

安装和配置 MI 侦听器

MI 侦听器组件是用来为防火墙外的负载生成器提供服务的组件。有关如何配置 LoadRunner 的 MI 侦听器的详细信息，请参见《HP LoadRunner Controller 用户指南》。有关如何配置 Performance Center 的 MI 侦听器的详细信息，请参阅《HP Performance Center Administrator's Guide》。

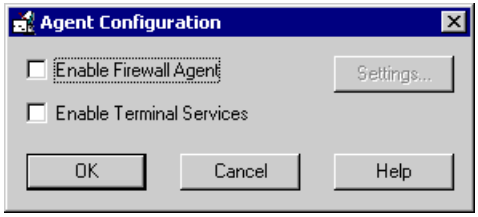
配置 Diagnostics Mediator 服务器以使用防火墙

要配置 Diagnostics Mediator 服务器，使它跨防火墙工作，必须完成以下附加配置步骤。如果尚未安装和配置 Diagnostics Mediator 服务器，则必须在尝试上述步骤之前完成。有关安装 Diagnostics Mediator 服务器的说明，请参见第 2 章，“安装 Diagnostics 服务器”。

要为 Windows 计算机上的防火墙配置 Diagnostics Mediator 服务器，请执行以下操作：

1 通过运行 **<Diagnostics 服务器安装目录 >/nanny/windows/bin/AgentsConfig.exe**，启动代理配置。

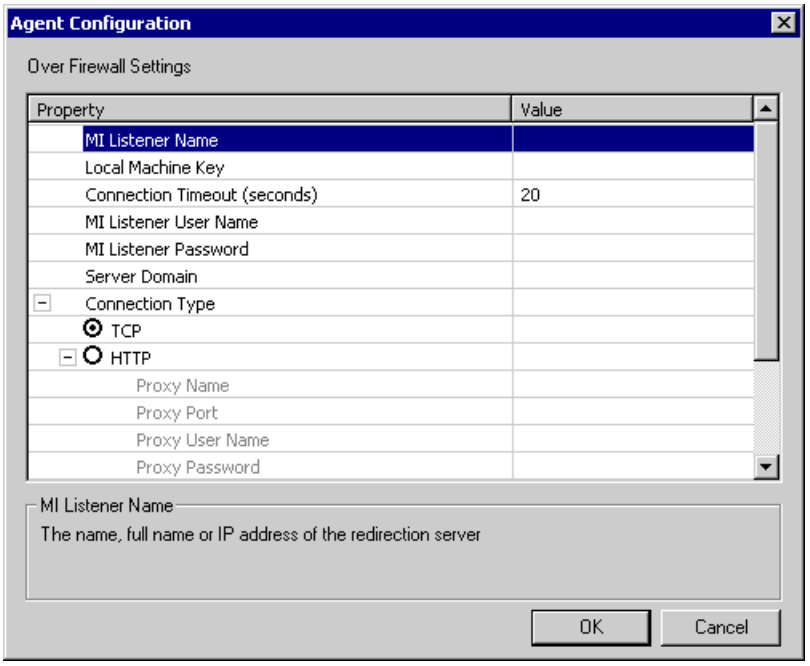
此时将打开 “Agent Configuration”对话框：



2 选择 “Enable Firewall Agent”，“Settings”按钮将启用。

3 单击 “Settings”，代理配置过程将打开 “代理配置设置”对话框。

4 在 “MI Listener Name” 属性的 “Value” 列中，输入装有 MI 侦听器的计算机的主机名或 IP 地址。



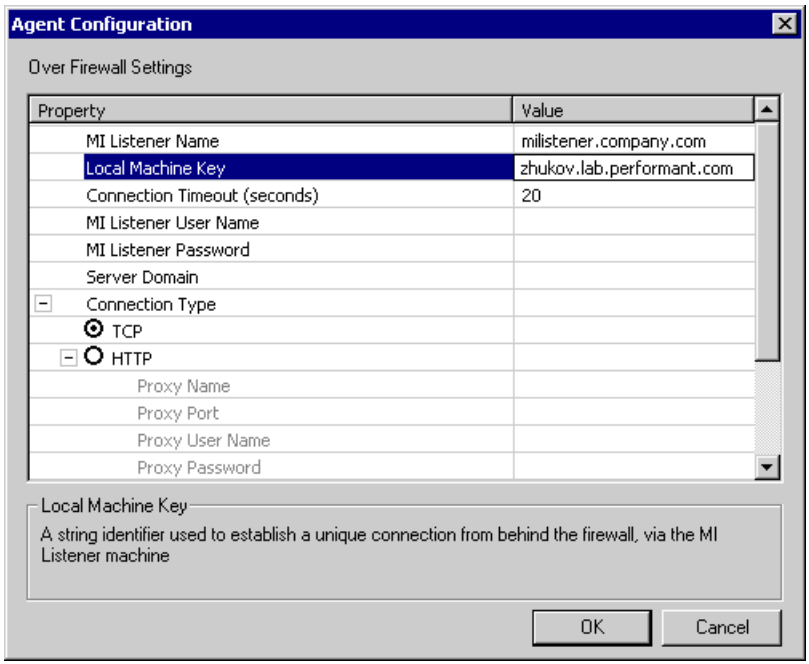
5 在 “Local Machine Key” 属性中，输入 Diagnostics Mediator 服务器主机的计算机名。

重要信息：

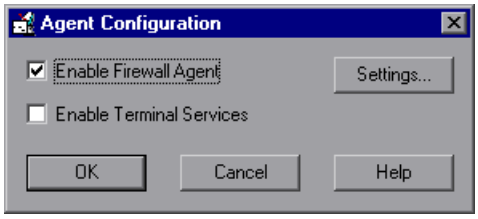
- 输入 Diagnostics 组件的主机名时，必须使用完全限定主机名，也就是计算机名和域名。
- 可使用 Diagnostics UI 中的 “系统运行状况” 视图来确定 Diagnostics Mediator 服务器主机的计算机名。有关系统视图的详细信息，请参见附录 D，“使用管理员的系统视图”。

---

6 单击 “OK”关闭对话框。



7 再次单击 “OK”，关闭 “Agent Configuration”对话框。



8 此时将打开 “Restart Agent”对话框，单击 “OK”重新启动代理。

要为 UNIX/Linux 计算机上的防火墙配置 Diagnostics Mediator 服务器，请执行以下操作：

1 修改 **<Diagnostics 服务器安装目录 >/nanny/< 平台 >/dat/br_lnch_server.cfg** 文件。

将 **FireWallServiceActive** 属性的值更改为 1。

2 运行以下命令，以启动代理配置实用程序。

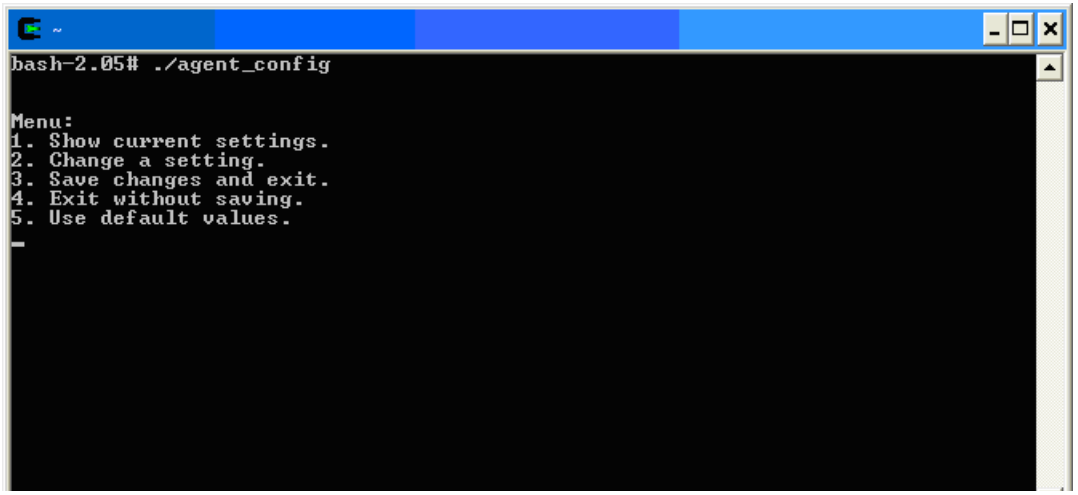
对于 Solaris 和 Linux， <platform> 可以是 Solaris 或 Linux：

```
export LD_LIBRARY_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/<platform>
cd $M_LROOT/bin
./agent_config
```

对于 HP-UX：

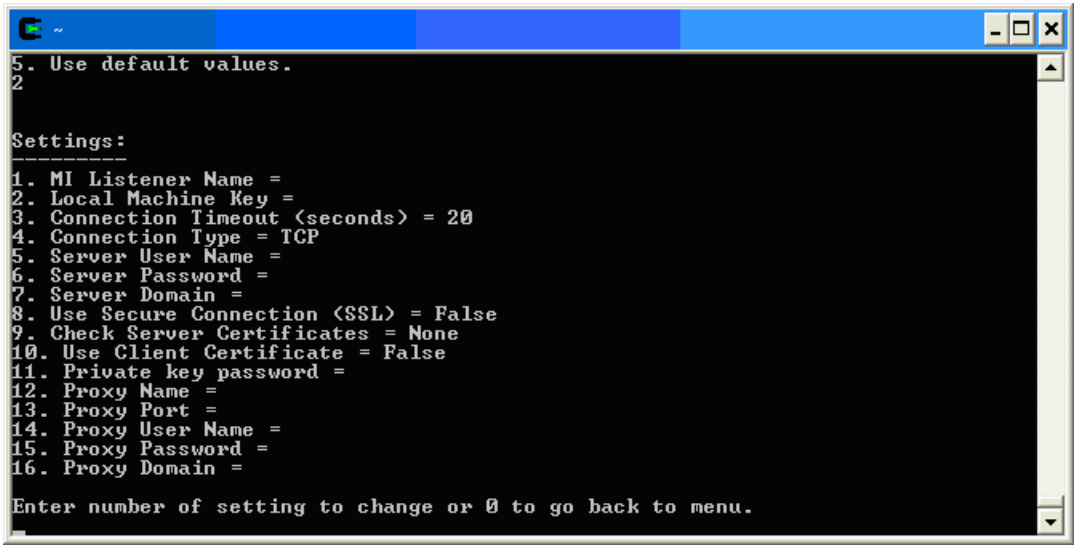
```
export SHLIB_PATH=.
export M_LROOT=<diagnostics_server_install_dir>/nanny/hpux
cd $M_LROOT/bin
./agent_config
```

3 在 “代理配置实用程序”窗口中，按 2 以选择 **Change a Setting**。



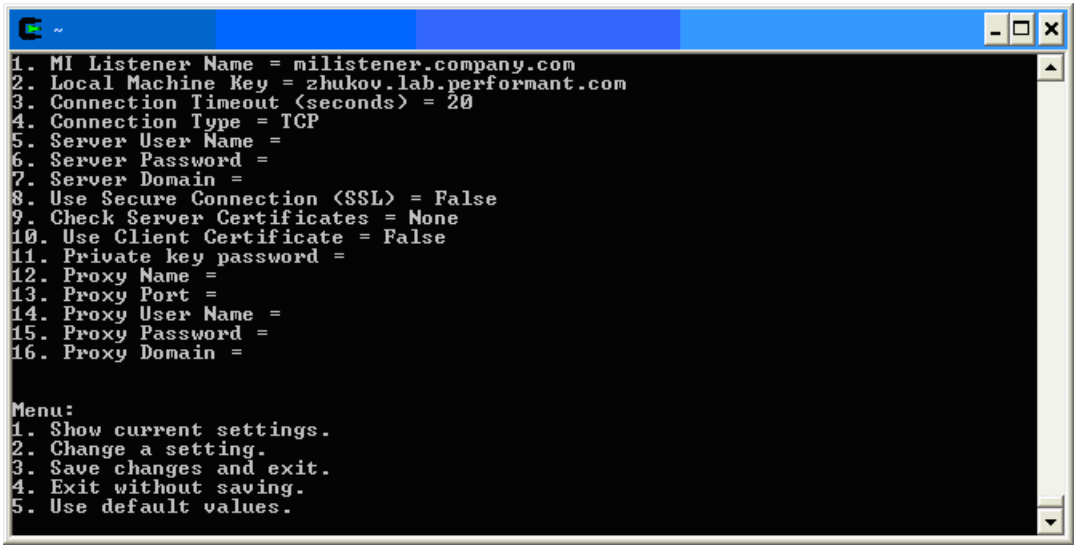
4 此时将显示设置列表。

按 1 以选择 **MI Listener Name**，然后输入 MI 侦听器主机的计算机名或 IP 地址。



5 按 2 以选择 **Change a Setting**，然后输入 Diagnostics Mediator 服务器主机的计算机名。

可使用 Diagnostics UI 中的 “系统运行状况” 视图来确定 Diagnostics Mediator 服务器主机的计算机名。有关系统视图的详细信息，请参见附录 D，“使用管理员的系统视图”。



6 按 3 以选择 **Save changes and exit**，完成更新。

7 重新启动 Diagnostics Mediator 服务器。

./m_daemon_setup -remove（此命令停止服务器和 MI 代理）

./m_daemon_setup -install（此命令启动服务器和 MI 代理）

在 Linux 中，如果遇到错误，则可能需要安装 libstdc++.so.5 共享库。

配置 LoadRunner 和 Performance Center 以使用 Diagnostics 防火墙

安装 MI 侦听器并配置 Mediator 计算机之后，必须在 LoadRunner/Performance Center 中更新 Diagnostics 配置，以便当应用程序从防火墙外的 Mediator 传输脱机数据时知道使用 MI 侦听器。

对于 Performance Center ：

确保指定 MI 侦听器计算机的 IP 地址，该计算机已配置为通过防火墙收集应用程序 Diagnostics 的数据。有关详细信息，请参阅 《HP Performance Center Administrator's Guide》中关于 MI 侦听器的部分。另外，还应确保在 Performance Center 中启用 Diagnostics。有关详细信息，请参阅 《Performance Center User's Guide》中关于 HP Diagnostics 的部分。

对于 LoadRunner：

确保在 LoadRunner 中启用 Diagnostics。配置负载测试场景以使用 Diagnostics 时，确保选择了通过防火墙进行工作的选项，并指定相关 MI 侦听器服务器的名称。有关详细信息，请参阅 《HP LoadRunner Controller 用户指南》中关于 HP Diagnostics 的部分。

第 VI 部分

配置 Diagnostics 度量采集器

本部分包括：

- .NET 系统度量代理 - 系统度量捕获
- Java 代理度量采集器
- Java 代理 - 系统度量捕获
- Java 代理 - JMX 度量捕获

17

.NET 系统度量代理 - 系统度量捕获

说明如何捕获系统度量，以及如何配置随 .NET 代理一同安装的系统度量采集器。

本章包括：

关于 .NET 系统度量代理

系统度量采集器随 .NET 代理一同安装，并作为一个 Windows 服务 (HP Diagnostics Metrics Agent) 运行。.NET 系统度量代理可从代理的主机上收集系统级别度量数据，如 CPU 使用率和内存使用率。可以对其进行配置，以便控制要收集的度量数据，以及度量数据的收集和发布方式。

无论在主机上启动了多少个探测器实例，都只能在给定主机上运行一个 .NET 系统度量代理实例。

注意：要配置 .NET 代理的其他探测器度量捕获（不是此处描述的系统度量捕获），请参阅 “收集其他探测器度量或修改探测器度量”（第 622 页）。

默认情况下的系统度量捕获

以下是 .NET 系统度量代理在默认情况下收集的适用于所有受支持平台（不包括 z/OS）的系统度量：

- CPU
- MemoryUsage
- VirtualMemoryUsage
- ContextSwitchesPerSec
- DiskBytesPerSec
- DiskIOPerSec
- NetworkBytesPerSec
- NetworkIOPerSec
- PageInsPerSec
- PageOutsPerSec

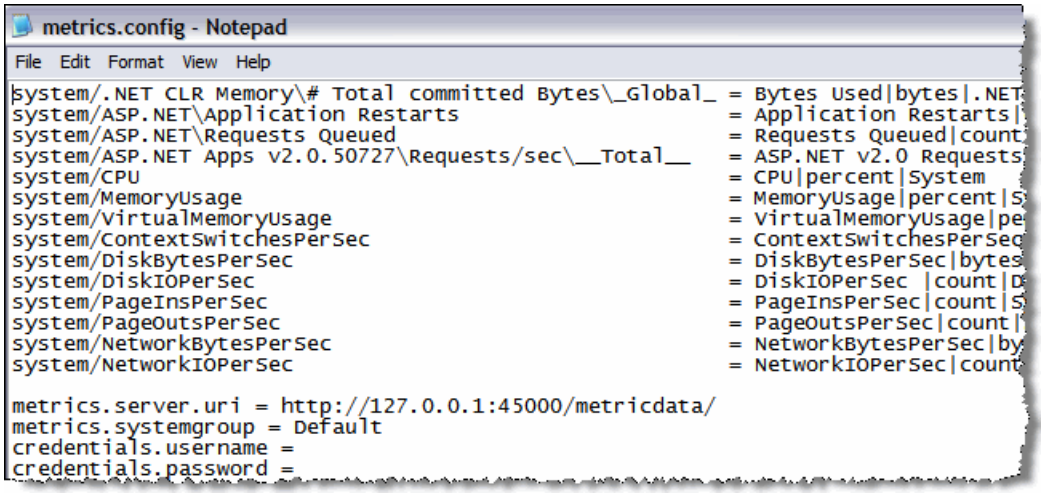
除了上述默认系统度量之外，默认情况下 .NET 代理系统还会捕获下列系统度量。（“了解 system/ 度量采集器条目”（第 650 页）中介绍了这些条目的布局）。

.NET CLR Memory\# Total committed Bytes_Global_
ASP.NET\Application Restarts
ASP.NET\Requests Queued
ASP.NET Apps v2.0.50727\Requests/sec__Total__

您可以控制 .NET 系统度量代理所收集的默认系统度量，并且可以使用 .NET 系统度量代理来捕获自定义的系统度量。

配置 .NET 系统度量捕获

.NET 系统度量代理的配置文件是 **< 探测器安装目录 >/etc/metrics.config** 文件。.NET 代理将动态地处理 metrics.config 文件更改。



注意：Java 代理包含有一个不同的 metrics.config 文件 （请参阅第 18 章，“Java 代理度量采集器”）。

了解 system/ 度量采集器条目

metrics.config 文件中的度量采集器条目可指示 .NET 系统度量代理收集特定度量。以 **system/** 开头的条目将作为 Windows 性能监控器计数器进行处理。

这些系统度量采集器条目使用以下格式：

```
system/<Counter_name>\<Performance_object>\<Instance>\<Remote_machine> =
<metric_id>|<metric_units>|<category_id>
```

除可选的 <Instance> 和 <Remote_machine> 字段以外，其他所有字段都是必需字段。

其中：

- **Counter_name** 表示 Windows 性能监控器计数器。有关如何在 Windows 性能监控器 UI 中识别计数器、性能对象和实例的详细信息，请参阅 “使用 Windows 性能监控器添加系统度量”（第 652 页）。
- **Performance_object** 表示与 Counter_name 相关的 Windows 性能监控器性能对象。
- **Instance** 表示计数器的 Windows 性能监控器实例。可以使用通配符 (*) 表示所需的所有实例。如果要为所有实例指定特定枚举，则可以在枚举索引编号前添加井号 (#1)。枚举索引编号必须是正数。
- 仅当运行 Windows 性能监控器计数器的计算机与运行 .NET 系统度量代理的计算机不相同 （远程）时，才需要 **Remote_machine**。使用此配置进行工作的最低要求是，运行 .NET 系统度量代理的计算机上的网络服务用户必须具有从远程计算机读取 Windows 性能监控器计数器的权限。

- **<metric_id>** 表示用于代表 Diagnostics UI 中的度量标准的名称。在 **metrics.config** 文件中， metric_id 必须唯一。如果 metric_id 的值与其中一个默认度量相同，则 Diagnostics 将用某个标准名称来替换条目中的 metric_id，此标准名称用于引用 UI 中的度量。如果 metric_id 的值与默认度量不同，则 metric_id 将用作 UI 中的度量名称 （与条目中显示的名称完全相同）。
- **<metric_units>** 表示用于报告度量的度量单位。此参数是必需的，并且必须包含下列度量单位之一：
  - 微秒、毫秒、秒、分钟、小时、天
  - 字节、千字节、兆字节、千兆字节
  - 百分比、分数百分比
  - 计数
  - 负载
- **<category_id>** 可将一组度量放在 Diagnostics UI 的 “详细信息”窗格中的同一个标题下。此参数对在 Diagnostics 视图中显示的数据没有影响。

不含 <Instance> 的示例：

```
system/ASP.NET\Requests Queued = Requests Queued|count|ASP
```

含有 <Instance> 的示例：

```
system/Processor\% Processor Time\_Total = CPU|percent|System
```

含有整型 <Instance> 的示例：

```
system/Processor\% Processor Time\#1 = CPU 1|percent|System
```

不含 <Instance> 且在 <Remote_Machine> 上运行的示例：

```
system/ASP.NET\Requests Queued\\IISAQUAH = Requests
Queued(IISAQUAH)|count|ASP
```

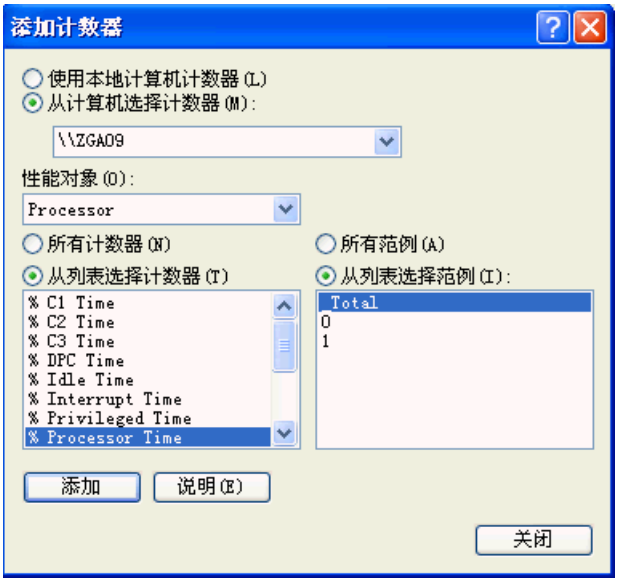
使用 Windows 性能监控器添加系统度量

要将系统度量计数器添加到 metrics.config 文件中，必须先使用 Windows 性能监控器 (Perfmon) 找到其定义。以下示例将使用版本为 5.x 的 Perfmon。版本 6.x 与版本 5.x 类似，但 UI 稍有不同。

要在 Perfmon 中添加计数器，请执行以下步骤：

1. 启动 Windows 性能监控器。例如，选择“开始”>“控制面板”>“管理工具”>“性能”。
2. 此时将打开“Perfmon 性能”对话框，其中将显示“系统监控器”图形，在图形下方是当前计数器的表格。右键单击“系统监控器”图形，然后从弹出菜单中选择“添加计数器 ...”。

   “添加计数器”对话框显示如下：



3. 选择“从计算机中选择计数器”条目，并确保从下拉列表中选择主机。

4 在 “性能对象”列表中选择计数器所属的对象。

5 选择 “从列表中选择计数器”，然后从实例列表中选择一个实例。

6 单击 “添加”按钮，添加计数器。下面将说明如何使用 system/ 度量条目来创建计数器的条目，如 “了解 system/ 度量采集器条目”（第 650 页）所述。

要收集 Perfmon 计数器的度量数据，请执行以下操作：

1 在 .NET 代理系统中打开 **< 探测器安装目录 >/etc/metrics.config** 文件。

2 使用 “了解 system/ 度量采集器条目”（第 650 页）中所述的布局为计数器创建 **system/** 度量。

可以在文件的任何地方添加此条目，但最好在这些类型条目的现有集合底部添加此条目。在以上屏幕截图所示的示例中：

- 选择的主机是 ROS59524ART
- 选择的性能对象是 Processor
- 选择的计数器是 % Processor Time
- 选择的实例是 _Total

因此，如果主机计算机是本地计算机，则性能监控器计数器的 metrics.config 文件中的条目为：

```
system/Processor\% Processor Time\_Total = CPU|percent|System
```

如果主机计算机是远程计算机，则性能监控器计数器的 metrics.config 文件中的条目为：

```
system/Processor\% Processor Time\_Total\ROS59524ART =
CPU(ROS59524ART)|percent|System
```

性能计数器的安全性

.NET 度量代理使用性能计数器来收集系统度量。度量代理作为一个网络服务运行，并且需要将此帐户添加到 “Performance Monitor Users”组中。

有关所添加的系统度量计数器的疑难解答

如果指定了无法正常工作的新计数器，则可以使用 Windows 事件查看器查看 .NET 系统度量代理源的 Diagnostics 日志，以查看是否存在错误和警告。

例如：

“找不到具有指定类别名称的性能计数器”警告条目通常表示可能键入了错误的计数器名称。例如，如果从有空白部分的“PerfMon 性能”窗格读取计数器名称，就可能会发生上述情况。 PerfMon 使用的默认字体不是等宽字体，因此很难发现计数器、类别和实例名称中的空白部分。可以将字体改为等宽字体类型，这样就能更清楚地看到计数器名称的确切格式。

例如：

“指定类别中不存在实例”警告条目通常表示所选实例当前未处于活动状态。建议您不要使用瞬态实例，类似于 __Total__ 的永久实例比较合适。

.NET 代理 metrics.config 文件中的默认条目

在安装时， **< 探测器安装目录 >/etc/metrics.config** 文件中包含三种条目：

- PerfMon 计数器的一组默认 **system/** 条目
- 一个 **metrics.server.uri** 条目，用于指定 .NET 系统度量代理发布数据的方式
- 一个默认的 **metrics.systemgroup** 条目

可以在这些默认条目后面添加其他条目。

metrics.config 文件中的关键字

可以在 < **探测器安装目录** >/**etc/metrics.config** 文件的条目中使用的关键字包括：

- credentials.password
- credentials.username
- default.sampling.rate
- metrics.server.uri
- metrics.systemgroup
- metrics.agent.publish.interval
- metrics.agent.registered_hostname
- proxy.password
- proxy.user
- proxy.uri
- system/

“配置 .NET 系统度量捕获”（第 649 页）中介绍了关键字 **system/** 的用法。

以下部分描述了其他关键字的用法。

credentials.password	此设置必须与 probe_config.xml 文件中 <credentials> 元素的 password 属性设置相匹配。有关更多详细信息，请参阅 “<credentials> 元素”（第 514 页）。
credentials.username	此设置必须与 probe_config.xml 文件中 <credentials> 元素的 username 属性设置相匹配。有关更多详细信息，请参阅 “<credentials> 元素”（第 514 页）。
default.sampling.rate	此设置定义了 .NET 系统度量代理对已配置的系统度量计数器进行采样的速率。默认速率是每 5 秒一次。值可表示为秒数、分钟数、小时数或天数，例如 nS、nM、nH 或 nD。以下示例中，速率设置为每 10 秒一次： default.sampling.rate = 10s

metrics.server.uri	此设置将在安装时自动生成。此设置定义了 .NET 系统度量代理用于将系统度量计数器发布到 Diagnostic Mediator Server 的 URI。 在以下示例中，Diagnostic Mediator Server 在 my_diag_server 计算机上运行，并且使用 metricport 2006 来发布度量： metrics.server.uri = http://<my_diag_server>:2006/metricdata/?sleep=false 对 <mediator> 元素的 metrichost 属性或 metricport 属性的 probe_config.xml 设置的任何更改也必须同时在 metrics.server.uri 设置中反映出来。 ?sleep 设置用于控制接收到已发布度量的 Diagnostic Mediator Server 是立即响应 .NET 系统度量代理，还是延迟响应。设置为 ?sleep=false 表示立即响应，设置为 ?sleep=true 表示延迟响应，默认延迟时间为 5 秒。
metrics.systemgroup	此设置将在安装时自动生成。请不要更改此设置。
metrics.agent.publish.interval	此设置定义 .NET 系统度量代理向 Diagnostic Mediator Server 发布系统度量计数器当前值的时间间隔。默认间隔为 5 秒。可以将设置的值表示为秒数或分钟数，例如 nS 或 nM。以下示例中，发布间隔设置为每 10 秒一次： metrics.agent.publish.interval = 10S
metrics.agent.registered_hostname	有关此设置的使用时间和使用方式的说明，请参阅 “覆盖默认探测器主机名”（第 612 页）。
proxy.password	此设置必须与 probe_config.xml 文件中 <diagnosticsserver> 元素的 proxypassword 属性设置相匹配。有关更多详细信息，请参阅 “<diagnosticsserver> 元素”（第 518 页）。另请参阅第 15 章，“针对 HTTP 代理配置 Diagnostics 服务器和代理”。

proxy.user	此设置必须与 probe_config.xml 文件中 <diagnosticsserver> 元素的 proxyuser 属性设置相匹配。有关更多详细信息，请参阅 “<diagnosticsserver> 元素”（第 518 页）。另请参阅 第 15 章，“针对 HTTP 代理配置 Diagnostics 服务器和代理”。
proxy.uri	此设置必须与 probe_config.xml 文件中 <diagnosticsserver> 元素的 proxy 属性设置相匹配。有关更多详细信息，请参阅 “<diagnosticsserver> 元素”（第 518 页）。另请参阅 第 15 章，“针对 HTTP 代理配置 Diagnostics 服务器和代理”。

18

Java 代理度量采集器

本部分介绍 Java 代理度量捕获以及如何配置度量采集器。

本章包括：

关于度量捕获

您可以借助 Java 代理，通过修改度量配置文件 **< 探测器安装目录 >/etc/metrics.config** 中的条目来配置度量采集器。

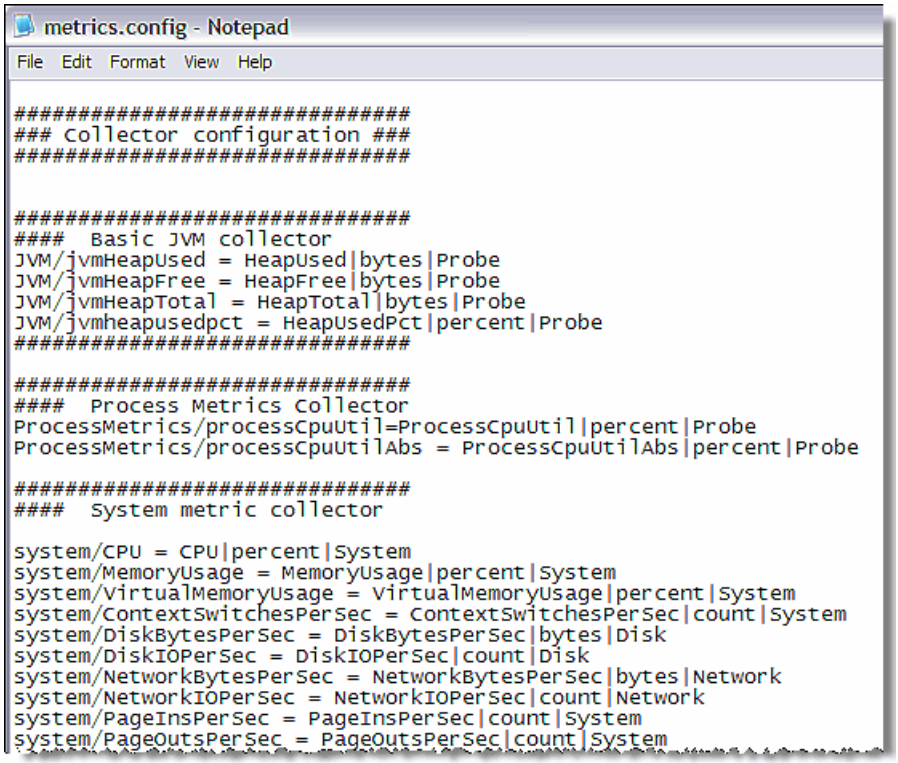
注意：.NET 代理包含一个不同的 metrics.config 文件（请参阅“.NET 系统度量代理 - 系统度量捕获”（第 647 页））。

度量配置文件中定义了用于代理安装的系统和 JMX 度量采集器。您可以通过度量配置文件 **< 探测器安装目录 >/etc/metrics.config** 中的属性和条目来控制度量采集器。

注意：如果更新度量配置文件，度量采集器会自动重新启动，以使更改生效。

Java 代理收集的度量类型

您可以在 **metrics.config** 文件中查看 Java 代理收集的度量。



列出可用度量

Java 代理的 metrics.config 文件可以将每个 JMX 采集器的所有可用度量的列表写入文件中。当 metrics.config 文件中的 **default.dump.available.metrics** 属性设置为 True 时，探测器会将此可用度量列表写入探测器日志目录下的文本文件中。这些文件的命名格式如下：< 探测器安装目录 >/log/< 探测器 ID>/jmx_metrics_< 收集器名称 >.txt。有关如何以此信息为模板配置其他度量捕获的详细信息和示例，请参阅“获取可用的 JMX 或 WebSphere PMI 度量列表”（第 681 页）。

了解度量采集器条目

度量采集器条目指示 Java 代理度量采集器收集特定度量。条目左侧的参数控制探测器从主机或 JVM 收集度量的方式，而条目右侧的参数则定义所收集的度量在 Diagnostics 中的处理方式以及在用户界面中的显示方式。

这些条目可以使用下列格式之一：

```
<collector_name>/<metric_config>=<metric_id>|<metric_units>|<category_id>
```

或

```
<collector_name>/<metric_config>=
RATE<rate_multiplier>(<metric_id>|<metric_units>|<category_id>)
```

其中：

- **<collector_name>** 表示 Diagnostics 度量采集器的名称。这些采集器在 **metrics.config** 中进行定义。

  对于系统度量，此参数的值为 system。对于 JMX 度量，此参数的值通常定义为应用程序服务器类型名称和版本，如 WebSphere5。

  在 Diagnostics UI（http://<diagnostics 服务器 >:2006/query）的“高级查询”页面同样也可找到采集器名称及其度量名称。

- **<metric_config>** 标识要在主机系统或应用程序服务器的 JVM 上进行监控的度量。此参数的格式会有所差异，具体取决于要创建系统度量条目还是 JMX 度量条目。有关格式化系统度量采集器的 **metric_config** 属性的信息，请参阅“捕获其他自定义系统度量”（第 671 页）。有关格式化 JMX 度量的 **metric_config** 属性的信息，请参阅 “创建新的 JMX 或 WebSphere PMI 度量条目”（第 684 页）。

- **RATE(...)** 表示度量值在采样期间转换为速率 （单位数 / 秒）。

例如，如果将 **Rate** 参数用于度量**启动后的总 servlet 请求数**，则收集的度量值将从 **servlet 请求数**转换成 **servlet 请求数 / 秒**。

如果不使用 **Rate**，则省略括号，如上面第一个示例所示。

注意：此参数仅适用于包含非递减值的度量。

- **<rate_multiplier>** 是可选参数，用于指示通过用 **<rate_multiplier>** 乘以速率，来调整速率。

  例如，如果将 **Rate** 参数和 **rate_multiplier** 用于度量**总 gc 时间** （以毫秒为单位），则收集的度量值将从**总 gc 时间**转换为 **gc 所用时间 %**。

- **<metric_id>** 指示在 UI 中表示度量的名称。在 **metrics.config** 文件中，metric_id 必须唯一。如果 metric_id 的值与其中一个默认度量相同，则 Diagnostics 将用某个标准名称来替换条目中的 metric_id，此标准名称用于引用 UI 中的度量。如果 metric_id 的值与默认度量不同，则 metric_id 将用作 UI 中的度量名称 （与条目中显示的名称完全相同）。
- **<metric_units>** 表示用于报告度量的度量单位。此参数是必需的，并且必须包含下列度量单位之一：
  - 微秒、毫秒、秒、分钟、小时、天
  - 字节、千字节、兆字节、千兆字节
  - 百分比、分数百分比
  - 计数
  - 负载

➤ **<category_id>** 可将一组度量组织在一起，放在 Java Diagnostics Profiler 中“度量”选项卡侧栏树中的同一标题下。此参数对 Diagnostics UI 视图“详细信息”窗格中显示的数据没有影响。

注意：创建度量采集器条目后，请在反斜线“\”、空格“ ”或冒号“:”前添加转义符“\”。此要求适用于从文件加载的 Java 属性。

关于收集其他探测器度量

要收集其他度量的信息，请使用 “了解度量采集器条目”（第 662 页）中描述的语法，将度量条目添加到 **metrics.config** 文件的相应度量采集器中。

有关捕获其他系统度量的详细信息，请参阅 “捕获其他自定义系统度量”（第 671 页）。

有关捕获其他 JMX 度量的详细信息，请参阅 “其他自定义 JMX 度量”（第 681 页）。

修改已捕获的探测器度量

您可以在 **metrics.config** 文件的度量采集器中更新默认和自定义度量条目。

停止捕获度量

要使度量采集器停止收集 **metrics.config** 中列出的度量，可以删除度量条目，或通过在开头添加“#”，使度量条目成为注释行。

为系统上的多个 JVM 应用程序使用自定义 metrics.config 文件

有时仅需收集某些度量，或为运行在具有多个 JVM 的系统上的选定 JVM 应用程序自定义度量采集器属性，这样的更改可能会对运行在此系统上的其他已插桩 JVM 产生负面影响。在这些情况下，您可以创建和自定义其他 **metrics.config** 配置文件，并配置这些 JVM 应用程序以使用自定义设置，操作步骤如下：

注意：您只需配置需自定义 **metrics.config** 文件的 JVM 应用程序。其他 JVM 应用程序可使用现有的 **metrics.config** 配置。

---

1 为每个要求特殊自定义的 JVM 应用程序复制 **etc/metrics.config** 文件，并命名此文件，如 **metrics_< 应用程序名称 >.config**。此文件必须与原始的 **metrics.config** 文件在同一个 **< 探测器安装目录 >/etc** 文件夹中。按照需要自定义此文件。

2 为每个已创建的 **metrics_< 应用程序名称 >.config** 文件创建 **lib/modules.properties** 文件的副本，并命名此文件，如 **modules_< 应用程序名称 >.properties**。此文件必须与原始的 **modules.properties** 文件在同一个 **< 探测器安装目录 >/lib** 文件夹中。

更改此新文件的 **metrics.properties** 属性，以指向新的 **metrics_< 应用程序名称 >.config** 文件，如下例所示：

```
################################################################################
## Metrics capture module
################################################################################
metrics.class.name=com.mercury.diagnostics.capture.metrics.MetricsModule
metrics.class.loader=probeLoader
metrics.properties=metrics_<app_name>.config
```

3 通过将以下命令添加到 JVM 属性定义中，可更新每个需要自定义度量收集的 JVM 启动脚本，以使用新对应的 **lib/modules_< 应用程序名称 >.properties** 文件：

-Dmodules.properties.file=module_< 应用程序名称 >.properties

19

Java 代理 - 系统度量捕获

介绍有关系统度量捕获过程的信息，以及如何配置 Java 代理系统度量采集器以捕获系统度量。

本章包括：

关于系统度量

系统度量采集器随 Java 代理一起安装。系统度量采集器可从代理的主机收集系统级别度量，如 CPU 使用情况和内存使用情况。您可以通过配置系统度量采集器来控制要收集的系统度量类型。

无论主机上启动了多少探测器实例，对于给定的主机，只能运行一个系统度量采集器实例。启动探测器实例时，它会尝试连接到在度量属性中指定的 UDP 端口。如果建立连接，则会启动系统度量采集器实例。如果无法建立连接，则其他探测器实例已经在主机上启动了系统度量采集器实例，因此新实例无法启动。

每个探测器会定期尝试连接到此端口，以确保系统度量采集器始终运行。如果启动系统度量采集器的探测器停止，则此探测器的另一个实例发现端口可用时，它会启动系统度量采集器的新实例。

默认情况下捕获的系统度量

下面是默认情况下度量采集器收集的适用于所有受支持平台（不包括 z/OS）的系统度量：

- CPU
- MemoryUsage
- VirtualMemoryUsage
- ContextSwitchesPerSec
- DiskBytesPerSec
- DiskIOPerSec
- NetworkBytesPerSec
- NetworkIOPerSec
- PageInsPerSec
- PageOutsPerSec

您可以控制系统度量采集器收集的默认系统度量的类型，还可以添加其他特定于平台的度量，以便采集器收集这些特定度量的信息。有关详细信息，请参阅 “配置系统度量采集器”（第 669 页）。对于 Windows、 Solaris 和 Linux 等特定平台，您可以创建由系统度量采集器收集的自定义系统度量。有关详细信息，请参阅 “捕获其他自定义系统度量”（第 671 页）。

有关 z/OS 系统度量的信息，请参阅 “启用 z/OS 系统度量捕获”（第 677 页）。

配置系统度量采集器

通过修改度量配置文件 < **探测器安装目录** >/etc/metrics.config 中的条目，可以将系统度量捕获流程配置为在您的环境中运行，并收集和报告您感兴趣的系统度量。有关度量采集器的常规信息，请参阅第 18 章，“Java 代理度量采集器”；有关度量采集器条目和语法的说明，请参阅“了解度量采集器条目”（第 662 页）。

注意：.NET 代理包含一个不同的 metrics.config 文件 （请参阅 “.NET 系统度量代理 - 系统度量捕获”（第 647 页））。

注意：如果更新度量配置文件，系统度量采集器会自动重新启动，以使更改生效。

系统度量采集器条目示例

以下示例介绍了如何创建适用于系统度量的度量采集器条目。要在主机平台上创建名为 CPU 的系统度量条目，请输入以下内容：

```
system/CPU = CPU|percent
```

其中：

- **system** 表示度量由系统度量采集器收集
- 第一个 **CPU** 表示在平台上称为 **CPU** 的度量正在受到监控
- 第二个 **CPU** 是 UI 中用于标记度量的名称
- **percent** 表示在主机上测量度量以及在 UI 中报告度量时，所采用的单位

修改默认端口

度量采集器的默认端口是 **35000**。如果代理主机配置为需要使用其他端口，则可以通过 **system.udp.port** 属性修改此值。

要修改默认端口，请执行以下操作：

1. 在 **metrics.config** 中找到 **system.udp.port** 属性。
2. 将 **system.udp.port** 属性的值更改为系统度量采集器要使用的端口号。默认端口是 **35000**。

注意：分配到系统度量采集器的端口与代理的 Web 服务器端口无关。

禁用系统度量收集

要禁用系统度量收集，从而 UI 中不在收集或显示他们，请将 **system.udp.port** 属性的值设置为 -1。

捕获其他自定义系统度量

通过 Java 代理系统度量采集器，可以在 Windows、Solaris 和 Linux 平台上捕获自定义系统度量。

以下各部分介绍如何捕获度量以及如何更新系统度量采集器中的条目，从而监控自定义度量。

本节包括：

- 在 Windows 主机上捕获自定义系统度量
- 在 Solaris 主机上捕获自定义系统度量
- 在 Linux 主机上捕获自定义系统度量

在 Windows 主机上捕获自定义系统度量

通过使用 Windows 系统监视器功能，可以添加计数器来表示系统或服务在某些特定方面的性能。Windows 系统监视器可跟踪和报告计数器；同时，Java 代理系统度量采集器可以监控计数器。

要使用 Windows 系统监视器添加计数器，请执行以下操作：

1. 启动 Windows 性能监控器：

   a 从 “开始” 菜单选择 “开始” > “运行”。

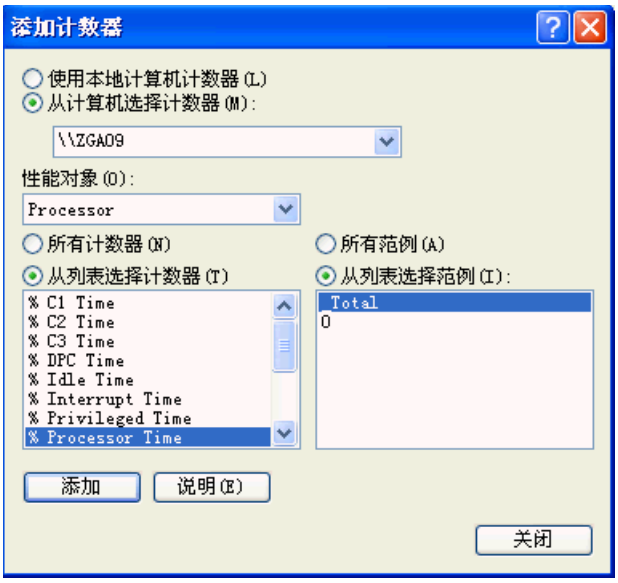
   b 在 “运行” 对话框的 “打开” 框中键入 perfmon。

   此时将打开 “性能” 对话框，显示 “系统监视器” 图形，图形下方为当前计数器的表。

2. 显示 “添加计数器” 对话框：

   右键单击 “系统监视器” 图形，然后从弹出的菜单中选择 “添加计数器 ...”。

Windows 将显示 “添加计数器”对话框：



3 确保从 “从计算机选择计数器”列表选择主机计算机。

4 在 “性能对象”列表中选择计数器所属的对象。

5 选择 “从列表选择计数器”，然后从后续计数器列表中选择计数器。

6 选择 “从列表选择范例”，然后从后续实例列表中选择实例。

7 单击 “添加”。

将计数器添加到系统监视器后，可以将系统度量采集器配置为收集计数器的度量。下面的说明将指导您基于以下模板创建 **metrics.config** 条目：

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

“了解度量采集器条目”（第 662 页）中对此模板进行了描述。

要收集适用于 Windows 系统监视器计数器的度量，请执行以下操作：

1 打开 < **探测器安装目录** >/etc/metrics.config。

2 使用以下模板创建条目的 < 度量配置 > 部分，键入计数器条目：

```
\<performance_object>(<instance>)\<counter>
```

在以上屏幕截图所示的示例中：

- 选择的性能对象是 %Processor
- 选择的实例是 _Total
- 选择的计数器是 Processor Time

为此示例创建的条目的 <metric_config> 部分为：

```
\Processor(_Total)\% Processor Time
```

3 填写系统度量条目模板其余部分，如以下示例所示：

```
system/\Processor(_Total)\% Processor Time = ProcessorTime|percent
```

4 通过在初始条目的每个反斜线 “\”、空格 “ ” 或冒号 “:” 之前添加反斜线 “\”，格式化初始条目。

执行此步骤后，上一个步骤中的初始条目会变为：

```
system/\\Processor(_Total)\\%\ Processor\ Time = ProcessorTime|percent
```

这是 **metrics.config** 的正确格式化条目，从而系统度量采集器可收集适用于 Windows 系统监视器计数器的度量。

```
system/\\\\RemoteMachine\\Processor(_TOTAL)\\%\ Processor\ Time=
Processor\ Time(Remote Machine)|percent
```

注意：假定已在远程计算机上正确设置 **perfmon**，则通过在性能对象名称前添加 **\\MachineName**，可以使用系统度量采集器从远程计算机上获取度量，如以下示例所示：

```
system/\\\\RemoteMachine\\Processor(_TOTAL)\\%\ Processor\
Time=Processor\ Time(Remote Machine)|percent
```

在 Solaris 主机上捕获自定义系统度量

通过使用 **kstat** 命令，可以找到可由系统度量采集器监控的 Solaris 系统度量。系统度量采集器仅可监控一部分通过 **kstat** 命令找到的度量。

要收集适用于 Solaris 系统度量的度量，请执行以下操作：

1 执行 **ksat** 命令，并标识要监控的度量。

Solaris 系统度量的格式如下：

```
module:instance:name:statistic
```

以下是一个示例：

```
vmem:35:ptms_minor:free
```

2 要使度量采集器收集更多系统度量标准的度量数据，请使用以下模板，将度量条目添加到 **metrics.config** 文件中的系统度量采集器中：

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

“了解度量采集器条目”（第 662 页）中对此模板进行了描述。

如果使用此模板，则上一步骤中的示例最初会显示如下：

```
system/vmem:35:ptms_minor:free = Virtual Memory (35) Free | count
```

3 通过在每个反斜线 “\”、空格 “ ” 或冒号 “:” 之前添加反斜线 “\”，格式化初始条目。

执行此步骤后，上一个步骤中的初始条目会变为：

```
system/vmem\:35\:ptms_minor\:free = Virtual\ Memory\ (35)\ Free | count
```

这是 **metrics.config** 的正确格式化条目，从而系统度量采集器可收集适用于 Solaris 系统度量的度量。

在 Linux 主机上捕获自定义系统度量

在 **/proc** 文件系统中可以找到可由系统度量采集器监控的 Linux 系统度量。要配置系统度量采集器以收集自定义 Linux 度量，请扫描 **/proc** 文件系统，找到所需度量，然后根据度量信息的位置在 **metrics.config** 中为度量创建系统度量采集器条目。

要收集 Linux 系统度量的度量，请执行以下操作：

1 扫描 **/proc** 文件系统，找到 Diagnostics 系统度量采集器要监控的度量。

要在 Linux 度量的 **metrics.config** 中创建系统度量配置条目，必须明确地指定系统度量值的位置。可以通过以下值指定位置：

- **文件名。**度量信息所在的文件的名称，包括 **/proc** 目录路径。
- **行偏移。**系统度量所在的行在文件中的行数计数。第一行计数为行 0。
- **字符偏移。**字符数的计数，度量值是文件中行内偏移量。行内的第一个字符计数为行 0。指定的偏移值必须是无符号的整数。

例如，如果希望系统度量采集器监控 SwapFree 系统度量，以便在 Diagnostics 视图中查看此度量，请扫描 **/proc** 目录以查找此度量，您会发现此度量位于 **meminfo** 文件中。此文件的布局如下：

```
MemTotal: 515548 kB
MemFree: 1552 kB
Buffers: 41616 kB
Cached: 152084 kB
SwapCached: 46064 kB
Active: 402720 kB
Inactive: 75328 kB
HighTotal: 0 kB
HighFree: 0 kB
LowTotal: 515548 kB
LowFree: 1552 kB
SwapTotal: 1048568 kB
SwapFree: 779192 kB
Dirty: 4544 kB
Writeback: 0 kB
Mapped: 300056 kB
Slab: 28764 kB
Committed_AS: 801364 kB
PageTables: 3184 kB
VmallocTotal: 499704 kB
VmallocUsed: 2184 kB
VmallocChunk: 497324 kB
HugePages_Total: 0
HugePages_Free: 0
Hugepagesize: 4096 kB
```

此文件中的 SwapFree 度量的位置会导致以下值：

- **文件名：**meminfo
- **行偏移：**12
- **字符偏移：**1

2 要收集其他系统度量的度量，请使用以下模板将度量的条目添加到 **metrics.config** 文件中的采集器度量中：

```
<collector_name>/<line>:<word>:<file>= <metric_id>|<metric_units>
```

此模板是 “了解度量采集器条目”（第 662 页）中描述的模板版本。**<metric_config>** 属性替换为 **<line>:<word>:<file>** 属性。

如果使用此模板，则上一步骤中的示例最初会显示如下：

```
system/12:1:meminfo = Swap Free | kilobytes
```

3 通过在每个反斜线 “\”、空格 “ ” 或冒号 “:” 之前添加反斜线 “\”，格式化初始条目。

执行此步骤后，上一个步骤中的初始条目会变为：

```
system/12\:1\:meminfo = Swap\ Free | kilobytes
```

这是 **metrics.config** 的正确格式化条目，从而系统度量采集器可收集适用于 Solaris 系统度量的度量。

启用 z/OS 系统度量捕获

可以为 z/OS 平台收集以下系统度量：

- CPU
- DiskIOPerSec
- DiskBytesPerSec

默认情况下不捕获系统度量，因为这需要更改某些系统配置。必须执行以下配置步骤，才能启用 z/OS 系统度量捕获。

要启用 z/OS 系统度量捕获，请执行以下操作：

1. 更改 **< 探测器安装目录 >/bin/** 目录权限，以允许递归执行。可以使用以下命令完成此操作：

```
chmod -R 770...
```

2. 更改 **< 探测器安装目录 >/bin/390-zos/systemmetrics** 目录权限，以允许执行。可以使用以下命令完成此操作：

```
chmod -R 0+x ...
```

3. 启动 RMF 监控器 III，并确保 SMF 记录 70 - 79 开始收集。
4. 在一个或多个 sysplex 系统上启动 RMF 数据缓冲。
5. 查看传递到 ERBDSQRY 服务的系统名称列表。
6. 确保系统开始收集包含子类型 5 的 SMF 记录 92。

20

Java 代理 - JMX 度量捕获

本章介绍捕获 JMX 度量的过程，以及如何配置 Java 代理度量采集器以捕获这些度量。

本章包括：

关于 JMX 度量

Java 代理附带有预定义的 JMX 度量采集器，这些采集器可通过以下应用程序服务器访问 JMX 度量：

- IBM WebSphere
- BEA WebLogic
- SAP NetWeaver
- Oracle AS
- Apache Tomcat
- JBoss J2EE 服务器
- TIBCO Business Works

Java 代理还可以从任何支持 JMX 标准的 J2EE 服务器中收集 JMX 数据。

Java 代理会定期运行 JMX 度量采集器，从应用程序服务器中收集度量。收集的度量显示在 HP Diagnostics 和 Diagnostics Java Profiler 的用户界面中。

针对 JMX 度量收集配置 WebSphere

对于 WebSphere JMX 度量收集，可能需要在 WebSphere 服务器上配置性能监控基础结构 (PMI) 服务，以便接收 JMX 度量。

有关如何针对 JMX 度量收集配置 WebSphere 5.x、6.x 和 7.0 服务器的信息，请参阅 “配置 WebSphere 以收集 JMX 度量”（第 209 页）。

配置 TIBCO 以收集 JMX 度量

对于 TIBCO JMX 度量收集，需要启用 JMX 度量收集，有关说明，请参阅 “要配置 TIBCO BusinessWorks 以进行 JMX 度量收集，请执行以下步骤：”（第 182 页）。

有关配置 JMX 度量采集器

可以通过配置 JMX 度量采集器来控制所收集的 JMX 度量类型。JMX 度量采集器在 <probe_install_dir>/etc/metrics.config 文件中定义。

通常需要为每个应用程序服务器的各主要版本定义单独的采集器。

有关度量采集器的常规信息，请参阅第 18 章，“Java 代理度量采集器”；有关度量采集器条目和语法的说明，请参阅 “了解度量采集器条目”（第 662 页）。

其他自定义 JMX 度量

Java 代理 与一些预定义的 JMX 度量采集器一起安装在 “关于 JMX 度量”（第 679 页）中列出的应用程序服务器上。通过定义 metrics.config 文件中的条目可以配置这些采集器，详情请参阅 “了解度量采集器条目”（第 662 页）。如果希望 Diagnostics 监控其他 JMX 度量，还可以在现有度量采集器中新建条目，甚至新建采集器。

为在 JMX 度量采集器中创建新的条目，您可获取可用的 JMX 度量和 WebSphere 性能监控基础结构 (PMI) 度量的列表。然后，在 **metrics.config** 文件中创建新的度量条目。以下各节提供在 JMX 度量采集器中新建条目，以监控其他 JMX 度量和 PMI 度量的说明。

获取可用的 JMX 或 WebSphere PMI 度量列表

与 Java 代理一起安装的度量采集器包括很多可供应用程序服务器使用的 JMX 度量的条目。当然，可能仍会有其他可监控的 JMX 度量或 WebSphere PMI 度量，或者由应用程序服务器供应商公开的新度量。

为了更加轻松地配置用于执行收集的新 / 其他 JMX/PMI 度量，metrics.config 文件具备一个特殊功能，可以将每个 JMX 采集器的所有可用度量的列表写入一个文件中。当 **metrics.config** 文件中的 **default.dump.available.metrics** 属性设置为 True 时，探测器会将此可用度量列表写入探测器日志目录下的文本文件中。这些文件的命名格式如下：**< 探测器安装目录 >/log/< 探测器 ID>/jmx_metrics_< 收集器名称 >.txt**。

可以在运行时更改探测器 **metrics.config** 文件的 **default.dump.available.metrics** 属性。建议只是临时将此属性设置为 True，以便写入可用 JMX/PMI 度量列表。将度量列表写入文件之后，应将属性重新设置回 False （或添加注释），避免因定期向文件写入度量列表而造成的探测器开销。

以下显示了度量列表文件的一些示例。可以使用此类信息在探测器的 **etc/metrics.config** 文件中配置其他 JMX 或 PMI 度量。

以下示例显示可用的 MBean ObjectNames 及其可收集属性：

```
======= MBean ObjectNames and Available Attributes =======
MBean ObjectName:
WebSphere:J2EEServer=server1,JDBCProvider=Derby JDBC
Provider,JDBCResource=Derby JDBC
Provider,Server=server1,cell=yli87Node01Cell,diagnosticProvider=true,j2eeType=JDB
CDataSource,mbe
anIdentifier=cells/yli87Node01Cell/nodes/yli87Node01/servers/server1/
resources.xml#DataSource_12442
31364323,name=WST_PriceGen,node=yli87Node01,platform=dynamicproxy,process=
server1,spec=1.0,
type=DataSource,version=6.1.0.0
Available Attributes:
name: loginTimeout, type: int
name: statementCacheSize, type: int
name: testConnectionInterval, type: java.lang.Integer
.....................
```

以下示例显示可用的 MBean ObjectNames 及其可收集属性和字段：

```
======= MBean ObjectNames and Available Attributes and Fields =======
MBean ObjectName:
java.lang:name=PS Old Gen,type=MemoryPool
Available Metrics:
Attribute: CollectionUsage type: javax.management.openmbean.CompositeData
    Field: committed, type: java.lang.Long
    Field: init, type: java.lang.Long
    Field: max, type: java.lang.Long
    Field: used, type: java.lang.Long
```

以下示例显示可用的 MBean ObjectNames 及其可收集操作和字段：

```
======= MBean ObjectNames and Available Operations and Fields =======
MBean ObjectName:
com.tibco.bw:key=engine,name="MortgageBroker-BrokerService"
Available Metrics:
Operation: java.lang.Integer GetActiveProcessCount()
Operation: javax.management.openmbean.CompositeData GetExecInfo()
    Field: Threads, type: java.lang.Integer
    Field: Uptime, type: java.lang.Long

Operation: javax.management.openmbean.CompositeData GetMemoryUsage()
    Field: FreeBytes, type: java.lang.Long
    Field: PercentUsed, type: java.lang.Long
    Field: TotalBytes, type: java.lang.Long
    Field: UsedBytes, type: java.lang.Long
```

对于 WebSphere JMX 采集器，除了常规 MBean JMX 度量以外，还会将可用 WebSphere 的特定 PMI 度量转储到 WebSphere 采集器的转储文件中。这包括 PMI 树实例路径及其可用统计信息，以及 PMI 模块配置信息，如下例所示：

```
======= PMI Tree and Available PMI Statistics =======
connectionPoolModule
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime
connectionPoolModule->Derby JDBC Provider
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime
connectionPoolModule->Derby JDBC Provider->jdbc/ALBUM
Available Statistics:
CreateCount, CloseCount, AllocateCount, ReturnCount, PoolSize, FreePoolSize,
WaitingThreadCount, FaultCount, PercentUsed, PercentMaxed, UseTime, WaitTime,
ManagedConnectionCount, ConnectionHandleCount, PrepStmtCacheDiscardCount,
JDBCTime
```

创建新的 JMX 或 WebSphere PMI 度量条目

以下说明将引导您按照以下模板完成创建 JMX 或 PMI 度量条目的整个过程：

```
<collector_name>/<metric_config>= <metric_id>|<metric_units>
```

“了解度量采集器条目”（第 662 页）中对此模板进行了描述。

要捕获 JMX 或 WebSphere PMI 度量，请执行以下操作：

1. 打开 **<probe_install_dir>/etc/metrics.config**，查找适用于受 Java 代理 监控的应用程序的 JMX 度量采集器。
2. **<collector_name>** 参数与采集器中的其余条目相同。如果要为 WebLogic 创建条目，则此参数的值将为 WebLogic。
3. 创建 **<metric_config>** 参数。
   a. 对于 JMX 度量， **<metric_config>** 参数是采集器用于查找匹配 MBean 的模式。此模式由两个组件组成，用 “.” 字符分隔。请见以下语法。

   MBean 对象和属性：

   ```
   <MBean object name pattern>.<attribute name>
   ```

   MBean 对象、属性和字段：

   ```
   <MBean object name pattern>.<attribute name>#<field name>
   ```

   MBean 对象和操作：

   ```
   <MBean object name pattern>.(<operationname>())
   ```

   MBean 对象、操作和字段：

   ```
   <MBean object name pattern>.(<operationname>()#<field name>)
   ```

- **<MBean object name pattern>** 是表示 MBean 对象名称的字符串。有关度量模式的说明，请参阅“了解度量模式”（第 687 页）。有关如何对 JMX 度量进行分组的说明，请参阅“JMX GROUPBY 和 EXPAND_PMI 修改器”（第 688 页）。

- **<attribute name>** 是表示度量的 MBean 属性的名称。如果 <attribute name> 中有任何 “.” ，则应将其用括号括起来：<MBean object name pattern>.(<attribute name>)

  例如，对于 WebLogic 应用程序服务器，所有 “执行队列”吞吐量的 <metric_config> 参数配置为：

  ```
  *:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount
  ```

  有关显示可用属性的度量转储的示例，请参阅 “获取可用的 JMX 或 WebSphere PMI 度量列表”（第 681 页）。

- **<attribute name>#<field name>** JMX 属性返回复合数据，可将其数字字段用作度量。只需将符号 # 及其后的字段名附加到 MBean 名称的后面。

  例如：

  ```
  Java\ Platform/java.lang\:type\=MemoryPool,name\=Perm\
  Gen.Usage#used
  ```

  将跟踪 <Perm Gen> MBean 的 <Usage> 复合数据属性的 <used> 字段。

- **(<operationname>())**，其中操作名称后跟随一对圆括号。整个操作名称括在圆括号内。如果操作返回复合属性，则会在 () 之后添加复合属性字段作为属性的后缀。

  例如：

  ```
  Tibco/com.tibco.bw\:key\=engine,name\=*.(GetActiveProcessCount()) =
  Active Process Count|count|Tibco
  ```

  请注意，仅支持不带参数的操作。

- **(<operation name>()#<field name>)** JMX 操作返回复合数据，可将其数字字段用作度量。只需将符号 # 及其后的字段名附加到 MBean 名称的后面。

  例如：

  ```
  Tibco/com.tibco.bw\:key\=engine,name\=*.(getStatus()#Total\ Errors) =
  Total Errors|count|Tibco
  ```

  将跟踪 getStatus() 操作返回的复合数据对象的 “Total Errors” 字段。

b 对于 WebSphere PMI 度量，**<metric_config>** 参数是采集器用于查找匹配 PMI 统计信息的模式。该模式由两个组件组成，用 “.” 字符分隔。

```
<PMI StatDescriptor>.<statistics name>
```

- **<PMI StatDescriptor>** 用于查找和访问 WebSphere PMI 树中的特定统计信息。它可以是 PMI 模块名称 （如 webAppModule），或 PMI 模块分支 （如 [webAppModule][AccountManagement#AccountManagementWar.war]
- **<statistics name>** 是表示度量的 PMI 统计信息的名称。如果统计信息名称中有任何 “.”，则应将其用括号括起来：
  [webAppModule][AccountManagement#AccountManagementWar.war].
  (webAppModule.numLoadedServlets)

有关 PMI 模块和 PMI 模块分支及其可用统计信息名称的示例，请参阅 “获取可用的 JMX 或 WebSphere PMI 度量列表”（第 681 页）。

有关如何对 PMI 度量进行分组的示例，请参阅 “JMX GROUPBY 和 EXPAND_PMI 修改器”（第 688 页）。

4 填入其余 JMX 度量条目模板，如下例所示：

```
WebLogic/*:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount =
RATE(Execute Queues Requests / sec|count|Execute Queues)
```

5 通过在每个反斜线 “\”、空格 “ ”、等号 (=) 或冒号 “:” 之前添加反斜线 “\”，来格式化初始条目。

执行此步骤后，上一个步骤中的初始条目会变为：

```
WebLogic/*\:Type\=ExecuteQueueRuntime,*.ServicedRequestTotalCount =
RATE(Execute Queues Requests / sec|count|Execute Queues)
```

对于 JMX 度量采集器而言，只有包含这样正确格式化的条目，采集器才能收集 WebLogic JMX 度量。

了解度量模式

对于 JMX 度量，<metric_config> 参数是采集器用于查找匹配 MBean 的模式；例如：

```
*:Type=ExecuteQueueRuntime,*.ServicedRequestTotalCount
```

在上述示例中，对象名称为 ***:Type=ExecuteQueueRuntime,***。该名称实际上可以解析为多个 MBean，其名称中的 **Type** 组件等同于 **ExecuteQueueRuntime**。**ServicedRequestTotalCount** 是属性名，其度量值由 JMX 度量采集器收集。

注意：当前实施的 JMX 采集器仅支持类型为数值 （如长整型、整等）的属性。

JMX 度量采集器首先使用 MBeanServer 的查询机制为配置中提供的所有对象名称查找匹配的 MBean。对于 JMX 度量，对象名称是采集器用于查找匹配 MBean 的模式。有关对象名称的详细信息，请参阅 http://java.sun.com/j2ee/1.4/docs/api/javax/management/ObjectName.html。

由于 MBean 对象名称是可以解析为多个 MBean 的模式， JMX 采集器将针对所有与模式匹配的 MBean 验证条目中的所有属性名，并聚合这组匹配的 MBean 和属性值。当然，对象名称并非总是解析为多个 MBean。例如，以下对象名称会解析为单个 MBean （在 WebLogic 应用程序服务器上）：

```
*\:Name\=weblogic.kernel.Default,Type\=ExecuteQueueRuntime,
*.ServicedRequestTotalCount
```

JMX GROUPBY 和 EXPAND_PMI 修改器

您可以使用可选的 GROUPBY 修改器，为每组匹配的 MBean ObjectName （其键值与 GROUPBY 指定的键值相同）创建单独的度量。在探测器的 etc/metrics.config 文件中，可以为描述 MBean 对象名称模式的 JMX 度量添加可选的修改器 GROUPBY，这意味着基于 JMX 的采集器会将 metric_config 视为多实例表达式：

```
collector_name/GROUPBY[oname_key]/metric_config = ...
```

采集器将查找所有与 metric_config 匹配的 MBean，并为每个 MBean 创建一个对应的度量 （通过将对象名称关键字 oname_key 附加到 category_id 来为 MBean 提供唯一命名）。

```
WebSphere6/GROUPBY[name]/
WebSphere\:type\=DataSource,*.statementCacheSize = JDBC Statement
Cache Size|bytes|JDBC DataSource
```

例如：

```
WebSphere6/connectionPoolModule.CreateCount = JDBC Connection
Creates|count|JDBC ConnectionPools
```

```
WebSphere6/[connectionPoolModule][Derby\ JDBC\ Provider][jdbc/
ALBUM].AllocateCount = JDBCConnection Allocates|count|JDBC
ConnectionPools
```

或者，您也可以使用可选 EXPAND_PMI 修改器对 PMI 度量进行分组，分组方法与 JMX 度量分组类似。

对于 PMI，EXPAND_PMI 修改器专用于根据指定级别从给定模块或 StatDescriptor 分支展开 PMI 树。展开级别“n”可以是 1、2 ... 或 *，默认级别为 1，而 * 表示全部展开：

```
collector_name/EXPAND_PMI[n]/metric_config = ...
```

例如：

```
WebSphere6/EXPAND_PMI[*]/connectionPoolModule.AllocateCount = JDBC Connection Allocates|count|JDBC ConnectionPools
```

为每个 JDBC 连接池提供程序和每个提供程序的 DataSource 创建“JDBC 连接分配次数”度量。

第 VII 部分

设置与其他 HP 软件产品的集成

本部分包括：

- 设置 Business Service Management 和 Diagnostics 之间的集成
- 安装 LoadRunner Diagnostics 插件
- 设置 HP LoadRunner 与 HP Diagnostics 集成
- 设置 Performance Center 以使 Diagnostics

21

设置 Business Service Management 和 Diagnostics 之间的集成

本章提供有关设置 HP Business Service Management 和 Diagnostics 之间的集成的信息。

注意：除非另有声明，否则，本文档仅涉及 Diagnostics 与 Business Service Management 9.x 的集成。有关受支持的集成的最新信息，请参阅 HP Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

本章包括：

关于设置 Business Service Management 和 Diagnostics 之间的集成

在将 HP Diagnostics 与 Business Service Management 一起使用之前，请向 Business Service Management 提供它与 Diagnostics 组件通信时所需的信息。

要在 Business Service Management 中设置 Diagnostics，请执行以下操作：

1 指定 Diagnostics 服务器的详细信息。

在 Business Service Management 中输入 Diagnostics 服务器的详细信息。有关详细信息，请参阅“在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）。

2 分配相关权限 （可选）。

将不同的权限授予不同的 Diagnostics 用户（此为可选步骤）。有关详细信息，请参阅 “在 Business Service Management 中为 Diagnostics 用户分配权限”（第 704 页）。

3 仅限于 Windows 2003：更改 Internet 浏览器设置。

当 Internet 浏览器在 Windows 2003 环境中运行时，必须更改 Internet 浏览器设置，才能在 Business Service Management 中访问 Diagnostics 配置和应用程序页面。请参阅 “在 Windows 2003 中访问 Diagnostics 页面”（第 707 页）。

4 在 Diagnostics Commander 主机上启用 Cookie。

必须启用 Cookie 才能在 Business Service Management 中查看 Diagnostics 数据。通常情况下，通过在浏览器配置中将注册的 Diagnostics Commander 服务器添加为可信站点，即可完成此任务。

在 Business Service Management 中注册 Diagnostics 服务器

要能通过 Business Service Management 访问 HP Diagnostics，请注册 Diagnostics 服务器。下一节描述在 Business Service Management 中注册的步骤。同时还列出了与 BAC 8.x 之间的差异。

重要信息：升级 Business Service Management 之后，必须重新注册 Business Service Management-Diagnostics 集成。

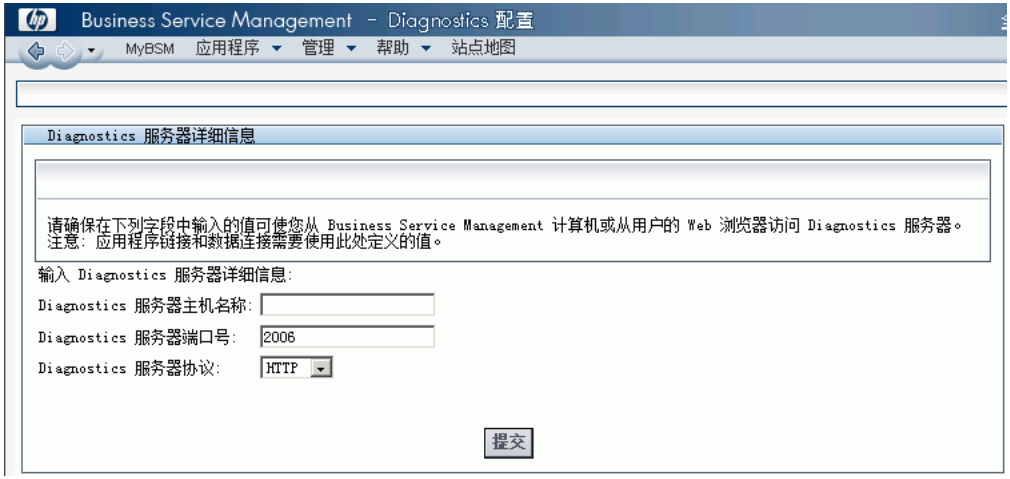
注意：如果使用的是 Windows 2003，则必须配置 Internet 浏览器设置，才能访问 “Diagnostics 管理”页。有关详细信息，请参阅 “在 Windows 2003 中访问 Diagnostics 页面”（第 707 页）。

如果 Diagnostics 已与 Business Service Management 集成，则在 Diagnostics Commander 服务器升级后，必须从 **etc.old** 文件夹把 **RegistrarPersistence.xml** 文件复制到新的 **etc** 文件夹。然后在 “BSM > 管理 > Diagnostics”页面检查 Diagnostics 集成，如果无法正常运行，请在 BSM 内重新注册 Diagnostics 服务器。

首次在 Business Service Management 中注册 Diagnostics 服务器时请执行以下操作：

1 登录 Business Service Management。

2 选择 “管理” > “Diagnostics”。首次设置集成时，会显示 “Diagnostics 服务器详细信息” 页面。



注意：如果尝试在配置 Diagnostics 服务器之前访问 HP Diagnostics （通过在 “Site 映射” 页上单击 “Diagnostics”，或选择 “应用程序” > “Diagnostics”），您将收到一条消息提示您注册 Diagnostics 服务器。单击链接，打开 “Diagnostics 配置” 页。

---

3 输入 Diagnostics 命令服务器的详细信息。

➤ **Diagnostics 服务器主机名称。**输入 Diagnostics 命令服务器主机的计算机名称。

即使 Diagnostics 服务器与 Business Service Management 安装在同一系统上，仍然需要在 “Diagnostics 服务器主机名称” 框中输入主机的实际名称，而不能仅输入 **localhost** 来代替主机名。

如果要通过完全限定域名访问 Business Service Management，请使用完全限定域名注册 Diagnostics 服务器主机。

- **Diagnostics 服务器端口号。**输入由 Diagnostics 命令服务器使用的端口号，默认端口号是 **2006**。
- **Diagnostics 服务器协议。**选择通信协议， Business Service Management 将通过该协议连接到 Diagnostics。该协议可以是 **HTTP** 或 **HTTPS**。

注意：如果选择 **HTTPS** 作为通信协议，则需要执行其他配置步骤。有关所需步骤的详细信息，请参阅附录 C，“在组件之间启用 HTTPS”。

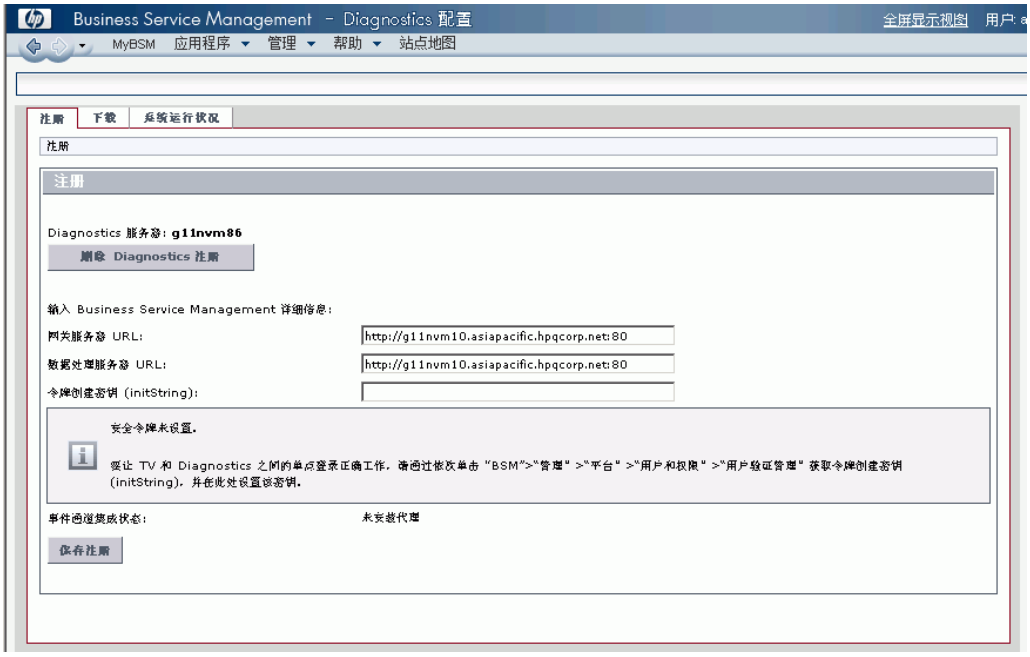
---

4 在输入 Diagnostics 服务器详细信息并验证这些信息的准确性之后，单击“提交”完成 Diagnostics 服务器的配置过程。

如果输入的服务器名不正确或者服务器不可用，则会显示错误消息。

单击 “提交”后， Diagnostics 服务器的详细信息将保存在 Business Service Management 中，并且 Business Service Management 服务器的详细信息会自动在 Diagnostics 服务器计算机上注册。

5 此时“Diagnostics 配置”页中的“注册”选项卡将打开，显示可用的 Business Service Management 服务器详细信息。



Business Service Management 详细信息

如果需要，可以在“注册”选项卡的“输入 Business Service Management 详细信息”部分中手动更改 Business Service Management 服务器的详细信息。

网关服务器 URL。验证此根 URL 与用于访问 Business Service Management 的根 URL 是否匹配。

数据处理服务器 URL。验证此根 URL 与用于访问 Business Service Management 的根 URL 是否匹配。如果数据处理器 URL 与网关服务器 URL 不同，则通常会在此 URL 中使用端口 8080。

注意：如果无法从 Diagnostics 服务器访问 BSM 处理服务器，且 BSM 网关服务器已配置为向处理服务器传送证书请求 （例如，当网关和处理服务器位于负载平衡器或 SSL 加速器的另一端时），则请在此 Diagnostics 服务器注册页面将 BSM 网关作为 BSM 网关服务器和处理器使用。您可能仍需要按照下页的说明，在处理服务器上手动授予证书。

令牌创建密钥 (initString)。如果您使用的是 TransactionVision，并且希望避免从 TransactionVision 向下搜索到 Diagnostics 时用户重新登录，请在该字段中输入 Business Service Management 令牌创建密钥。输入标记密钥可将密钥提供给 Diagnostics。如果没有使用 TransactionVision，则无需执行此操作。

可以在 Business Service Management 中的“管理”>“平台”>“用户和权限”>“身份验证管理”中查找“令牌创建密钥 (initString)”。完成注册时，此令牌创建密钥会写入 Diagnostics **lwsso.properties** 文件。

事件通道集成状态。将 Diagnostics 9.x 与 Business Service Management 9.x 集成时， Diagnostics 使用事件通道将运行状况指标状态事件发送到 Business Service Management 网关服务器 （实际使用的是 OM 代理和 IAPA 组件）。

注册执行脚本，以进行事件通道集成 （**<Diagnostics 安装目录 >/server/bin/switch_ovo_agent.sh** 或 .vbs） 。运行脚本需要 UNIX 上的 root 用户访问权限。如果 Diagnostics Commander 服务器未作为 UNIX 上的根运行，则 BSM 内的 Diagnostics 的注册将由于权限拒绝错误而失败。收到此错误后，必须在 Diagnostics Commander 服务器上以 root 身份手动执行脚本。在手动运行脚本之前先完成 BSM 注册步骤是至关重要的。

重要信息：要使用事件通道集成在 BSM 网关服务器和 BSM 处理服务器之间通信，如果服务器位于不同的系统上，则在这两个计算机之间必须存在信任关系。如果需要设置此关系，请参见 “用于 DPS 和网关的单独 BSM 服务器的配置” （第 716 页）。

“事件通道集成状态”为红色时，请查看显示的消息是否表示证书请求处于挂起状态。注册执行脚本，以进行事件通道集成（**<Diagnostics 安装目录 >/server/bin/switch_ovo_agent.vbs** 或 **.sh**） 。但前提是手动执行了某些步骤授予证书，该状态才会表示证书请求处于挂起状态。

这种情况下，必须执行以下步骤完成事件通道集成。

要手动授予证书，请执行以下操作：

1 转到 Business Service Management 数据处理服务器，并授予 OM 代理证书。如果有多个证书列出，请选择一个合适的证书。

ovcm -listpending -l

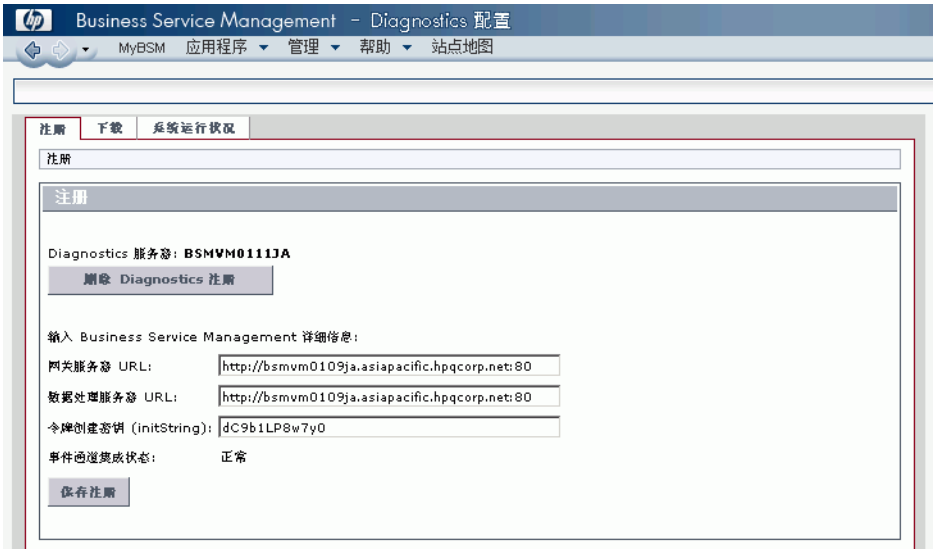
ovcm -grant < 上述输出中的核心 ID>

如果有关于 OM 代理和 IAPA 安装或证书的任何问题，请参阅 “OM 代理疑难解答”（第 875 页）。

注意：将 Diagnostics 与 Business Service Management 8.x 集成后，您将看见“事件通道集成状态”为“暂缺”，因为与 Business Service Management 8.x 集成后， Diagnostics 不再使用 OM 代理和 IAPA 组件。

---

2 单击“保存注册”，验证“事件通道集成状态”为“正常”，并且其他值都正确。



删除 Diagnostics 注册

Diagnostics 注册可完全删除。

要删除 Diagnostics 注册，请执行以下操作：

1 选择 “管理” > “Diagnostics”。

2 在 “注册”选项卡中，单击 “删除 Diagnostics 注册”按钮。

3 在打开的消息中单击 “确定”，确认要删除 Diagnostics 注册。

此时将显示一条消息，确认已成功删除 Diagnostics 注册。

要注册新的 Diagnostics 服务器，请选择“管理”>“Diagnostics”，并执行“在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）中所述的步骤。

了解 “Diagnostics 管理”页

通过选择 “管理” > “Diagnositcs”，访问 “Diagnositcs 配置”页面。Business Service Management 中的 “Diagnositcs 管理”页由以下所述的三个选项卡组成：

“注册”选项卡

“注册”选项卡显示以下详细信息：

- 在 Business Service Management 中注册的 Diagnostics 服务器详细信息。要更改这些详细信息，请参阅 “删除 Diagnostics 注册”（第 703 页）。
- 自动注册在 Diagnostics 服务器计算机上的 Business Service Management 服务器详细信息。可以在 “输入 Business Service Management 详细信息”部分中手动更改 Business Service Management 服务器详细信息。

有关首次在 Business Service Management 中注册 Diagnostics 的信息，请参阅“在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）。

“下载”选项卡

“下载”选项卡提供 Diagnostics Agent 和采集器安装程序的链接，供您下载相关平台的代理或采集器安装文件。

如果未在 Diagnostics 服务器安装期间指定 Agent 和 Collector 安装程序的路径，则“下载”选项卡不会显示任何组件。有关详细信息，请参阅第 2 章，“安装 Diagnostics 服务器”。

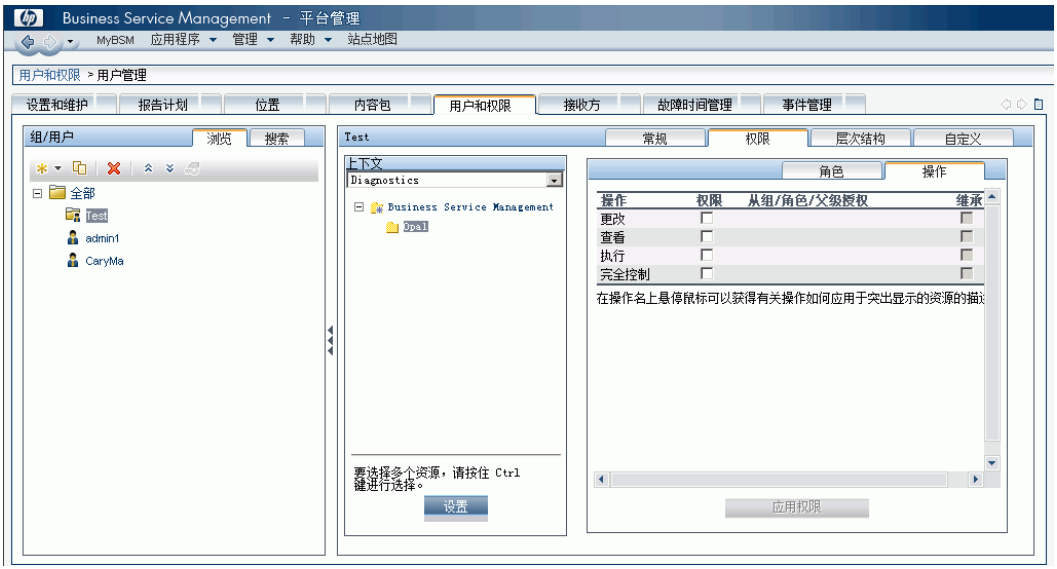
“系统运行状况”选项卡

向您提供 HP Diagnostics 部署的所有组件图，以及每个组件的实时状态和运行状况信息。

在 Business Service Management 中为 Diagnostics 用户分配权限

Business Service Management 支持您针对系统中定义的特定资源，将权限应用于用户或用户组。有些特定类型的权限操作可由管理员授予 Diagnostics 用户。

下例显示 Business Service Management 中的“管理”>“平台”>“用户和权限”>“用户管理”页：



Diagnostics 的权限在名为“Diagnostics”的上下文下。树显示 Business Service Management 以及它下面的 Diagnostics。在 Business Service Management 8.x 中， Diagnostics 的权限则在 “监控器”上下文下。

在 Business Service Management 中应用权限时，管理员可以将以下类型的权限操作授予 Diagnostics 用户：

- **更改**：允许查看 Diagnostics 管理和配置 Diagnostics 设置。
- **查看**：允许用户在通过 Business Service Management 访问 Diagnostics 时，查看 Diagnostics 应用程序。
- **执行**：允许在 Diagnostics 中设置阈值。
- **完全控制**：允许对 Diagnostics 执行所有操作，并且允许授予和删除执行这些操作的权限。

Diagnostics 权限可以从 BSM 角色继承。

有关如何在 Business Service Management 中分配用户权限的详细信息，请参阅 *HP Business Service Management* 文档库中的 《平台管理》。

数据采集器用于访问 RTSM 的密码

Business Service Management 配置服务器启动过程中，可以选择覆盖用于数据收集的默认密码。在 “Business Service Management 设置和数据库配置实用程序的登录设置”页面覆盖此密码后，同一密码将用于所有的数据采集器 （TV、BPI、 RUM 和 Diagnostics）。

如果在 BSM 内覆盖了此默认密码，则必须对 Diagnostics 内的密码做出对应更改以匹配新密码。可在 **etc/cmdbProperties.xml** 文件内的 Diagnostics 服务器上做出更改。

```
<customer>
        <!-- customerId is an Integer -->
        <customerId>1</customerId>
        <customerName>Default Client</customerName>
        <userName>diagnostics</userName>
        <!-- userPassword may be obfuscated -->
        <userPassword>integration</userPassword>
    </customer>
<customer>
```

在 Windows 2003 中访问 Diagnostics 页面

当 Internet 浏览器在 Windows 2003 环境中运行时，必须更改 Internet 浏览器设置，才能在 Business Service Management 中访问 Diagnostics 配置和应用程序页面。

要在 Windows 2003 环境中访问 Diagnostics 页面，请执行以下操作：

1 在 Internet 浏览器中，选择“工具”>“Internet 选项”，打开“Internet 选项”对话框。

2 在 “隐私”选项卡中，单击 “站点”，打开 “每个站点的隐私操作”对话框。

3 在 “网站地址”框中，输入 Diagnostics 服务器的名称。

- 如果在 Business Service Management 中注册 Diagnostics 服务器时输入了 IP 地址，请输入该 IP 地址。如果在 Business Service Management 中输入了主机名，请输入该主机名。
- 包括 “http://”或 “https://”前缀以及端口号，如以下示例中所示：

  http://<Diagnostics_Commander>:2006/

4 单击 “允许”。

5 单击 “确定”，关闭 “每个站点的隐私操作”对话框。

6 单击 “确定”，关闭 “Internet 选项”对话框。

从 Business Service Management 访问 Diagnostics 应用程序

设置 Business Service Management 以使用 Diagnostics 之后，即可从 Business Service Management 中访问 Diagnostics UI。

在 Business Service Management 中，选择 “应用程序”>“Diagnostics”，即可打开 Diagnostics UI。

通过 Business Service Management 和 Diagnostics 之间的集成，可将 Diagnostics 中的数据发送到 Business Service Management。

还可以在 Business Service Management 的各种视图中查看有关应用程序基础结构元素和业务事务的这些 Diagnostics 数据。并且，由于 Diagnostics 和 Business Service Management 共享一个公用数据模型，您可以选择通过上下文直接从 Business Service Management 中向下搜索到 Diagnostics。

对于 Business Service Management 中由 Diagnostics 填充的选定 CI，可以右键单击该 CI，选择 “搜索 Diagnostics”。在 Business Service Management 的各种报告中，可以选择一个通过上下文向下搜索到 Diagnostics 的图标。

发送到 Business Service Management 的数据样本

将 Diagnostics 与 Business Service Management 集成时， Diagnostics 会监控企业应用程序，并将应用程序性能和可用性数据作为“数据样本”发送到 Business Service Management。有关详细信息，请参阅《HP Diagnostics User's Guide》关于集成的章节。

Diagnostics 向 Business Service Management 提供以下数据样本：

- ws_perf_aggr_t （SOA 样本）
- ws_event_aggr_t （SOA 样本）
- appmon_vu_t （事务 (BPM) 样本）
- dg_trans_t （业务事务 (Diagnostics) 样本）

有关数据样本的详细信息，请参阅 *HP Business Service Management 文档库*。

Diagnostics 填充 Business Service Management 内的 CI 和模型

对于 Diagnostics 9.0 或更高版本，Diagnostics 会在 Business Service Management 运行时服务模型 (RTSM) 中填充有关应用程序基础结构元素和业务事务的 CI 和模型关系。有关详细信息，请参阅 《HP Diagnostics User's Guide》关于集成的章节。

对于 ASP.NET 应用程序，如果您要修改现有的 ASP.NET 应用程序部署，请参阅“为在 IIS 下部署的 ASP.NET 应用程序的 CI 填充搜寻 IIS 元数据”（第 282 页），了解如何搜寻填充运行时服务模型时所需的 IIS 元数据以及如何运行重新扫描。

如果您偶尔希望更改将这些 CI 添加到运行时服务模型的进程计时， Diagnostics 的 **server.properties** 文件中提供了一些属性。

在 Diagnostics 和 Business Service Management 之间同步 CI

如果需要为 Diagnostics 填充模型 (CI) 在 Diagnostics 与 Business Service Management 之间强制同步，可以使用 Diagnostics 服务器上的 “同步”功能。

从主 Diagnostics UI 中选择 “配置诊断程序”（或在任何 Diagnostics 视图中选择右上角的 “维护”链接），即可显示 “组件”页面。选择 “同步”链接，以显示用于同步模型的页面。

只要更新或重新安装了 Business Service Management 系统，都需要在将 CI 从 Diagnostics 转发到 Business Service Management 之前手动执行硬同步（或等待 12 小时）。要执行硬同步，请选择“硬”。

Diagnostics 向 Business Service Management 提供 KPI/HI 颜色

由 Diagnostics 填充的业务事务和 Web 服务 CI 的运行状况指标状态（颜色）是**基于度量**的，基于度量的 KPI 和运行状况指标状态将以数据样本的形式从 Diagnostics 发送到 Business Service Management。Diagnostics 将数据样本发送到 Business Service Management，Business Service Management 中的规则用于评估数据以及设置指标状态。

可以在 Business Service Management “管理” > “服务运行状况”中更改业务事务和 Web 服务运行状况指标的默认目标。有关使用服务运行状况管理的信息，请参阅 *Business Service Management 文档库*。

由 Diagnostics 填充的应用程序基础结构 CI 的运行状况指标状态（颜色）是**基于事件**的，当相关度量存在阈值冲突时，会将基于事件的运行状况指标状态从 Diagnostics 发送到 Business Service Management。

使用随 Diagnostics Commander 服务器一起安装的 OM 代理和 IAPA 组件，将阈值冲突事件数据发送到 Business Service Management。

有关安装 OM 代理/IAPA 组件的信息，请参阅第 2 章，“安装 Diagnostics 服务器”。有关检查事件通道状态的详细信息，请参阅 “在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页）。如果有问题，请参阅 “基于事件的运行状况指标状态排疑解难流程”（第 872 页）。

有关 KPI 和 HI 状态颜色的概述信息，以及由 Diagnostics 应用程序和基础结构性能数据填充的 Business Service Management 视图的概述信息，请参阅《HP Diagnostics User's Guide》。

启用 Diagnostics 与 BSM Service Health Analyzer 的集成

您可以启用 Diagnostics 与 BSM Service Health Analyzer (SHA) 的集成。利用此集成，包含主机度量和探测器度量的样本将通过 Diagnostics Commander 服务器从 Diagnostics Mediator 服务器发送并传播至度量将置于 BSM SHA 数据库内的 BSM。

SHA 应用程序将使用这些 Diagnostics 探测器和主机度量以及来自其他样本的度量创建基线。 SHA 应用程序会将度量与基线对比，并将异常情况作为检测出的性能问题报告。有关异常情况，可搜索至 “Diagnostics 探测器”视图或 “主机”视图以了解详细的 Diagnostics 数据 （请参阅 BSM Service Health Analyzer 文档以了解使用 SHA 的详细信息）。

默认情况下 Diagnostics 与 SHA 的集成为禁用状态。

要启用集成，请执行以下步骤：

1. 在每个 Diagnostics Mediator 上找到 **/etc/server.properties** 文件。
2. 做出以下更改。

```
# Send host metrics for Service Health Analyzer (SHA)
bac.diag.sha.host.metric.create.samples=true
# Send probe metrics for Service Health Analyzer (SHA)
bac.diag.sha.probe.metric.create.samples=true
```

3. 重新启动每个 Diagnostics Mediator。
4. 启动集成，且来自 Diagnostics 的主机度量和探测器度量在 Business Service Management 的 SHA 数据库内可用后，则可在 SHA 管理应用程序内收集这些 CI 以用于异常情况检测。

   可在 Diagnostics 内定义筛选器以确定要发送至 SHA 数据库的主机和探测器度量。请使用 Diagnostics 服务器 **/etc** 目录内的以下 XML 文件对这些度量进行筛选。筛选器基于与数据报告类似的正则表达式匹配。

- **shaHostMetrics.xml.** 包含 / 排除用于主机度量的筛选器
- **shaProbeMetrics.xml.** 包含 / 排除用于探测器度量的筛选器

Diagnostics 和 OM 服务器共存

Diagnostics 9.x 捆绑并使用 OM 代理以向 BSM 9.x 发送运行状况指标 (HI) 更新事件。以下步骤描述了在用于 Diagnostics Commander 服务器的系统已包含 OM 代理且希望对 OM 服务器和 BSM 服务器配置报告时，需作出的必要更改。

配置受信任的证书

在具有多 BSM/OM 服务器的环境中，必须对每个服务器进行配置，以使其信任其他服务器颁发的证书。此任务涉及导出每台服务器的受信任证书，然后将此受信任证书分别导入其他每台服务器。必须更新代理的受信任证书，以便代理也可信任 BSM/OM 服务器。

要为每个 BSM/OM 配置受信任的证书，请执行以下步骤：

1. 在每台 BSM/OM 服务器上，使用以下命令将受信任证书导出到文件：

   ovcert -exporttrusted -file <file>

   此命令将生成一个具有您指定的名称的文件。

2. 将每个文件分别复制到其他每台服务器，然后使用以下命令导出受信任证书：

   ovcert -importtrusted -file <file>

   ovcert -importtrusted -ovrg server -file <file>

3. 在 Diagnostics 系统上 （如果已安装代理），使用以下命令更新受信任证书：

   ovcert -updatetrusted

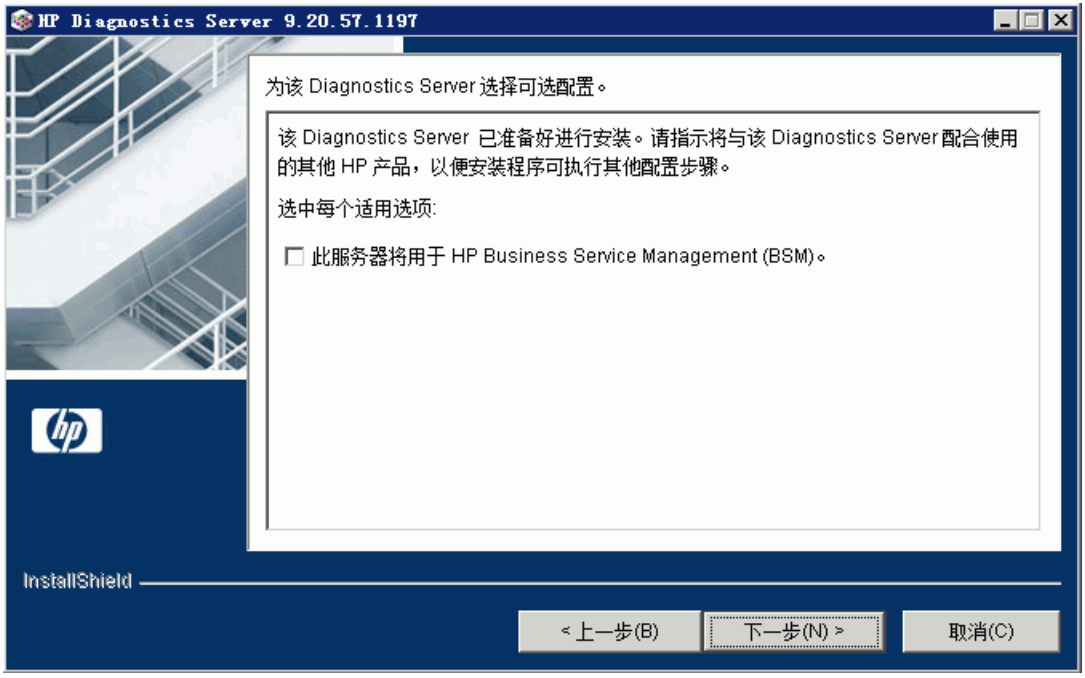
在安装 Diagnostics 之前已安装 OM 代理

此场景假定 OM 代理已存在，并对装有 Diagnostics Commander 服务器的系统上的 OM 服务器报告。

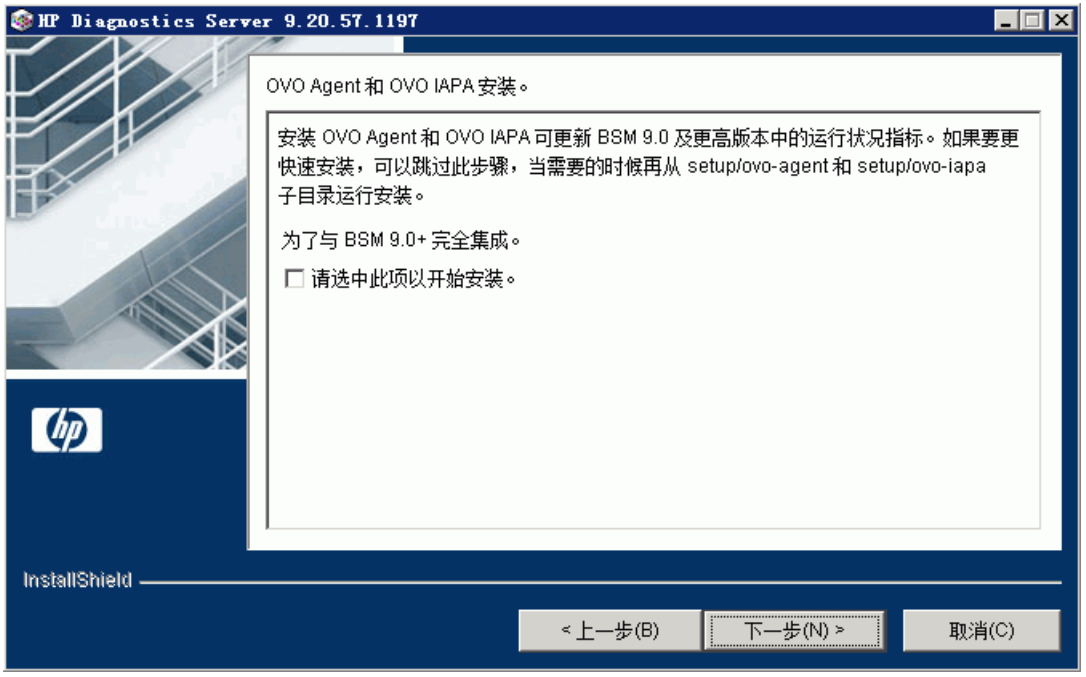
要在先安装了 OM 代理后设置共存，请执行以下步骤：

1 安装 Diagnostics Commander 服务器（请参阅“安装 Diagnostics 服务器”（第 51 页））。

在此步骤中，请检查 BSM 集成框：



安装进行到这一步时，请勿勾选下框：



2 在 Diagnostic Commander 服务器上，安装 IAPA 组件（请参阅“手动安装 OM 代理和 IAPA 组件”（第 72 页）了解相关说明）。

3 通过 BSM 内的“Diagnostics 管理”页面在 BSM 内注册 Diagnostics（请参阅“设置 Business Service Management 和 Diagnostics 之间的集成”（第 693 页））。请注意，如果 Diagnostics Commander 服务器位于 Linux 上，且未将 Diagnostics 服务器作为根运行，则将在 Diagnostics 的 BSM 注册过程中遇到错误。忽略此错误并继续。

4 在 Diagnostics Commander 服务器上，转至 **<Diagnostics_server_install_dir>\bin** 和 **execute switch_ovo_agent.vbs**，或在 UNIX 上转至 **switch_ovo_agent.sh**，指定 OM 服务器为 -server 和 -cert_srv 的目标。请注意，在 Linux 上，必须运行此命令作为根。

例如：

```
cscript switch_ovo_agent.vbs -server ovruxt65.rose.hp.com -cert_srv
ovruxt65.rose.hp.com
```

5 确定 BSM 网关服务器和 OM 服务器的核心 ID。在 Diagnostics Commander 上执行：

```
bbcutil -ping <OM 服务器 >
```

```
bbcutil -ping <BSM 网关服务器 >
```

6 复制目录：
<Diagnostics_server_install_Dir>\newconfig\ovo-agent\policies\mgrconf
到 **<Diagnostics_server_install_dir>\newconfig\ovo-agent\policies\tmp**

如果 mgrconf 目录不存在，请联系支持人员以获取此目录的内容。同样，如果您有更复杂的配置（例如，具有多个 OM Manager），则可能需要对以下文件作出其他更改。

7 编辑文件：
<Diagnostics_server_install_dir>\newconfig\ovo-agent\policies\tmp\mgrconf\FF9A8F04-B5E3-43C3-999B-7A9492C35014_data。

- 找到字符串 ${OM_MGR_SRV}，并用 HPOM 管理服务器的 FQDN 替换所有匹配项。
- 找到字符串 ${OM_MGR_SRV_ID}，并用 HPOM 管理服务器的核心 ID 替换所有匹配项。
- 找到字符串 ${OMi_MGR_SRV}，并用 BSM 网关服务器的 FQDN 替换所有匹配项。
- 找到字符串 ${OMi_MGR_SRV_ID}，并用 BSM 网关服务器的核心 ID 替换所有匹配项。

请注意，如果有更复杂的 OM 设置，则可能需要在此文件内添加其他条目。

8 转至目录：
<Diagnostics_server_install_dir>\newconfig\ovo-agent\policies\tmp 并安装策略：

ovpolicy -install -dir mgrconf

9 代理附带的 Diagnostics 特定日志文件压缩模板将向 BSM 服务器报告，而所有其他策略将向 OM 服务器报告。

Diagnostics 已安装

建议首先卸载 OM 代理，再使用上述步骤。

Diagnostics 9.x 捆绑并使用 OM 代理以向 BSM 9.x 发送运行状况指标 (HI) 更新事件。以下步骤描述了在已安装 Diagnostics 和 Diagnostics 随附的 OM 代理和 IAPA 组件之后，希望配置事件以流至 OM 服务器和 BSM 服务器时，需作出的必要更改。

要在已安装 Diagnostics 的前提下设置共存，请执行以下步骤：

1 卸载伴随 Diagnostics 一起安装的 OM 代理组件 （请参阅 “手动卸载 OM 代理和 IAPA 组件”（第 74 页）了解相关说明）。

2 转至您的 OM 服务器，并将 OM 代理部署到 Diagnostics Commander 服务器。

3 然后执行上一页的步骤 2 至步骤 9。

用于 DPS 和网关的单独 BSM 服务器的配置

将 Diagnostics 9.0 或更高版本与 Business Service Management 9.0 或更高版本集成时，Diagnostics 使用事件通道将运行状况指标状态事件从 Diagnostics 发送到 Business Service Management 网关服务器 （实际使用的是 OM 代理和 IAPA 组件）。

Diagnostics 需要设置 OMi 通信通道，有关在 BSM 内检查事件通道集成状态的信息，请参阅 “Business Service Management 详细信息”（第 699 页）。

BSM 网关服务器和 BSM 数据处理服务器可分别在单独系统上设置 （有关执行此设置的方法，请参阅 BSM 文档）。如果这些服务器在不同的系统上运行，且已具备事件通道集成，则在这些计算机之间必须具有信任关系，以便 BSM 网关服务器和 BSM 处理服务器进行通信。

要在单独的网关服务器上设置证书，请执行以下操作：

1 在 BSM 网关服务器上，执行以下命令：

ovconfchg -ns sec.cm.client -set CERTIFICATE_SERVER <processing_server>

和：

ovcert -certreq

2 在 BSM 处理服务器上，执行以下命令：

ovcm -listpending -l

和：

ovcm -grant <reqid>

3 在 BSM 网关服务器上，执行以下命令：

ovcert -list

和：

bbcutil -ping <processing_server>

有关更多信息，请参阅 *HP Business Service Management* 联机帮助。

有关集成的其他信息

下方提供了设置 Diagnostics 和 Business Service Management 之间集成的其他信息。

显示身份验证对话

在 Business Service Management 服务器之外的其他域安装了 Diagnostics 服务器时，如果未设置轻量单一登录 (LWSSO) 将 Diagnostics 服务器添加为受信任的域，“MyBSM Diagnostics 控制面板”可能会在 Diagnostics 控制面板小程序显示之前显示身份验证对话。

要解决此问题，请确保 Diagnostics 服务器运行所在的域在 Business Service Management 的 “单一登录”页面内列出。

1 在 Business Service Management 内选择 “管理” > “平台” > “用户和权限” > “身份验证管理” > “单一登录配置”。

2 单击 “配置”按钮。

3 在向导内单击 “下一步”转到 “单一登录”页面。

4 单击 “添加受信任的主机 / 域”图标，然后输入 Diagnostics 服务器的域。

5 单击 “下一步”。

6 单击 “下一步”。

7 单击“完成”。此操作会注销 Business Service Management。重新登录 Business Service Management，然后打开 “MyBSM Diagnostics 控制面板”。

Diagnostics UI 内缺失链接

如果从 Business Service Management 启动 Diagnostics UI，且在同一系统内以独立模式启动 Diagnostics UI，则独立模式下 Diagnostics UI 内的 “维护”链接将不可用。

要解决此问题，请关闭 Diagnostics UI 的两个实例，然后重新启动 Diagnostics 独立 UI。

HI 事件未流至 Business Service Management

与 Business Service Management 9.x 集成时，Diagnostics 会将运行状况指标状态事件发送到 Business Service Management 网关服务器。如果 HI 事件流至 Business Service Management 存在问题，则请参阅附录 H，“HP Diagnostics 疑难解答”内 “OM 代理”、“BSM 网关信任关系”和 “基于事件的 HI 状态排疑解难流程”的相关章节。

第 VIII 部分

Diagnostics 服务器、 Java 代理和 .NET 代理的高级配置

本部分包括：

- Diagnostics 服务器高级配置
- 高级 Java 代理与应用程序服务器配置
- 了解 .NET 代理配置文件
- 高级 .NET 代理配置

22

安装 LoadRunner Diagnostics 插件

通过 LoadRunner Diagnostics 插件，可以从 LoadRunner 内访问 Diagnostics UI。LoadRunner Diagnostics 插件安装完成后，可以配置 LoadRunner，从而在负载测试期间用 Diagnostics 组件收集性能度量以及连接到 Diagnostics UI。

本章包括：

安装 LoadRunner Diagnostics 插件前的准备工作

安装 LoadRunner Diagnostics 插件之前，必须先安装 Diagnostics Commander 服务器和 LoadRunner。要安装 LoadRunner，请参阅《HP LoadRunner 安装指南》。

对于 Diagnostics 版本 8.0x、 9.0x、 9.10 和更高版本，需使用最新发布的 Diagnostics 9.10 LoadRunner 插件。

安装 LoadRunner Diagnostics 插件

LoadRunner Diagnostics 插件安装在 LoadRunner Controller 主机上。

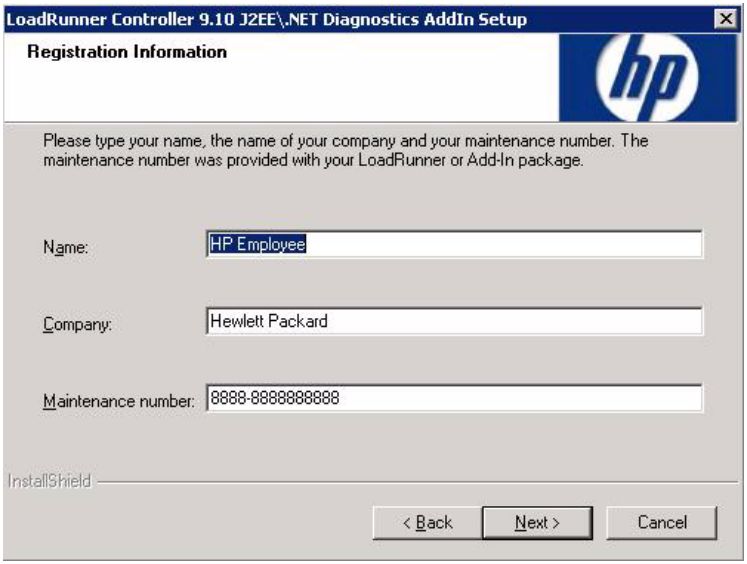
注意：LoadRunner Diagnostics 插件首次执行时，会使用小型引导程序，动态从 Diagnostics 服务器下载所需的大多数软件。如果 Diagnostics 已更新，则在下次执行 LoadRunner 时，会自动选取新的 Diagnostics 文件。

要安装 LoadRunner Diagnostics 插件，请执行以下操作：

1. 关闭在 LoadRunner Controller 系统上运行的 LoadRunner 或与 LoadRunner 相关的所有进程，如 LoadRunner 代理。
2. 从 Diagnostics 安装光盘的 LR_AddIn 目录中运行 **setup.exe** 文件。此时将启动安装程序。

3 此时将显示软件许可证协议。阅读协议，然后单击 “Yes”，接受协议。

此时将打开 “Registration Information”对话框。



在 “Registration Information”对话框中，键入姓名、公司名称和 LoadRunner 序列号。可以在 LoadRunner 附带的维修包中找到该序列号。

单击 “Next”，启动安装过程。安装过程随即开始。

4 安装过程完成后，安装向导会显示确认消息。

单击 “Finish”即完成安装过程。

如果您的计算机上运行有相关的 LoadRunner 进程，如 LoadRunner 代理，则系统将会要求您重新启动计算机，完成 LoadRunner 插件安装过程。

注意：未提供 LoadRunner Diagnostics 插件的卸载实用程序。

注意：对于应用了 Service Pack 1 和 Windows XP Hotfix Q328310 的 Windows XP 计算机，如果要在其上安装 LoadRunner Diagnostics 插件，则会收到 **iKernal.exe** 应用程序错误消息。因为 Windows XP Hotfix Q328310 包含的 Win32 API 不按 InstallShield 引擎期望的方式执行，所以系统会发出此消息。要解决此问题，请参阅 Java 技术帮助网站上建议的解决方案，网址是 http://java.com/en/download/help/ikernel.jsp。

必须先配置 LoadRunner，并提供启用与 Diagnostics 组件的通信所必需的信息，才能从 LoadRunner 内访问 Diagnostics UI。有关配置 LoadRunner 以与 Diagnostics 集成的信息，请参阅第 23 章，“设置 HP LoadRunner 与 HP Diagnostics 集成”。

23

设置 HP LoadRunner 与 HP Diagnostics 集成

本章提供有关在负载测试运行和脱机分析中设置 HP LoadRunner 与 HP Diagnostics 集成的常规信息。

本章包括：

如何将 HP Diagnostics 与 LoadRunner 一起使用

LoadRunner diagnostics 模块以及与 HP Diagnostics 的集成可提供 LoadRunner 中详细的性能信息，帮助快速识别和确定 Siebel、Oracle、SAP、J2EE 以及 .NET 环境中的性能问题。

在 LoadRunner 中，由 HP Diagnostics 提供的 **J2EE/.NET diagnostics** 功能可允许您监控、分析以及解决 J2EE 和 .NET 应用程序测试环境中出现的复杂的性能问题。

一旦设置 LoadRunner 与 HP Diagnostics 的集成后，就可以以多种方式在 LoadRunner 中查看 HP Diagnostics 数据：

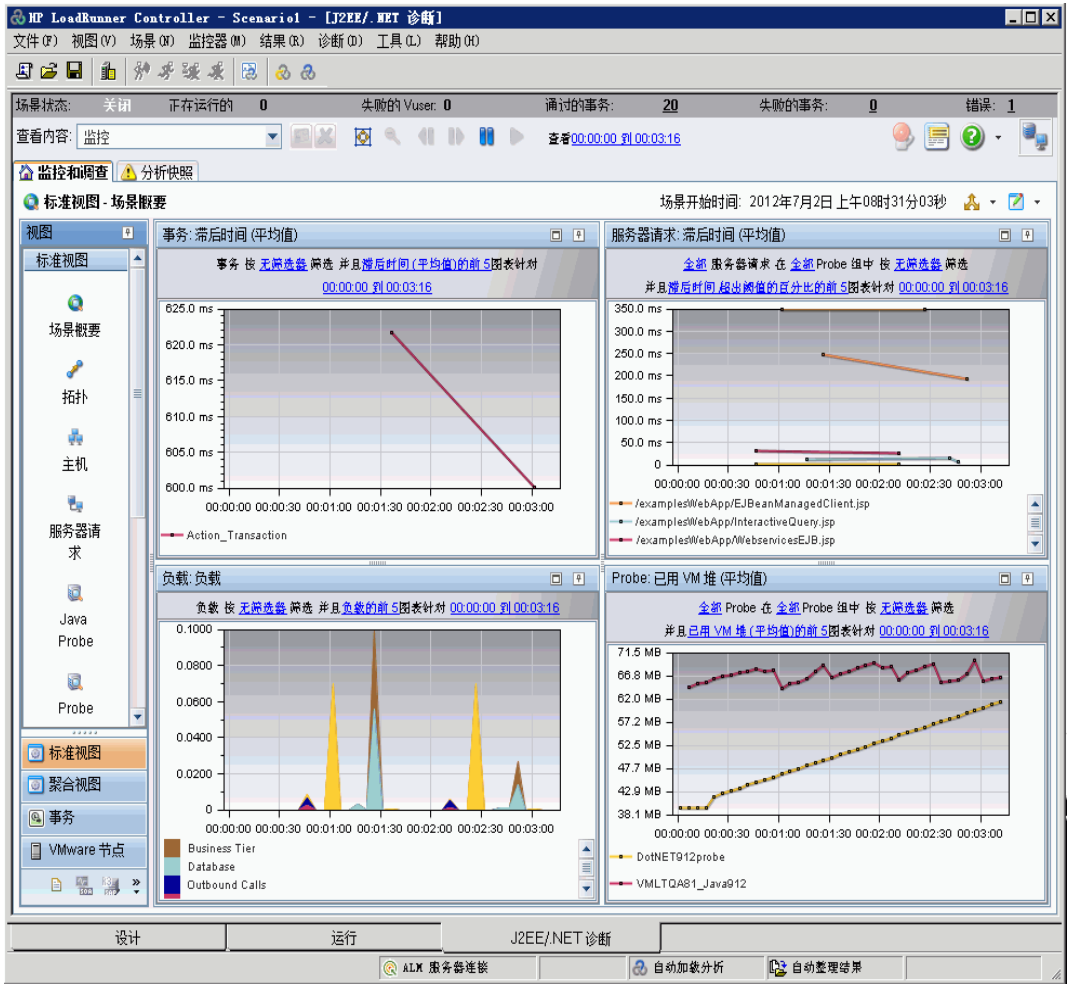
- LoadRunner 场景的 Diagnostics UI 视图
- 向下搜索到 Diagnostics UI 以获取有关事务的详细信息
- LoadRunner Analysis J2EE 和 .NET Diagnostics 图

LoadRunner 场景的 Diagnostics UI 视图

针对 LoadRunner 负载测试场景，可打开 HP Diagnostics UI 并获得整个场景的详细性能数据。

在 LoadRunner 中，可在场景窗口底部选择 J2EE/.NET 选项卡的 Diagnostics，此时会打开 HP Diagnostics UI，显示场景详细的诊断数据。在 HP Diagnostics UI 中，可以导航到其他视图，以识别、隔离、分析和解决运行中检测到的性能问题。

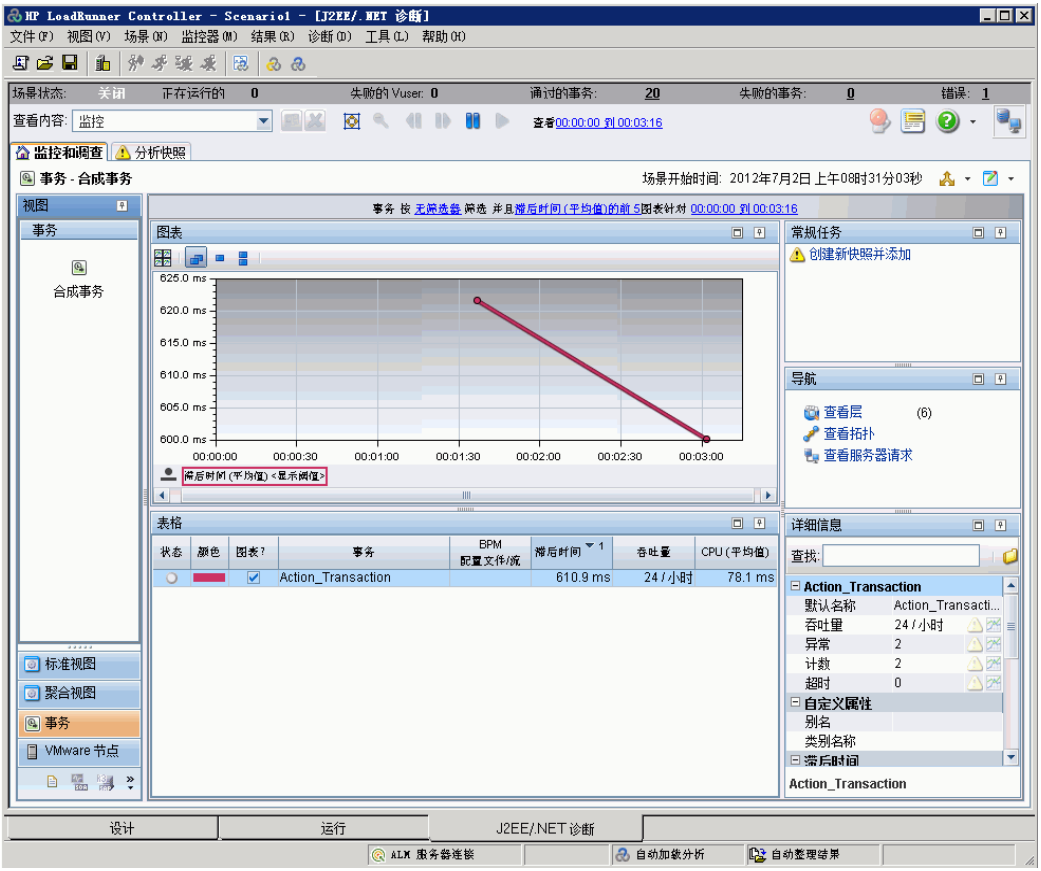
以下为 HP Diagnostics UI 的示例：



向下搜索到 Diagnostics UI 以获取有关事务的详细信息

在 LoadRunner 中，可向下搜索到特定事务的 HP Diagnostics UI，从而获取此事务的 Diagnostics 数据。

在 LoadRunner 中，选择其中一个事务图（如事务响应时间），并从此图向下搜索到 HP Diagnostics UI，此时会打开此 UI，显示事务视图。从此处可导航到 HP Diagnostics UI 中的其他视图，以解决与此事务相关的问题。



LoadRunner Analysis J2EE 和 .NET Diagnostics 图

在 LoadRunner Analysis 中，“J2EE 和 .NET Diagnostics 图”基于 HP Diagnostics 所提供的数据。通过这些图，您可以跟踪、计时和解决通过 J2EE、.NET web、应用程序以及数据库服务器的单个事务和服务器请求。此外，还可以快速确定有问题的 servlet 和 JDBC 调用，以最大化提高业务流程性能、扩展性和效率。

LoadRunner 中的 J2EE 和 .NET Diagnostics 图由两组图构成：J2EE 和 .NET Diagnostics 图以及 J2EE 和 .NET 服务器 Diagnostics 图。

有关在 LoadRunner 中查看 HP Diagnostics 数据的信息，请参阅《HP LoadRunner Controller 用户指南》和 《HP LoadRunner Analysis 用户指南》。

有关设置 LoadRunner 以与 HP Diagnostics 集成

从 LoadRunner 内部访问 Diagnostics UI 之前，必须为 LoadRunner 提供它与 HP Diagnostics 组件通信时所需的信息。

要实现从 LoadRunner 访问 Diagnostics UI，必须配置与 Diagnostics 服务器的集成。只需在首次配合使用 Diagnostics 和 LoadRunner 时指定 Diagnostics 服务器详细信息即可。有关配置 LoadRunner 以访问 Diagnostics 服务器的详细信息，请参阅 《HP LoadRunner Controller 用户指南》。

注意：指定 Diagnostics 服务器详细信息之前，请确保 LoadRunner Controller 已关闭。 Controller 打开后，可以查看 Diagnostics 配置设置，但不能更改这些设置。

配置 LoadRunner 场景以使用 HP Diagnostics

每次在负载测试方案中捕获 Diagnostics 度量时，必须针对方案配置 Diagnostics 参数，并选择将包括在方案中的探测器。可以从 LoadRunner Controller 配置 Diagnostics 的方案。有关详细信息，请参阅《HP LoadRunner Controller 用户指南》。

注意：如果保存了已配置 Diagnostics 设置的方案，则不必在每次运行该方案时重新配置 Diagnostics 参数。

选择要包括在脱机分析文件中的探测器度量

可以包括 Diagnostics 探测器度量数据，以供在 LoadRunner 脱机分析中使用。

默认情况下，脱机分析仅包含 “HeapUsed”、“GC Collections/sec”和 “GC time Spent in Collections”度量数据。要选择在脱机分析文件 （.eve 文件）中包括其他探测器度量，请使用 Diagnostics 配置文件 **etc/offline.xml**。

可在 Diagnostics Mediator 上配置 offline.xml 文件，以指定要在脱机分析中包括的 Diagnostics 探测器度量。应针对其探测器参与运行的所有 Mediator 配置此文件。

探测器度量只在 Java 探测器中提供，而 .NET 探测器中不提供。

注意：确保 Diagnostics 服务器和 LoadRunner 的时钟同步。

Diagnostics 探测器安装目录下的 **metrics.config** 中列出了所有可能的 Diagnostics 度量。尽管其中某些度量可能无法用于脱机分析，此文件仍然包含所有度量，具体取决于要监控的平台 （仅限于 Java 探测器）、应用程序服务器类型和版本。

通常，可在 Diagnostics UI “详细信息”窗格中的 Java 探测器下显示的所有度量均可在相关 Mediator 的 offline.xml 文件中使用。包含在 **metrics.config** 文件中的以下采集器无法在 offline.xml 文件中使用：“system”和“Mercury System”。

下面是 **offline.xml** 文件的一个示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<probeMetrics xmlns="http://hp.com/diagnostics/offline/1.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="offline.xsd">
<metric>
   <name>HeapUsed</name>
 </metric>
 <metric>
   <name>GC Collections/sec</name>
 </metric>
 <metric>
   <name>GC Time Spent in Collections</name>
 </metric>
</probeMetrics>
```

<metric> 元素用于指定匹配条件。可以对度量名称 (<name>)、类别 (<category>) 和采集器 (<collector>) 进行匹配。可以将 <name>、<category> 和 <collector> 元素组合起来，此时，所有元素都必须匹配。

匹配示例：

匹配并包括名为 “HeapUsed”的度量

```
<metric>
 <name>HeapUsed</name>
</metric>
```

匹配并包括显示有 “JVM” 采集器的所有度量。。

```
<metric>
  <collector>JVM</collector>
</metric>
```

匹配并包括名为 “HeapFree” 的由 “JVM” 采集器提供的度量。

```
<metric>
  <name>HeapFree</name>
  <collector>JVM</collector>
</metric>
```

默认情况下，匹配操作将在子字符串上执行，这意味着在 <name>、<category> 和 <collector> 中指定的文本需要存在于 （或包含于）实际度量名称、类别或采集器中。例如，在 <name> 中指定 “HeapFree” 文本，将匹配名称中包含 “HeapFree” 的所有度量 （例如 “MyHeapFreeMetric”、“YourHeapFreeMetric”）。

此外，还可以进一步通过 <metric> 的匹配属性（如 <metric match="regex">），为 <name>、 <category> 和 <collector> 指定正则表达式。请注意，正则表达式的使用成本很高，还会影响输出 .eve 文件所用的时间。

可按如下所示指定将所有探测器度量发送到 LoadRunner：

```
<metric match="regex">
  <collector>.*</collector>
  </metric>
```

注意：在 offline.xml 中添加过多度量将增加脱机 (.eve) 文件的大小，进而影响在 LoadRunner 分析应用程序中分析脱机文件所用的时间。

将每隔 15 秒自动检测并应用对 **offline.xml** 文件的更改。配置错误 （根据 **offline.xsd** 进行验证）将记录到 **server.log** 文件中。

改善大型脱机分析文件的传输

在 LoadRunner 或 Performance Center 测试运行期间，由 Diagnostics 生成的脱机分析文件 (.eve) 可能会变得非常大。运行结束后，这些文件将从 Diagnostics 服务器传输到 LoadRunner/Performance Center 控制器，以进行排序和分析。可以通过降低 .eve 文件的数据密度，来缩短用于传输和加载其中包含 Diagnostics 数据的脱机分析文件的时间。

可使用 Diagnostics 服务器中 **server.properties** 文件中的 **bucket.lr.offline.duration** 和 **bucket.lr.offline.sr.duration** 属性来增大聚合时间段（例如，从 5 秒增加到 15 秒）。利用这些属性，可确定要聚合的五秒趋势点的数目，以生成脱机分析样本。

LoadRunner Controller 的 Diagnostics UI 中的内存不足问题

如果在 LoadRunner UI 日志 (Mercury_Diagnostics_UI.log) 中发现 OutOfMemory 错误，此错误可能是由于对在 LoadRunner Controller 的 “Diagnostics” 选项卡中显示数据的 Diagnostics 小程序的内存限制而造成的。

要解决此问题，可以增加 Diagnostics 小程序的堆内存，方法是在 LoadRunner 系统的 OS 中定义系统环境变量。 APPCRITIC_MAX_MEM=256m 和 JAVA_TOOL_OPTIONS=-Xmx256m（如果还需要 256MB 的最大 VM 堆）。

24

设置 Performance Center 以使 Diagnostics

本章提供有关配置 Performance Center 以在负载测试中使用 HP Diagnostics 的常规信息。

本章包括：

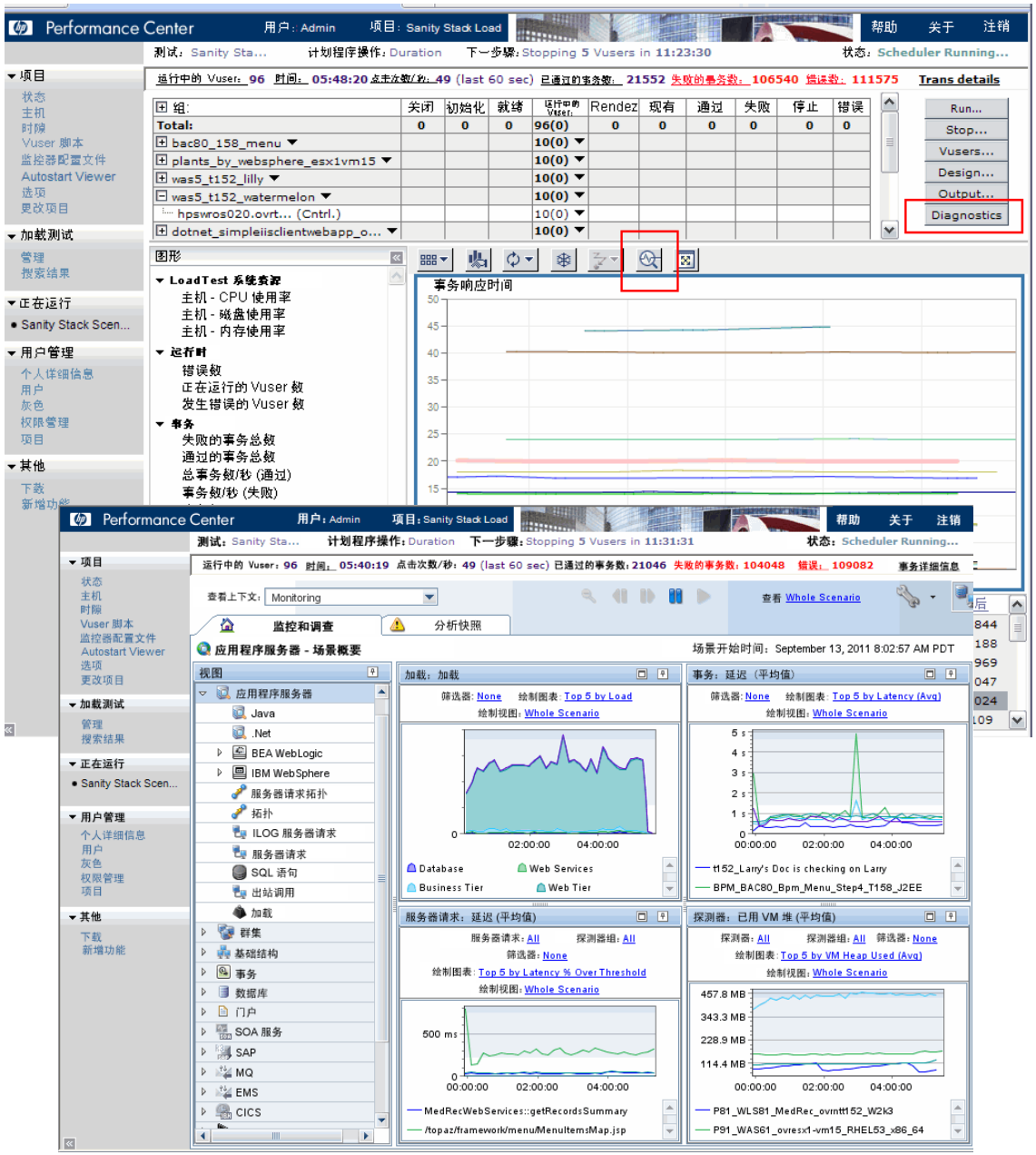
如何将 HP Diagnostics 与 Performance Center 一起使用

Performance Center Diagnostics 模块以及与 HP Diagnostics 的集成可提供 Performance Center 中详细的性能信息，帮助快速识别和确定 Siebel、Oracle、SAP、J2EE 以及 .NET 环境中的性能问题。

在 Performance Center 中，由 HP Diagnostics 提供的 **J2EE/.NET diagnostics** 功能可允许您监控、分析以及解决 J2EE 和 .NET 应用程序测试环境中出现的复杂的性能问题。

一旦设置 Performance Center 与 HP Diagnostics 的集成后，就可从 Performance Center 查看 HP Diagnostics 数据。

针对 Performance Center 负载测试运行，可向下搜索到 HP Diagnostics UI 并获取整个负载测试或特定事务的详细性能数据。负载测试运行之后，可使用 HP LoadRunner 分析对负载测试期间生成的脱机 Diagnostics 数据进行分析。


Performance Center
用户：Admin
项目：Sanity Stack Load
帮助
关于
注销
测试：Sanity Sta...
计划程序操作：Duration
下一步骤：Stopping 5 Vusers in 11:23:30
状态：Scheduler Running...
运行中的 Vuser：96 时间：05:48:20 点击次数/秒：49 (last 60 sec) 已通过的事务数：21552 失败的事务数：106540 错误数：111575
Trans details
Run...
Stop...
Vusers...
Design...
Output...
Diagnostics
事务响应时间
Performance Center
用户：Admin
项目：Sanity Stack Load
测试：Sanity Sta...
计划程序操作：Duration
下一步骤：Stopping 5 Vusers in 11:31:31
状态：Scheduler Running...
运行中的 Vuser：96 时间：05:40:19 点击次数/秒：49 (last 60 sec) 已通过的事务数：21046 失败的事务数：104048 错误：109082
查看上下文：Monitoring
监控和调查
分析快照
应用程序服务器 - 场景概要
场景开始时间：September 13, 2011 8:02:57 AM PDT
加载：加载
事务：延迟（平均值）
服务器请求：延迟（平均值）
探测器：已用 VM 堆（平均值）
Database
Web Services
Business Tier
Web Tier

有关设置 Performance Center 以使用 Diagnostics

Performance Center 和 Diagnostics 是集成产品，旨在通过协同工作以提供有助于了解和改进应用程序性能的信息。

要从 Performance Center 访问 Diagnostics，需在 Performance Center 中指定下列 Diagnostics 服务器详细信息。

- **服务器名。** Diagnostics 命令服务器主机的名称。
- **端口号。** Diagnostics 命令服务器使用的端口号。默认端口号是 2006。
- **登录名。**用于登录到 HP Diagnostics 的用户名。默认用户名是 admin。

  所指定的用户名应当具有 “查看”、“更改”和 “执行”权限。有关用户权限的详细信息，请参阅 “了解用户权限”（第 757 页）。

- **密码。**用于登录到 HP Diagnostics 的密码。默认密码是 admin。
- **通信**。 Performance Center 用于访问 Diagnostics 服务器的通信协议。

  如果通信协议是 HTTPS，则还需要执行其他一些配置步骤。有关所需步骤的详细信息，请参阅附录 C，“在组件之间启用 HTTPS”。

注意：只需在首次配合使用 Diagnostics 和 Performance Center 时指定这些详细信息即可。请在 Performance Center 管理站点的 Diagnostics 页面上提供这些信息。

配置 Performance Center 负载测试以使用 Diagnostics

每次在负载测试中捕获 Diagnostics 度量时，必须针对负载测试配置 Diagnostics 参数，并选择将包括在负载测试中的探测器。

有关如何配置 Performance Center 以与 Diagnostics 集成的完整说明，请参阅《HP Performance Center User's Guide》中有关 HP Diagnostics 与 Performance Center 集成的部分。

如果参与负载测试的 Performance Center 控制器与 Diagnostics 服务器之间存在防火墙，则必须将控制器和 Diagnostics 服务器配置为使用 MI 侦听器实用程序，以传输脱机分析文件。此外，还必须在 Performance Center 中指定 MI 侦听器计算机的 IP 地址。

同时，您还必须配置 Mediator 模式下的 Diagnostics 服务器，以使其跨防火墙工作。请参阅 “配置 Diagnostics 以在防火墙环境中工作”（第 631 页）。

在集成中启用 “监控服务器请求”功能的好处是，即使在下列情况中也能捕获到后端 VM 的调用：

- 探测器不捕获 RMI 调用。
- 无法捕获 RMI 调用 （可能是由于使用了不支持的应用程序容器）。
- 应用程序对多个 VM 间的通信使用了其他机制。

注意：如果将集成配置为监控服务器请求，此功能会在探测器上导致额外开销。

要检测 Diagnostics 组件间存在的任何连接问题，请使用可从 Performance Center 访问的系统运行状况监控器。

管理 Performance Center 脱机文件

默认情况下会保存 HP Performance Center 脱机文件。要管理脱机文件，必须配置 Mediator 模式下的 Diagnostics 服务器，才能删除这些文件。

为此，请在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 中，将 **distributor.offlinedelivery.preserveFiles** 属性设置为 true。设置为 true 后，此属性可以使存储在服务器数据目录中的特定于运行的 “脱机” 文件保留一段时间，此时间在服务器 **webserver.properties** 文件的 **facade.run_delete_delay** 属性中进行指定 （默认为 5 天）。

在此保留期内可整理运行。保留期结束后，会从系统中删除相关脱机文件。

第 IX 部分

附录

本节包括：

- Diagnostics 管理 UI
- 用户身份验证和授权
- 在组件之间启用 HTTPS
- 使用管理员的系统视图
- Diagnostics 数据管理
- Diagnostics 技术图
- 升级和修补程序安装说明
- HP Diagnostics 疑难解答
- 一般参考信息
- 数据导出

A

Diagnostics 管理 UI

本章提供有关如何访问和使用 Diagnostics 服务器管理 UI 的信息，此 UI 可用于配置 Diagnostics 属性以及管理 Diagnostics 软件。

本章包括：

访问 Diagnostics 管理 UI

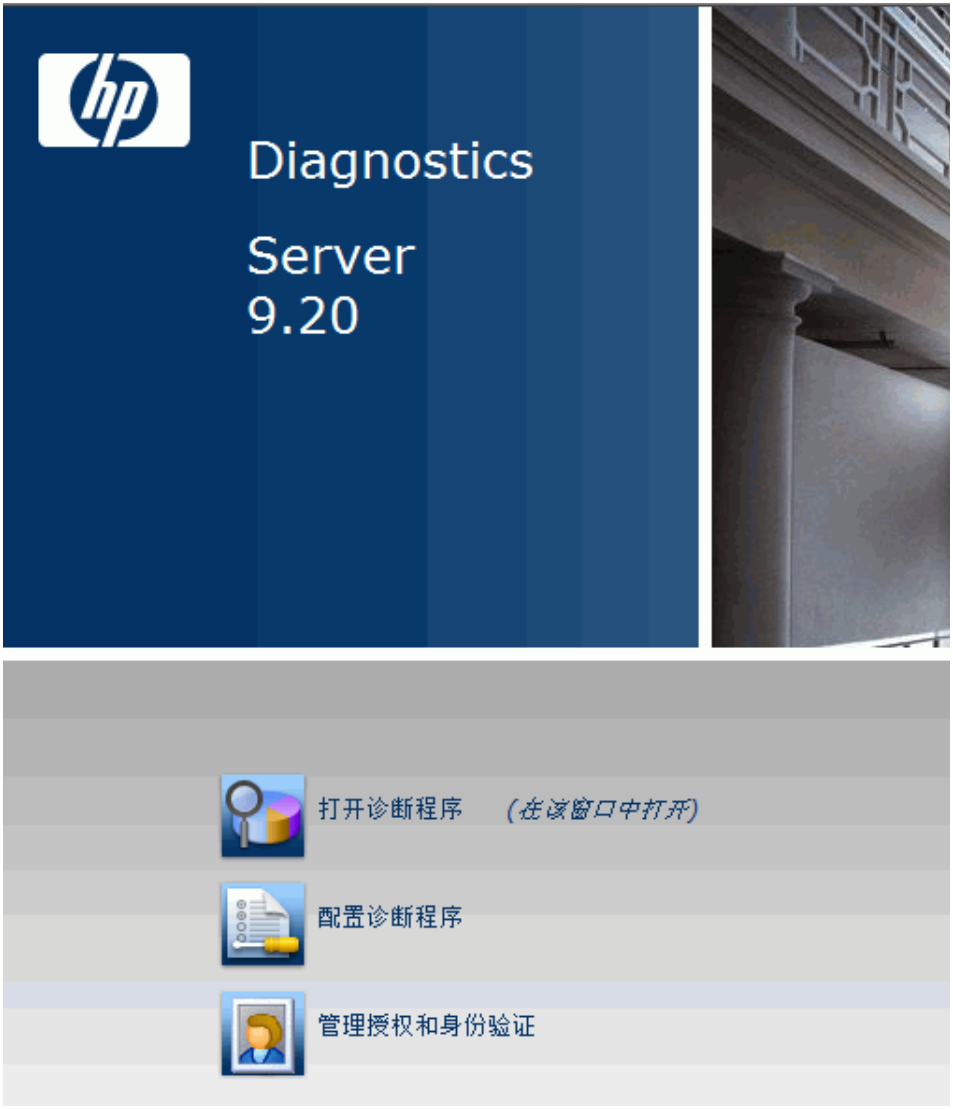
可以在主 Diagnostics UI 中直接查看 Diagnostics 配置信息、设置用户权限、配置 Diagnostics 设置以及管理 Diagnostics 软件。

要访问 Diagnostics 管理 UI，请执行以下操作：

1 通过在浏览器中导航到 http://<diagnostics 服务器主机 >:2006 或选择 “开始” > “所有程序” > “Diagnostics Server” > “Administration”，打开主 Diagnostics UI。URL 中的端口号 2006 是 Diagnostics 服务器的默认端口。如果将 Diagnostics 服务器配置为使用其他端口，请在 URL 中使用此端口号。

如果尚未登录到 Diagnostics 服务器，则会提示您输入用户名和密码。必须使用有效的用户名，且用户名必须具有 “查看” 和 “更改” 权限。有关有效用户名和权限的信息，请参阅附录 B，“用户身份验证和授权”。

此时，将在浏览器中打开主 Diagnostics UI。



该 UI 中包含如下所述的三个选项。

- **打开诊断程序。**打开 Diagnostics UI，可在其中查看由代理收集并报告给 Diagnostics 服务器的性能度量数据。这些性能度量数据将显示在各个 Diagnostics 视图中。

  有关 Diagnostics 视图的详细信息，请参阅联机帮助或 《HP Diagnostics User's Guide》。

➤ **配置诊断程序。**打开组件管理页面，此页面中包含有转到 Diagnostics 服务器配置页面的链接。

有关配置 Diagnostics 属性的详细信息，请参阅 “更改服务器配置”（第 751 页）。

➤ **管理授权和身份验证。**打开“用户管理”页面，可以在此页面中添加和维护安全信息以及特定用户的用户权限。有关安全性和用户权限的详细信息，请参阅“用户身份验证和授权”（第 755 页）。

2 在主 Diagnostics UI 中选择 “配置诊断程序” 。有关详细信息，请参阅 “使用 Diagnostics 管理 UI”（第 748 页）。

注意：

➤ 在输入有效凭据之前， Diagnostics 。

➤ 如果单击 “取消”，则浏览器中将显示以下错误消息：**访问被拒绝。必须指定有效的用户名和密码。**

➤ 如果输入了有效的用户名和密码，但没有适当的权限，则浏览器中将显示以下错误消息：**访问被拒绝。您不具有查看该屏幕所需的权限。**

要以其他用户身份登录，必须先关闭然后重新打开浏览器。

使用 Diagnostics 管理 UI

在 Diagnostics 管理 UI 中，可以查看 Diagnostics 配置信息，也可以设置一些属性值，这些属性用于控制 Diagnostics 服务器与其他 Diagnostics 组件的通信方式，以及 Diagnostics 服务器对从探测器接收的数据的处理方式。

为确保输入有效的属性值，建议您使用配置页面修改 Diagnostics 服务器属性，而不要直接编辑属性文件。

在主 Diagnostics UI 中选择 “配置诊断程序”，随即会显示 “组件”页面。

也可以在任意 Diagnostics 视图中选择 “维护”链接来访问此 “组件”页。

可以选择以下链接进入不同的 Diagnostics 管理页面。其中某些链接会转到信息页面，其他链接则允许您更改配置。

- **Registrar。**所有 Diagnostics 组件部署的集中列表。
- **Query。**查询 API，支持您以 HTML、XML 或 Java 对象的形式下载 Diagnostics 数据。在 /contrib 目录中提供了一个示例，说明如何使用 Diagnostics 查询 API 创建自定义控制面板。此外，也可参阅联机帮助主页和文档目录中提供的《HP Diagnsotics Data Model and Query API Guide》 (pdf)。

  如果在初始查询页面底部选择“Active Users”链接，可获取 Diagnostics 服务器在最近 60 秒内发现的活动用户的列表。同时，还可以查看 “Queries/sec”，此数值表示用户通过概要或趋势查询生成的负载量。
- **Security。**内置用户管理。请参阅 “了解 Diagnostics 服务器权限页”（第 760 页）。
- **Logging。**配置日志文件和日志记录详细信息。
- **Configuration。**配置 Diagnostics 服务器。有关其他配置页面的详细信息，请参阅 “更改服务器配置”（第 751 页）。
- **Files。**安装目录浏览器，用于上载和下载属性文件、日志文件和其他文件。
- **License。**许可证管理。有关详细信息，请参阅 “HP Diagnostics 许可”（第 77 页）。
- **Synchronize。**与 Business Service Management 同步 CI。可以强制执行硬同步（与 Business Service Management 完全同步）或软同步 （仅与 Business Service Management 同步新 CI）。
- **Thresholding。**设置阈值和警报的脚本语句。

“组件”页面上显示的组件均是常用组件。默认情况下已隐藏更高级的组件。

重要信息：如果没有 HP 软件客户支持代表的指导，请勿操作高级选项。

要显示高级选项，请执行以下操作：

➤ 在页面底部单击“显示高级选项”。

页面上的选项列表将更新为包括高级配置选项，并且链接将更改为“隐藏高级选项”。

此时将显示其他高级配置选项。

要隐藏高级选项，请执行以下操作：

➤ 在页面底部单击“隐藏高级选项”。

页面上的选项列表将更新为不再显示高级配置选项，并且链接将更改为“显示高级选项”。

更改服务器配置

在主 Diagnostics UI 中选择 “配置诊断程序”，然后选择 “Configuration”链接，以访问如下所示的 “配置”页面。

1 单击要更新其属性的页面的链接。对于 Diagnostics 服务器，您可以配置：

- Customer information
- Alert properties
- Component Communications
- Memory Diagnostics
- Online cache
- Logging

此页面上显示的配置选项均是常用配置选项。默认情况下已隐藏更高级的配置选项。选择 “显示高级选项”可查看更多配置选项。

重要信息：如果没有 HP 软件客户支持代表的指导，请勿操作高级选项。

---

2 例如，如果选择 “Customer Information”，则将显示如下页面。检查所显示的属性并进行更新。

3 完成所需更改后，单击 “提交”保存更改。如果不需要提交任何更改，可单击 “全部重置”将所有值重置为默认设置，或直接关闭对话框。

此时，页面顶部将显示一条消息，表明更改已保存。

注意：

- 对于大多数更新的属性，会显示一条消息，提醒您重新启动 Diagnostics 服务器。在重新启动 Diagnostics 服务器后，属性更改才会生效。

  如果还需要对 Diagnostics 服务器属性进行其他更改，则应在完成所有更改后再重新启动 Diagnostics 服务器。

  重新启动服务器将导致丢失少量数据（最多 6 分钟）。因此，应当在方便的时候重新启动服务器。

- 修改日志记录级别详细信息时不需要重新启动 Diagnostics 服务器，但可能需要一分钟的时间才能使更改生效。

B

用户身份验证和授权

提供有关 Diagnostics 身份验证和授权过程的信息，同时介绍如何创建和维护用户安全权限。

本章包括：

关于用户身份验证和授权

所有 Diagnostics 组件的用户身份验证和授权设置均在 Diagnostics Commander 中配置。

身份验证是验证个人身份的过程，而授权是验证一个已知用户是否有权 （权限或特权）执行特定操作的过程。角色是分配给用户的权限包。

您可以管理身份验证和授权，方法是创建和编辑用户名并授予用户权限，以便用户可以执行应用程序中其负责的相应功能。

连接到特定 Diagnostics 服务器的探测器的 Profiler （.NET Diagnostics Profiler 或 Java Diagnostics Profiler）的用户权限和特权也在 Diagnostics Commander 中定义。您可以向用户分配一组用于访问特定探测器组中 Profiler 的权限，同时再分配一组用于访问 Diagnostics 服务器的权限。

重要信息：

- 如果将代理仅安装为 Profiler （不连接任何 Diagnostics 服务器），则可以在代理中管理 Profiler 用户的身份验证和授权。
- 有关为仅安装为 Profiler 的 Java 代理 管理身份验证和授权的信息，请参阅 “Diagnostics Java Profiler 的身份验证和授权”（第 474 页）。
- 有关为仅安装为 Profiler 的 .NET 代理 管理身份验证和授权的信息，请参阅 “.NET Profiler 的身份验证和授权”（第 614 页）。

在查看任何 Diagnostics 数据，或者对 Diagnostics 配置或用户权限进行任何更改之前，必须使用具有适当权限的有效安全访问权限的用户名登录到 Diagnostics Commander。

在特定浏览器会话中登录到 Diagnostics 服务器之后，用户名仍然有效，直到浏览器会话结束为止。使用完 Diagnostics 后，请关闭浏览器以防止他人利用您的权限访问 Diagnostics。

了解用户权限

可以将以下级别权限分配给 Diagnostics 用户：

权限	描述
查看	用户可以在 UI 中查看 Diagnostics 数据。
执行	用户可以更改 UI 上的设置，如更改阈值或添加注释。在 Profiler 上，此权限允许用户执行垃圾收集和清除 Profiler 保留的性能数据。
更改	用户可以访问“配置诊断程序”菜单，改变组件配置和维护用户信息。在 Profiler 上，此权限允许用户运行具有潜在风险的操作，如进行堆转储或更改设备。

注意：

- 权限级别 “rhttpout” 和 “系统” 仅供内部使用。“rhttpout” 用于向用户授予访问 rhttp/URL 的权限，以便用户对分布式服务器进行远程管理。
- “系统” 是内部权限，通常只有 **mercury** 特殊用户才能享有此权限。它是允许 Diagnostics 组件相互对话的权限；例如，探测器用于进行 Diagnostics 服务器注册的权限。要查看系统运行状况，就需要 “系统” 权限。

每个权限级别均独立存在。各个级别之间没有权限继承关系。您必须向用户授予执行功能所必需的所有级别权限。

例如，用户必须获得 “查看” 和 “执行” 权限，才能够对阈值进行更改。只获得 “执行” 权限的用户名没有多大用处，因为用户无法利用此权限来查看有权进行更改的 UI。

有关向用户分配权限的信息，请参阅 “跨 Diagnostics 部署分配权限”（第 770 页）。

了解角色

除了用户 / 权限分配以外，还可以将权限分配给角色，然后将这些角色分配给用户。这样，就可以更加方便地管理多个用户：将新用户添加到 Diagnostics 后，只需执行用户 / 角色分配即可。如果用户设置有不同的权限，可访问特定探测器组的 Diagnostics 服务器和 Profiler，则此特性就十分有用。

请考虑以下示例：

两个开发团队 （Dev1 和 Dev2），需要他们所拥有的代理系统上 Profiler 的所有权限 （查看、执行、更改），以及非他们拥有的代理系统上的查看权限。两个团队都应当拥有 UI 的查看和执行权限。

因此，必须创建以下角色：

角色	权限
企业 （访问 UI）	[DevUI] = 查看、执行
Dev1 探测器组	[Dev1All] = 查看、执行、更改 [Dev2View] = 查看
Dev2 探测器组	[Dev2All] = 查看、执行、更改 [Dev1View] = 查看

请注意，角色需要包含在方括号中以区别于其他用户。例如，如果将 Dev1 团队的新用户添加到 Diagnostics 中，则此用户应属于角色：
[DevUI],[Dev1All],[Dev1View]。

使用默认用户名访问 Diagnostics

下面是为 Diagnostics 定义的默认用户名：

默认用户名	权限	描述
user	查看	只能从 UI 查看数据。
superuser	查看、执行	可以查看数据，更改阈值以及从 UI 创建警报和注释。
admin	查看、更改、执行、系统	可以查看数据，更改阈值以及从 UI 创建警报和注释。同时，还可以配置组件和维护用户信息。

您可以使用这些默认用户名访问 Diagnostics 功能。

默认用户名的密码与用户名相同。例如，用户名 admin 的密码就是 admin。

您可以修改默认用户名的密码或权限以满足您的需要，还可以定义新用户名以控制用户对 Diagnostics 的访问。

重要信息：默认的用户 **mercury** 和 **bac** 仅供内部使用，不应对其执行修改。这些用户用于组件之间的内部通信。

了解 Diagnostics 服务器权限页

您可以在 “权限”页中管理用户和分配用户权限。

本节包括：

访问 “权限” 页

您可以通过选择“管理授权和身份验证”，从主 Diagnostics UI 访问“权限”页。

或者，通过在任意 Diagnostics 视图中选择 “维护” 链接，然后选择安全链接，访问此 “权限” 页。

在查看任何 Diagnostics 数据，或者对 Diagnostics 配置或用户权限进行任何更改之前，必须使用具有适当权限的有效安全访问权限的用户名登录到 Diagnostics Commander。

“权限”页打开后，如果尚未登录到 Diagnostics 服务器，系统可能会提示您输入用户名和密码。必须至少具有 “查看”权限，才能查看权限和修改密码。要添加或删除用户，或更新用户权限，必须同时具有 “查看”和 “更改”权限。

注意：

- 在输入有效的详细信息之前， Diagnostics 会继续提示输入用户名和密码。
- 如果单击 “取消”，则浏览器中将显示以下错误消息：**访问被拒绝。必须指定有效的用户名和密码。**
- 如果输入了有效的用户名和密码，但没有适当的权限，则浏览器中将显示以下错误消息：**访问被拒绝。您不具有查看该屏幕所需的权限。**

“权限”页概览

以下屏幕是 Diagnostics 服务器 “权限”页的一个示例：

“权限”页分为以下三个部分：

- **企业 Diagnostics 权限。**在此部分中，可以管理 Diagnostics 用户，以及跨整个 Diagnostics 部署 （包括 Diagnostics 服务器和代理）分配权限。

  默认情况下，如果用户得到授权，可访问特定 Diagnostics 服务器，则可以利用相同授权 （和权限）访问连接到服务器的所有探测器。

注意：Diagnostics 有一个集中式权限系统，可在其中为用户设置权限。这些权限将应用到所有连接到 Diagnostics 系统的分布式服务器和探测器。但是，由于每隔 5 分钟才会将权限推送到分布式组件，因此权限更改不会立即生效。

- **控制已连接至以下对象的探测器 <Diagnostics_commander_server>。**在此部分中，可以为访问探测器 Profiler 的用户分配权限。您可以向用户分配一组用于访问特定探测器组中 Profiler 的权限，同时再分配一组用于访问 Diagnostics Commander 服务器的权限。
- **加密内部 Diagnostics 密码。**可以访问 EncryptPassword 实用程序以加密密码。

企业和应用程序权限

除了在 “权限”页上设置的企业和探测器级别权限以外，您还可以设置应用程序级别权限。应用程序权限在初始 “应用程序”窗口的 Diagnostics UI 中设置。有关设置应用程序权限的详细信息，请参阅 《HP Diagnostics User's Guide》。

三组权限如下：

- 企业
  - **查看。**用户可以在 Diagnostics UI 中查看性能数据。
  - **执行。**用户可以更改阈值，并添加注释和创建应用程序。
  - **更改。**用户具有对系统的完整管理访问权限；例如，可以创建用户。

- 每个探测器组 （适用于 Profiler）
  - **查看。**用户可以查看由 Profiler 收集的性能数据。
  - **执行。**用户可以运行垃圾收集和清除 Profiler 保留的性能数据。
  - **更改。**用户可以运行诸如进行堆转储或更改插桩这样的操作。
- 应用程序
  - **查看。**用户可以查看应用程序和编辑实体属性 （需要将 “企业”权限设置为 “更改”）。
  - **修改。**用户可以删除、重命名、修改应用程序，还可以在应用程序中添加或删除实体。
  - **编辑屏幕。**用户可以在 “应用程序概述”屏幕中执行编辑操作。

注意：权限不是包含性的 （“执行”不包括 “查看”）。

区域和操作	企业权限			应用程序权限		
	查看	执行	更改	查看	修改	编辑屏幕
Diagnostics UI						
在 UI 中查看 Diagnostics 数据	X					
更改自定义属性		X				
在 UI 中设置阈值		X				
在 UI 中创建 / 修改 / 删除注释		X				
在 UI 中创建警报规则		X				
配置 Diagnostics 页			X			
配置业务事务		X				

区域和操作	企业权限			应用程序权限		
	查看	执行	更改	查看	修改	编辑屏幕
查看系统运行状况 **注意：**要查看系统运行状况，还需要“系统”企业权限。			X			
管理其他用户的授权和身份验证			X			
访问维护页			X			
使用事件	X					
使用自定义视图	X					
Profiler UI						
在 Profiler 中执行垃圾收集		X				
清除 Profiler 中的性能数据		X				
查看 Profiler 中的 Diagnostics 数据	X					
执行堆转储 （内存和分配分析）			X			
更改配置			X			
用户定义的应用程序						
创建应用程序		X				
删除应用程序		X			X	
重命名应用程序		X			X	
更改应用程序路径		X			X	

区域和操作	企业权限			应用程序权限		
	查看	执行	更改	查看	修改	编辑屏幕
修改应用程序权限		X			X	
编辑自定义应用程序屏幕	X					X
向应用程序添加实体	X				X	
从应用程序中删除实体	X				X	
编辑实体属性 （阈值、注释等）		X		X		
查看应用程序	X			X		
自动搜寻到的应用程序、事务应用程序或整个企业						
不允许创建、删除和更改应用程序路径						
修改应用程序权限		X			X	
编辑应用程序屏幕	X					X
向应用程序添加实体	X				X	
从应用程序中删除实体	X				X	
编辑实体属性		X		X		
查看应用程序	X			X		

创建、编辑和删除用户

同时具有 “查看” 和 “更改” 权限的用户可以新建用户、编辑现有用户的密码，或者删除用户。仅具有 “查看” 权限的用户可以维护自己的密码。

要创建新用户，请执行以下操作：

1. 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。
2. 在 “权限” 页上，单击 “用户管理”，打开 “用户管理” 页。
3. 在 “用户管理” 页上，单击 “创建用户”。
4. 在 “新建用户名” 框中，键入新用户的用户名，然后单击 “确定”。此时，新用户将出现在用户名列表中。

注意：由于浏览器在处理基本身份验证时存在一定限制，用户名和密码必须仅包含英文字符。

---

5. 在 “更改密码” 下面的 “密码” 框中，键入新用户的密码，然后在 “确认密码” 框中重新键入密码以进行确认。
6. 在 “< 当前用户 > 的密码” 框中，键入当前登录的用户的密码。
7. 单击 “保存更改”。

默认情况下，新用户具有 “查看” 权限。有关更改分配给用户的权限的信息，请参阅 “跨 Diagnostics 部署分配权限”（第 770 页）。

要分配角色，请执行以下操作：

1 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。

2 在 “用户管理” 页上，为用户分配角色。确保角色位于方括号中 ([aRole])。可以用逗号分隔角色 ([Role1],[Role2])。

注意：必须在 “企业” 和 / 或 “每个探测器组” 对话框下面为角色设置权限 （请参阅 “跨 Diagnostics 部署分配权限”（第 770 页）和 “为探测器组分配权限”（第 771 页））。

要删除用户，请执行以下操作：

1 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。

2 在 “权限” 页上，单击 “用户管理”，打开 “用户管理” 页。

3 在 “用户管理” 页的 “< 当前用户 > 的密码” 框中，键入当前登录的用户的密码。

4 单击与要删除的用户对应的红色 X （删除用户）按钮。

5 此时将打开一个消息框，询问您是否要删除所选用户。

单击 “OK” 删除用户。

要更改用户密码 （如果有 “查看” 和 “更改” 权限），请执行以下操作：

1 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。

2 在 “权限” 页上，单击 “用户管理”，打开 “用户管理” 页。

3 在 “用户管理” 页上找到代表相关用户的行，然后在 “密码” 和 “确认密码” 框中键入新密码。

4 在 “< 当前用户 > 的密码” 框中，键入当前登录的用户的密码。

5 单击 “保存更改”，保存对不同用户名所做的全部更改。

要更改自己的密码 （如果只有 “查看” 权限），请执行以下操作：

1 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。

2 在 “权限” 页上，单击 “用户管理”，打开 “用户管理” 页。

3 在 “用户管理” 页上的 “密码” 和 “确认密码” 框中键入新密码。

4 在 “旧密码” 框中，键入旧密码。

5 单击 “保存更改”。

跨 Diagnostics 部署分配权限

同时具有 “查看” 和 “更改” 权限的用户可以跨整个 Diagnostics 部署授予用户权限。

注意：有关可分配给 Diagnostics 用户的用户权限的说明，请参阅 “了解用户权限”（第 757 页）。

要跨整个企业分配用户权限，请执行以下操作：

1 访问 Diagnostics 服务器的 “权限” 页，如 “访问 “权限” 页”（第 761 页）所述。

2 在 “权限” 页上，单击 “编辑企业权限”，打开 “编辑企业权限” 页。

“编辑企业权限” 页是可编辑页面，您可以通过此页面修改用户权限。

3 找到要修改其权限的用户名。

重要信息：在 “用户管理” 页上添加用户，如 “创建、编辑和删除用户”（第 768 页）所述。

---

4 将权限作为以逗号分隔的值添加到用户名。

例如，如果用名称 **newuser** 定义用户，并且要向此用户分配查看和执行权限，则必须找到 **newuser** 并编辑对应的行，使其按如下方式显示：

```
newuser = view,execute
```

“编辑企业权限” 页还包括一组默认用户。“使用默认用户名访问 Diagnostics”（第 759 页）中对这些用户进行了介绍。可以修改这些默认用户的权限。

为探测器组分配权限

同时具有 “查看” 和 “更改” 权限的用户可以授予用户权限，以访问属于特定探测器组的探测器 Profiler。

默认情况下，如果用户得到授权，可访问特定 Diagnostics 服务器，则可以利用相同授权 （和权限）访问连接到服务器的所有探测器 Profiler。

但是，您可以为用户分配一组不同的权限，以访问与 Diagnostics 服务器所拥有的探测器组不同的其他探测器组。

注意：有关可分配给 Diagnostics 用户的用户权限的说明，请参阅 “了解用户权限”（第 757 页）。

可以为每个探测器组单独修改用户权限，也可以修改 “权限”模板，此模板可用于为所有将来会添加到系统的探测器组定义用户权限设置。

注意：保存更改 1 分钟后，用户和权限设置才会生效。

在每个探测器组中有三组具有特定权限的默认用户组。可以选择取消这些组的注释或修改其权限。默认情况下，所有探测器组中均定义了以下用户组：

用户组	权限
any_diagnostics_admin	此组代表在 Diagnostics 服务器上具有管理 （更改）权限的任何用户。默认情况下，属于此类别且没有任何其他预定义权限设置的用户对连接到服务器的所有探测器具有管理权限。
any_diagnostics_superuser	此组代表在 Diagnostics 服务器上拥有超级用户 （执行）权限的任何用户。默认情况下，属于此类别且没有任何其他预定义权限设置的用户对连接到服务器的所有探测器具有执行权限。
any_diagnostics_user	此组代表在 Diagnostics 服务器上拥有用户 （查看）权限的任何用户。默认情况下，属于此类别且没有任何其他预定义权限设置的用户对连接到服务器的所有探测器具有查看权限。

要分配用户权限以访问特定探测器组，请执行以下操作：

1 访问 Diagnostics 服务器的 “权限”页，如 “访问 “权限”页”（第 761 页）所述。

2 在权限页面的 “控制已连接至以下对象的探测器 <Diagnostics commander 服务器 >”部分中，单击 “编辑 < 探测器组名称 >”。

此时将打开“编辑权限”页。此为可编辑页面，您可以通过此页面修改用户权限。

3 输入要向其分配唯一权限的用户名，并将权限作为以逗号分隔的值添加到用户名。

例如，如果用名称 **newuser** 定义用户，并且要向此用户分配对特定探测器组的查看和执行权限，请输入以下行：

```
newuser = view,execute
```

要使用 “权限”模板分配用户权限，请执行以下操作：

1 访问 Diagnostics 服务器的“权限”页，如“访问“权限”页”（第 761 页）所述。

2 在权限页面的 “控制已连接至以下对象的探测器 <Diagnostics commander 服务器 >”部分中，单击 “编辑权限模板”。

此时将打开 “编辑模板权限”页。此为可编辑页面，您可以通过此页面修改用户权限。

3 输入要向其分配唯一权限的用户名，并将权限作为以逗号分隔的值添加到用户名。

例如，如果用名称 **newuser** 定义用户，并且要向此用户分配对特定探测器组的查看和执行权限，请输入以下行：

```
newuser = view,execute
```

您也可以修改模板中定义的单个用户组设置，或取消对设置的注释。

重要信息：所有将来会连接到 Diagnostics 服务器的探测器组将从此 “权限”模板中继承用户权限设置。

集成式 HP 软件产品的用户身份验证和授权

Diagnostics 可以与其他 HP 软件应用程序 （Business Service Management、Performance Center 或 LoadRunner）集成。本节描述如何对这些集成式产品的用户进行身份验证和授权，包括以下部分：

- “Business Service Management 用户的身份验证和授权”（第 774 页）
- “Performance Center 和 LoadRunner 用户的身份验证和授权”（第 775 页）

Business Service Management 用户的身份验证和授权

在 Business Service Management 中，可以定义 Diagnostics 的用户权限。有关详细信息，请参阅“在 Business Service Management 中为 Diagnostics 用户分配权限”（第 704 页）。

当现有或新 Business Service Management 用户从
Business Service Management 打开 Diagnostics 时，即会从 Business Service Management 会话中获取其权限并复制到 Diagnostics 权限系统中（在 SaaS 客户下面，如果相关）。

只有当用户打开 Diagnostics 时，才会应用 Business Service Management 用户权限的更新 （如果 Diagnostics 已打开，则在先关闭再重新打开 Diagnostics 之前将检测不到更改）。

Business Service Management 密码不会发送到 Diagnostics 中 - Diagnostics 信任成功的 Business Service Management 登录。

如果 Business Service Management 用户的权限发生更改，则在用户重新打开 Diagnostics 之前将不会应用更改。

如果删除了 Business Service Management 用户，建议您手动从 Diagnostics 中删除其权限。有关详细信息，请参阅 “创建、编辑和删除用户”（第 768 页）。

注意：Diagnostics 服务器可能会需要 5 分钟来检测用户的权限更改。

Performance Center 和 LoadRunner 用户的身份验证和授权

设置 LoadRunner 或 Performance Center 以与 Diagnostics 集成时，可以在 LoadRunner / Performance Center 中指定 Diagnostics 服务器详细信息。这些详细信息包括登录到 HP Diagnostics 时所使用的用户名和密码。

从 LoadRunner 或 Performance Center 访问 Diagnostics 时，可以使用集成设置期间指定的相同用户名和密码登录到 Diagnostics。

因此，从 LoadRunner 或 Performance Center 内部访问 Diagnostics 用户将拥有与集成设置期间指定的用户名相关联的权限。

跟踪用户管理活动

每次用户进入 Diagnostics 服务器用户管理页时，发生的所有活动均记录在以下日志文件中：**<Diagnostics 服务器安装目录 >\log\useradmin.log**。

记录到此文件中的数据包括每个操作的执行日期和时间、操作的描述，以及执行操作的用户名。

要查看日志文件，请执行以下操作：

1 通过以下方式之一打开 Diagnostics 服务器管理页：

- 选择“开始”>“所有程序”>“HP Diagnostics Server”>“Administration”。
- 在浏览器中导航到 http://<diagnostics 服务器主机 >:2006。URL 中的端口号 **2006** 是 Diagnostics 服务器的默认端口。如果将 Diagnostics 服务器配置为使用备用端口，请在 URL 中使用此端口号。

此时将打开 Diagnostics UI 主页。

2 单击 “配置诊断程序”。

3 如果尚未登录到 Diagnostics 服务器，则会提示您输入用户名和密码。必须使用有效的用户名，且用户名必须具有 “查看” 和 “更改” 权限。有关有效用户名和权限的信息，请参阅 “了解用户权限”（第 757 页）。

注意：

- 在输入有效凭据之前， Diagnostics 会一直提示输入用户名和密码。
- 如果单击 “取消”，则浏览器中将显示以下错误消息：**访问被拒绝。必须指定有效的用户名和密码。**
- 如果输入了有效的用户名和密码，但没有适当的权限，则浏览器中将显示以下错误消息：**访问被拒绝。您不具有查看该屏幕所需的权限。**

此时将打开 “Diagnostics 服务器组件” 页。

4 单击 “日志记录”。此时将打开日志记录页。

5 单击 “查看日志文件”。此时将出现日志文件列表。

6 单击 **<Diagnostics 服务器安装目录 >\log\useradmin.log** 链接。

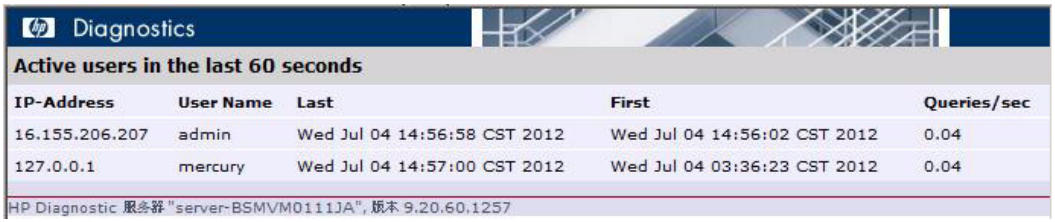
日志文件将显示在页面底部。

活动用户的列表

您可以获取 Diagnostics Server 在最后 60 秒内查看到的活动用户的列表。同时，还可以查看 “Queries/sec”，此数值表示用户通过概要或趋势查询生成的负载量。

在主 Diagnostics UI 中选择 “配置诊断程序”，随即会显示 “组件” 页。（也可以在任意 Diagnostics 视图中选择 “维护” 链接来访问此 “组件” 页）。

选择“query”链接，然后在页面底部选择“Active Users”链接以显示活动用户的列表。



如果发现查询负载过高，则可以对 UI 执行的查询配置限制。在服务器上使用 **ui.properties** 文件设置属性，控制 UI 查询对服务器的更新频率。

将 Diagnostics 配置为使用 JAAS

可以将 Diagnostics 配置为使用 JAAS（Java 身份验证和授权服务）以进行用户身份验证。如果启用了 JAAS，则可以通过配置的 JAAS 可插入身份验证模块 (LoginModule)，验证访问 UI 时在登录对话框中输入的用户名和密码。

注意：只有在 Diagnostics Commander 服务器上才可使用 JAAS 支持。

必须通过取消注释以下两行内容，在 **< 安装目录 >/etc/server.properties** 文件中启用 JASS：

```
authentication.jaas.config.file=jaas.configuration
authentication.jaas.realm=Diagnostics
```

authentication.jaas.config.file 属性指定定义 LoginModules 的配置文件（相对于 etc 目录），而 **authentication.jaas.realm** 指定配置文件中使用的条目。

jaas.configuration 示例：

```
Diagnostics
{
 com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule sufficient
        ip="1.2.3.4";

 com.mercury.diagnostics.server.jaas.spi.LDAPLoginModule sufficient
    useSSL="true"
    serverCertificate="etc/ldap.keystore"
    providerURL="ldap://ldap.yourdomain.com:636"
    baseDN="ou=People,o=yourdomain.com";

};
```

有关 JAAS 配置文件的详细信息，请参阅 Oracle 有关 JAAS 的文档以及 Oracle 有关 javax.security.auth.login.Configuration 的 javadoc。

注意：在验证用户名和密码时，*首先*会使用由 Diagnostics 通过 “管理授权和身份验证” Web 页创建的用户。只有当此身份验证失败时，才会执行 JAAS 身份验证。

Diagnostics 提供以下 LoginModule：

- **LDAP。**(com.mercury.diagnostics.server.jaas.spi.LDAPLoginModule)，允许针对 LDAP 服务器进行身份验证。
- **SiteMinder。**(com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule)，允许针对 SiteMinder 环境进行身份验证。

注意：

- 对 server.properties 和 / 或 jaas.configuration 文件做出任何更改之后，必须重新启动 Diagnostics Commander 服务器。
- 如果使用同时用于其他应用程序 （如 LDAP）的 JAAS 身份验证提供程序，则建议打开 HTTPS 以访问 Diagnostics UI。
- 使用 JAAS 身份验证提供程序时，用户帐户由身份验证源维护。有关用户名语法的详细信息，请咨询管理员。
- 可以使用权限页维护已验证用户权限的后续授权。如果配置了使用角色的合适 LoginModule，则还可以使用角色。在这种情况下，可以使用现有角色，也可以新建角色。

配置 LDAP 身份验证

要在 Diagnostics 中配置 LDAP 身份验证，必须首先将 Diagnostics 配置为使用 JAAS （请参阅 “将 Diagnostics 配置为使用 JAAS”（第 778 页）），然后在 Diagnostics Commander 服务器上配置 LDAPLoginModule。

此处内容描述了 Diagnostics 通常如何处理权限以及如何处理 LDAP 身份验证（请见下一页中的流程图）：

1. 接受用户提供的用户名和密码。
2. 将此用户名和密码与缓存的用户名和密码进行比较。
   a. 如果此用户名在缓存中存在且密码匹配，则将根据使用相同凭据的上一次登录尝试来确定此次登录的成败。
   b. 否则，继续到下一个步骤。

3 将用户名和密码与企业 Diagnostics 权限 (EDP) 进行比较。

   a 如果此用户名在 EDP 中存在且密码匹配，则转至步骤 6。

   b 否则，继续到下一个步骤。

4 循环通过 jaas.configuration 中配置的所有 LDAP 登录模块。

   a 如果 LoginModule 验证了用户名 / 密码，则转至步骤 6。

   b 否则，继续到下一个登录模块。

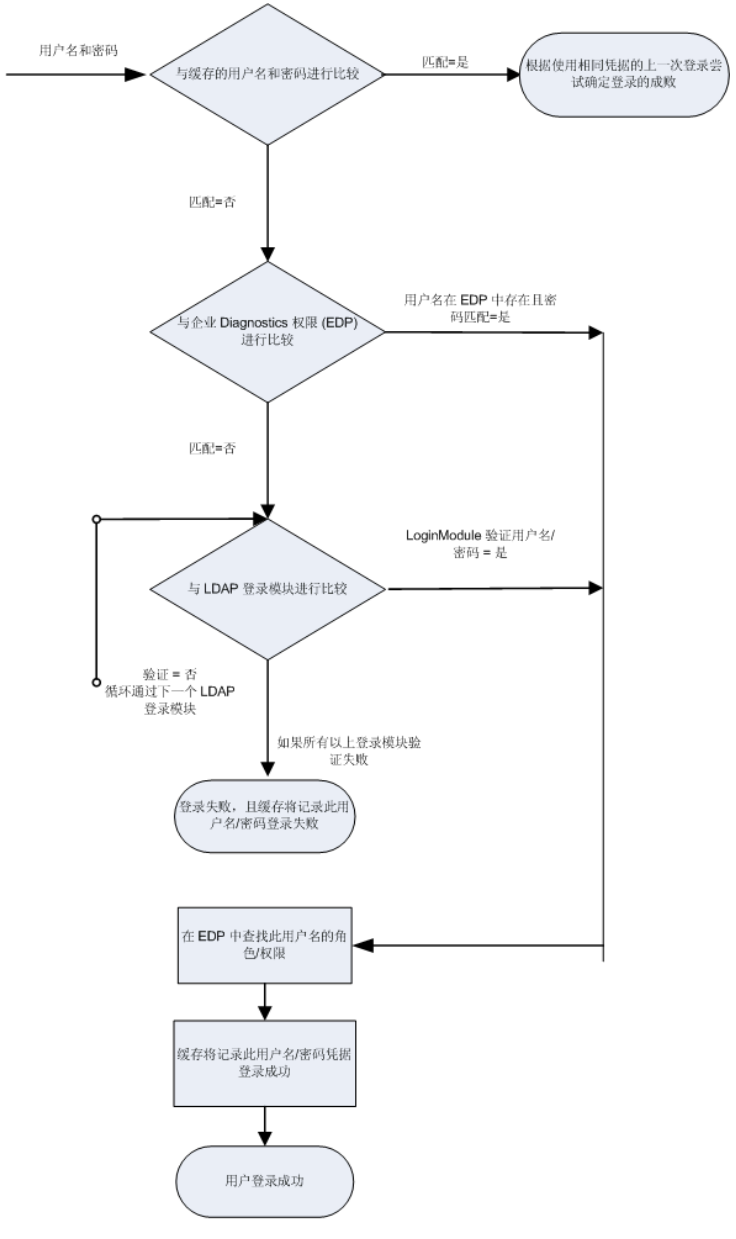
5 如果所有以上登录模块验证失败，则登录失败，且缓存将记录此用户名 / 密码登录失败。

6 在 EDP 中查找此用户名的角色 / 权限。

7 缓存将记录此用户名 / 密码凭据登录成功。

8 返回结果，显示用户登录成功。

Diagnostics LDAP 身份验证流程



使用特定于 LDAP 服务器的选项值，在 **< 安装目录 >/etc/ jaas.configuration** 中编辑 Diagnostics JAAS 领域 （应用程序）块。

LDAPLoginModule 可用于简单或高级模式中。

- 在两种模式中：
  - 可以配置 SSL 和服务器证书。
  - 可以配置角色。
  - 可以请求调试信息。
- 在简单模式中：
  - 可以完成匿名目录搜索。
  - 使用预定义搜索筛选器。
  - 可以配置唯一基础 DN （可分辨名称）。
  - 引用不可用。
- 在高级模式中：
  - 进行目录搜索必须提供凭据。
  - 可以使用 RFC 2254 兼容的搜索筛选器。
  - 可以配置多个基础 DN。
  - 引用可用。

  您可以在 Active Directory 系统中启动 **ldp.exe** 实用程序，以测试设置。

下表列出了 LDAPLoginModule 通用属性 （针对所有模式）：

属性	描述和示例	值
authType	指定验证用户身份时要使用的安全级别。	“simple”（默认） “none” “strong”
debug	指定是否将调试信息写入 server.log。	“false”（默认） “true”
defaultRoles	要分配给每个已验证用户的角色列表，用逗号分隔。 示例：“SuperUsers”	
roleAttributes	用户的 DN 属性列表，其值将用作用户的角色，用逗号分隔。如果还设置了 **defaultRoles**，则生成的角色将是 **defaultRoles** 和 **roleAttributes** 的组合。 示例： “employeeType,hpJobFunction”	“roles”（默认）
serverCertificate	包含 LDAP 服务器证书的信任存储文件的路径。路径可以绝对或相对于服务器安装目录。有关生成密钥库的信息，请参阅附录 C，“在组件之间启用 HTTPS”。 示例：“etc/jssecacerts”	
useSSL	如果设置为 **True**，则使用 SSL 连接 LDAP 服务器。	“false”（默认） “true”

下表列出了 LDAPLoginModule 简单模式属性：

属性	**描述和示例**	**值**
allowAnonymous	如果设置为 **True**，则允许 LDAP 服务器进行匿名搜索，以检索用户的主 DN。要使此属性生效，还需将 **searchFirst** 属性设置为 **True**。	“false”（默认） “true”
baseDN	用于构造主 DN。如果允许匿名搜索，则还可用于指定基础 DN 以搜索其中的用户。（必需） 示例： “OU=Users,DC=your,DC=ldap,DC=domain,DC=com”	
providerURL	LDAP 服务器的 URL。用于身份验证。如果允许匿名搜索，则还可用于搜索用户。（必需） 示例： “ldap://your.ldap.domain.com:389” SSL 示例： “ldaps://yourldap.domain.com:636”	
searchFirst	如果设置为 **True** 且 **allowAnonymous** 也设置为 **True**，则对用户执行匿名搜索；否则，则从 **uidAttribute** 构造用户的主 DN、用户登录名以及 **baseDN** 属性。	“false”（默认） “true”
uidAttribute	用于构造用户的主 DN。如果允许匿名搜索，则还可用于构造搜索筛选器。	“uid”（默认） 常见值：“uid”，“CN”

构造用户主 DN 的示例：

如果 uidAttribute 为 “UID”，用户登录名为 jsmith， baseDN 为 “OU=Users,DC=your,DC=ldap,DC=domain,DC=com”，则用户的主 DN 将为：

“UID=jsmith,OU=Users,DC=your,DC=ldap,DC=domain,DC=com”

下表列出了 LDAPLoginModule 高级模式属性：

属性	描述和示例	值
providerURL	用于身份验证的 LDAP 服务器的 URL。用于验证用户身份。 示例： “ldap://yourldap.domain.com:389” SSL 示例： “ldaps://your.ldap.domain.com:636”	默认为 searchProviderURL 属性的值
searchBaseDNs	应用搜索筛选器的分号分隔基础 DN 列表。（必需） 示例： “DN=America,DN=ns,DN=root,DN=com; DN=asia,DN=ns,DN=root,DN=com; DN=europe,DN=ns,DN=root,DN=com” 引用示例：“DN=ns,DN=root,DN=com”	

属性	描述和示例	值
searchDN	用于搜索用户主体以进行身份验证的主 DN。（必需）假设 searchFirst 设置为 true，即使您未指定它。 示例： “CN=SearchAdmin,OU=Administrators,DC=americas,DC=ns,DC=root,DC=com”	
searchFilter	RFC 2254 兼容搜索筛选器（请参阅 http://www.ief.org/rfc/rfc2254.txt）。搜索目录之前，会用用户登录名替换筛选器中的“{USERNAME}”字符串。连接到 Active Directory 时，将搜索筛选器放入 **jaas.configuration** 文件之前，使用 **ldp.exe** 来测试搜索筛选器，会十分有用。（必需） 示例 1：“(uid={USERNAME})” 示例 2： “(&(CN={USERNAME})(objectClass=user))” 示例 3：“(sAMAccountName={USERNAME})”	
searchFirst	设置为 **True**。	“true”

属性	描述和示例	值
searchPassword	**searchDN** 属性的密码。它可以是纯文本，也可以是模糊文本。（必需）有关密码模糊的信息，请参阅附录 C，“在组件之间启用 HTTPS”。 示例：“Secret123” 模糊密码示例：“OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn”	
searchProviderURL	用于搜索用户主 DN 的 LDAP 服务器的 URL。它可用于查找用户。 示例：“ldap://america.ns.root.com:389” SSL 示例：“ldaps://america.ns.root.com:636” 引用示例：“ldaps://ns-root.com:636”	默认为 **providerURL** 属性的值。
searchReferral	如果设置为 **follow**，则当 LDAP 服务器无法解决搜索或身份验证请求时，它将引用其他 LDAP 服务器的搜索请求。如果设置为 **follow**，则在 **searchBaseDNs** 中只需列出林的主 DN。	“ignore”（默认） “follow” “throw” （请参阅 http://download.oracle.com/javase/1.5.0/docs/guide/jndi/jndi-ldap-gl.html#referral）。

注意：对 **server.properties** 或 **jaas.configuration** 文件做出任何更改之后，均必须重新启动 Diagnostics Commander 服务器。

以下是一个配置示例，其中所有用户都具有以“CN”开头的相同基础 DN，因此可以确定其主数据库，而无需进行搜索。

```
Diagnostics {
  baseDN="OU=Users,DC=simple,DC=domain,DC=com"
  providerURL="ldap://simple.domain.com:389"
  uidAttribute="CN"
  ;
};
```

如果 Larry 登录，则他的主 DN 将为“CN=larry,OU=Users,DC=simple,DC=domain,DC=com”。

以下是一个配置示例，其中所有用户都具有以“CN”开头的相同基础 DN，因此可以确定其主 DN，而无需进行搜索，但是您仍然可以执行搜索。

```
Diagnostics {
  allowAnonymous="true"
  baseDN="OU=Users,DC=simple,DC=domain,DC=com"
  providerURL="ldap://simple.domain.com:389"
  searchFirst="true"
  uidAttribute="CN"
  ;
};
```

如果 Sally 登录，则她的主 DN 将为“CN=sally,OU=Users,DC=simple,DC=domain,DC=com”。

以下是一个配置示例，其中用户可能来自世界各地，但是我们只对来自三个区域的 IT 员工感兴趣。

```
Diagnostics {
  searchFirst="true"
  searchReferral="follow"
  useSSL="true"
  serverCertificate="etc/key.store"
  searchProviderURL="ldaps://america.ns.root.com:636"
  searchDN="CN=Searcher,OU=Admins,DC=america,DC=ns,DC=root,DC=com"
  searchPassword="OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn"
  searchFilter="(&(CN={USERNAME})(objectClass=IT))"
  searchBaseDNs="DC=america,DC=ns,DC=root,DC=com; \
                 DC=africa,DC=ns,DC=root,DC=com; \
                 DC=russia,DC=ns,DC=root,DC=com"
  ;
};
```

如果在非洲的 Ororro 登录，则她的主 DN 将为“CN=ororro,OU=IT,DC=africa,DC=ns,DC=root,DC=com”。

以下是一个配置示例，其中用户可能来自世界各地，且托管于不同的 LDAP 服务器。此外，用户 CN 可以本地化，但确保其 sAMAccountName 为 ISO 8859-1。

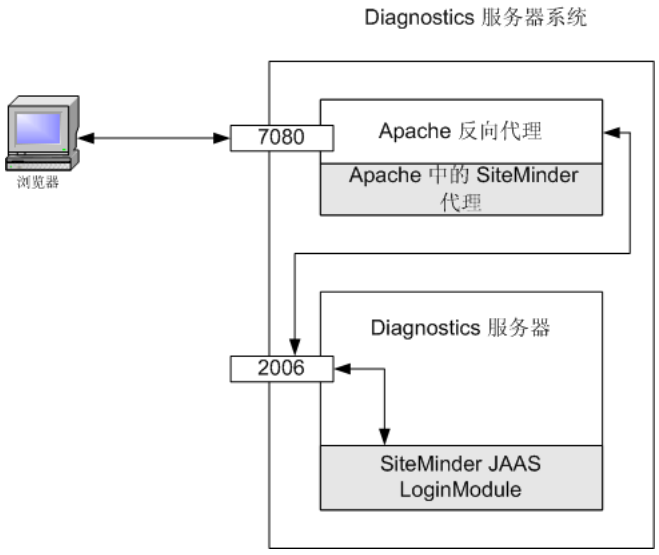
```
Diagnostics {
  searchFirst="true"
  searchReferral="follow"
  useSSL="true"
  serverCertificate="etc/key.store"
  searchProviderURL="ldaps://ns.root.com:636"
  searchDN="CN=Searcher,OU=Admins,DC=america,DC=ns,DC=root,DC=com"
  searchPassword="OBF:1fof1j1u1igh1ym51t331ym91idp1iz01fmn"
  searchFilter="(sAMAccountName={USERNAME})"
  searchBaseDNs="DC=ns,DC=root,DC=com"
  ;
};
```

如果希腊的 Saloq 以 sAMAccountName “Saloth”登录，则他用于身份验证的主 DN 将为“CN=Σαλοθ,OU=Υσερζ,DC=greece,DC=ns,DC=root,DC=com”。

使用带有 SiteMinder JAAS LoginModule 的反向代理

SiteMinder JAAS LoginModule 需要安装了 SiteMinder Web 代理的反向代理服务器。代理服务器只是将 HTTP/S 请求转发到 Diagnostics 服务器。

设置示例：



在上图中，侦听端口 7080 上请求的 Apache Web 服务器已配置为反向代理。它包含 SiteMinder Web 代理，此代理会执行身份验证；如果验证成功，则会允许 Apache 将请求传递到端口 2006 （或任何配置的 Diagnostics 服务器端口），Diagnostics 服务器在该端口上执行侦听。

注意：建议由其他 Web 服务器 （不同于反向代理的 Web 服务器）提供登录页，避免反向代理执行的重定向与 SiteMinder 模块执行的重定向之间产生冲突。或者，通过 Apache 2.2，使用 ProxyPass 指令控制特定 URL 的代理，例如“ProxyPass /loginpage !”。有关详细信息，请参阅 Apache 2.2 文档。

Diagnostics 服务器通过 SiteMinder LoginModule 检测来自 SiteMinder 的请求。

注意：要使用 SiteMinder JAAS 身份验证，用户必须转到反向代理的端口 （此示例中使用端口 7080，而不是端口 2006）。如果没有将代理服务器和 Diagnostics 服务器安装在同一系统上，则必须在 URL 中使用代理服务器的计算机名称，而不是 Diagnostics 服务器的计算机名称或 localhost。

在 HP-UX 上设置 Apache 反向代理的示例：编辑 Apache 配置文件 **httpd.conf**，并添加以下属性：

- ProxyPass /siteminderagent !
- ProxyPass / http://<Diagnostics 服务器的 IP 地址 >:2006/

  （2006 是默认的 Diagnostics 服务器端口，请使用为 Diagnostics 服务器配置的端口）
- ProxyPassReverse / http://<Diagnostics 服务器的 IP 地址 >:2006/

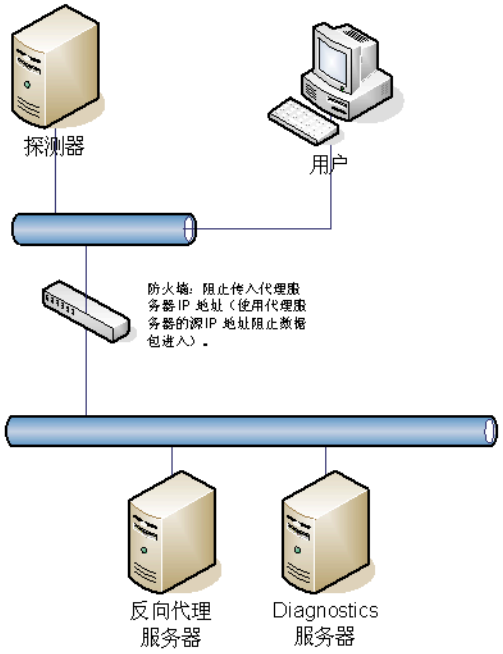
  （2006 是默认的 Diagnostics 服务器端口，请使用为 Diagnostics 服务器配置的端口）

注意：对 httpd.conf 文件做出任何更改之后，必须重新启动 Apache 服务器（先停止再启动 apachectl）。

如果担心电子欺骗，可执行以下可选操作以提供额外安全保护：

➤ 如果没有将代理服务器和 Diagnostics 服务器安装在同一系统上，则可以将 Diagnostics 服务器和代理服务器置于相同的子网上，并为交换机 / 路由器上的代理 IP 地址配置入口筛选器，以防止来自子网外部的反向代理 IP 地址的电子欺骗。

请见下图：



配置 SiteMinder JAAS 身份验证

要在 Diagnostics 中配置 SiteMinder 身份验证，必须在 Diagnostics Commander 服务器上配置以下内容：

1 将 Diagnostics 配置为使用 JAAS（请参阅 “将 Diagnostics 配置为使用 JAAS”（第 778 页））。

2 编辑 **< 安装目录 >/etc/webserver.properties** 文件。

   a 取消注释 **authentication.header.filter.username** 属性。将 **authentication.header.filter.username** 属性设置成 HTTP 请求标头中的字段，此字段应用来获取用户名。默认情况下，此属性设置为 **SM_UNIVERSALID**，它是 SiteMinder 在包含用户 ID 的 HTTP 请求中创建的字段。

   b 要使用 Diagnostics 角色，请取消注释 **authentication.header. filter.roles** 属性 （此为可选步骤）。将 **authentication.header. filter.roles** 属性设置成 HTTP 标头中的字段，此字段应用来获取角色信息。此字段可包含一个或多个角色，多个角色之间用逗号进行分隔。如果还设置了 defaultRoles，则生成的角色将是 defaultRoles 和这些角色的组合。

3 在 **< 安装目录 >/etc/jaas.configuration** 文件中编辑 Diagnostics JAAS 领域（应用程序）块，例如：

```
Diagnostics
{
    com.mercury.diagnostics.server.jaas.spi.SiteMinderLoginModule sufficient
    defaultRoles="Role1,Role2"
     ip="16.228.25.40";
};
```

SiteMinder LoginModule 选项

下面是可以在 JAAS 配置文件中为 SiteMinder LoginModule 指定的完整选项列表：

选项名称	描述	必需 / 可选	默认值	示例
IP	反向代理服务器的 IP 地址	必需		ip="16.228.25.40"
defaultRoles	要分配给每个已验证用户的角色列表，用逗号分隔。	可选		defaultRoles= "SuperUsers"

注意：对 server.properties、 webserver.properties 或 jaas.configuration 文件做出任何更改之后，必须重新启动 Diagnostics Commander 服务器。

C

在组件之间启用 HTTPS

本章提供用于在 HP Diagnostics 组件与 Business Service Management 之间启用 HTTPS 通信的配置步骤。

本章包括：

注意：这些配置说明适用于深入了解 HP Diagnostics 的经验丰富的用户。请谨慎修改 Diagnostics 组件的任何配置设置。

关于配置 HTTPS 通信

适用于每种组件类型的配置说明中包含有关以下主要步骤的详细信息：

- 在组件上生成密钥库
- 从密钥库导出证书
- 模糊用于访问密钥库的密码
- 将组件的证书复制到将启动通信的 Diagnostics 组件
- 配置组件的安全属性以启用 SSL，并提供 HTTPS 通信所需的密码

注意：查看此信息时，可参考组件通信图，具体请见附录 F，“Diagnostics 技术图”。

筛选加密密码套件

密码套件定义用于 HTTPS/SSL 加密的安全算法和密钥大小。支持的密码套件基于使用的 Java 版本。由于安全性不够或某些限制，您的组织可能会具有筛选出特定密码的策略。您可以在 **webserver.properties** 文件中设置两个与密码套件相关的属性：

- **log.cipher.suites**：将密码套件相关的消息打印到 **server.log** 文件中。这些消息包括支持的密码套件和在应用 cipher.suites.filters 之后启用的密码套件。
- **cipher.suites.filters**：以逗号分隔的正则表达式排除项和包含项的值列表，用于筛选密码套件。

例如，假设您的安装中列出了以下密码套件：

```
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
```

但是，您想要筛选出 40 位、匿名和基于 DES 的密码套件，并且只想包含基于 RC4 的加密。可以指定如下所示的筛选器：

```
cipher.suite.filters=\
  exclude:.*[0-9]?40[0-9]?.*, \
  exclude:.*_anon_.*, \
  exclude:.*_DES_.*, \
  exclude:.*_3DES_.*, \
  INCLUDE:.*_WITH_RC4_.*, \
  exclude:.*
```

每个 Diagnostics 组件的 HTTPS 清单

下表汇总了为每个 Diagnostics 组件启用 HTTPS 通信时必须执行的配置步骤：

配置步骤	Commander 服务器	Mediator 服务器	Java 探测器	采集器	.NET 探测器
生成密钥并且导出证书	**是**	**是**	**是**	**是**	否
模糊密码	**是**	**是**	**是**	**是**	否
复制 Commander 服务器证书	否	**是**	否	否	否
复制 Mediator 服务器证书	**是**	否	**是**	**是**	**是**

配置步骤	Commander 服务器	Mediator 服务器	Java 探测器	采集器	.NET 探测器
复制 Java 探测器证书	**是**	**是**	否	否	否
复制采集器证书	否	**是**	否	否	否
编辑 security.properties：enablessl=true, keystorepassword, keypassword	**是**	**是**	**是**	**是**	否
编辑 security.properties：将 Commander 服务器证书添加到 trusted.certificates	否	**是**	否	否	否
编辑 security.properties：将 Mediator 服务器证书添加到 trusted.certificates	**是**	否	**是**	**是**	否
编辑 security.properties：将 Java 探测器证书添加到 trusted.certificates	**是**	**是**	否	否	否
编辑 security.properties：将采集器证书添加到 trusted.certificates	否	**是**	否	否	否
将 Mediator 服务器证书导入到受信任根颁发机构	否	否	否	否	**是**
编辑 server.properties：设置 commander.url	**是**	**是**	否	否	否
编辑 dispatcher.properties：设置 registrar.url	否	否	**是**	否	否
编辑 collector.properties：设置 registrar.url	否	否	否	**是**	否
编辑 probe_configuration.xml：设置 diagnosticsserver url、mediator host、metricport 和 ssl	否	否	否	否	**是**
编辑 metric.config：设置 metrics.server.uri	否	否	否	否	**是**

为 Diagnostics 组件启用传入 HTTPS 通信

本节说明如何配置 Diagnostics 服务器、Java 代理和采集器，以接收传入的 HTTPS 通信。HTTPS 通信可从其他 Diagnostics 组件传入，也可在使用 Web 浏览器访问 Diagnostics 组件时传入，或者在其他外部应用程序访问组件时传入。

本节包括以下主题：

- “为传入的 HTTPS 连接配置 Diagnostics 服务器”（第 802 页）
- “为传入的 HTTPS 连接配置 Java 代理”（第 805 页）
- “为传入的 HTTPS 连接配置采集器”（第 808 页）

生成客户端证书

使用证书服务中的高级设置生成客户端证书并指定 FQDN。将密钥标记为可导出，然后使用 Friendly Name=CLIENT。

为传入的 HTTPS 连接配置 Diagnostics 服务器

注意：

- 要避免出现 DNS 和主机名解析问题，应将 Diagnostics Commander 服务器的 Commander URL 配置为 **localhost**。通过将 **< 服务器安装目录 >/etc/server.properties** 中的 commander.url 属性设置为 http://localhost:2006（或相应端口号）可实现此目的。
- 启用与 Diagnostics Commander 服务器的 HTTPS 通信时，必须使用端口 8443 才能运行 Enterprise UI，例如：https://<commander_server>:8443。

要为传入的 HTTPS 连接配置 Diagnostics 服务器，请执行以下操作：

1 在 **<Diagnostics 服务器安装目录 >/etc** 目录中生成密钥库。命令示例显示如下：

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -genkey -keystore
<diagnostics_server_install_dir>/etc/keystore -storepass <password> -alias SERVER
-keyalg RSA -keypass <password> -dname "CN=<diagnostics_server_hostname>,
OU=Diagnostics, O=Hewlett-Packard, L=Palo Alto, S=CA, C=USA" -validity 3650
```

要使用此命令示例，请执行以下操作：

- 用 Diagnostics 服务器的安装目录路径替换 **<diagnostics_server_install_dir>**。
- 使用 Diagnostics 服务器主机的计算机名替换 **<diagnostics_server_hostname>**（应当为证书中的对象 (CN) 使用完全限定域名）。
- 使用相同的密码字符串替换每次出现的 **<password>**。可以将不同的密码分配给 **storepass** 和 **keypass**。

执行此命令之后，将在 **<diagnostics_server_install_dir>/etc/keystore** 中创建一个密钥库，其中包含一个名为 **SERVER** 的条目，用于表示 Diagnostics 服务器的主机。

2 使用以下命令导出密钥库中的 SERVER 条目的证书。

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -export -keystore
<diagnostics_server_install_dir>/etc/keystore -storepass <password> -alias
SERVER -rfc -file <diagnostics_server_install_dir>/etc/
<server_certificate_name>.cer
```

要使用此命令，请执行以下操作：

- 用 Diagnostics 服务器的安装目录路径替换 **<diagnostics_server_install_dir>**。
- 使用在创建密钥库时分配为 **storepass** 密码的字符串替换 **<password>**。
- 使用要分配给证书文件的名称替换 **<server_certificate_name>**。对于已创建证书的组件，建议您分配一个可帮助识别该组件的证书名称。

使用 **diag_server_commander.cer** 或 **diag_server_mediator.cer**。

运行此命令之后，将在 **<diagnostics_server_install_dir>/etc** 目录中为 Diagnostics 服务器创建一个证书文件（其名称在 **<server_certificate_name>** 中指定），例如 **diag_server_commander.cer**。

注意：必须为将要与 Diagnostics 服务器通信的每个 Diagnostics 组件将证书文件导入主机计算机中。以下内容说明如何将证书文件导入每个 Diagnostics 组件。

---

3 使用以下示例中的命令，生成在创建密钥库时分配的 **storepass** 和 **keypass** 密码的模糊版本。

a 用 Diagnostics 服务器的安装目录路径替换 **<diagnostics_server_install_dir>**。

b 使用创建密钥库时分配为密码的字符串替换 **<password>**。

```
<diagnostics_server_install_dir>/_jvm/bin/java -cp <diagnostics_server_install_dir>/
lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

以下示例将显示模糊操作的输出。在此示例中，密码字符串是 “testpass”。输出包含三行内容。其中的一行是要模糊的原始字符串，另外的两行是对模糊后的密码的描述。在本过程的后续步骤中，将仅使用以 “OBF” 开头的行来设置属性。

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

注意：如果未对 **keypass** 和 **storepass** 使用相同的密码，则必须运行此命令两次，才能创建每个密码的模糊版本。

---

4 在 Diagnostics Commander 服务器的 **<Diagnostics 服务器安装目录 >/etc/security.properties** 文件中更改以下属性。

a 设置 **enableSSL=true**。

b 设置 **keyStorePassword=<obfuscated_password>**。

c 设置 **keyPassword=<obfuscated_password>**。

注意：为 **<obfuscated_password>** 输入的值必须包含上一步骤中命令输出的完整 “OBF” 行；例如：

keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta

为传入的 HTTPS 连接配置 Java 代理

注意：

- 可以在使用 Sun、 IBM 和 JRockit JVM 的 SUT 上为 Java 代理 启用 SSL 和 HTTPS 通信。但是，如果您使用的 JVM 版本低于 1.4，则必须将 Sun JSSE Optional Package 下载并安装到 SUT 服务器上才能启用 SSL。

  不支持其他 JSSE 实现，例如 IBM 的 JSSE 实现。

- 启用与 Java 代理系统的 HTTPS 通信时，必须使用端口 45000 才能运行配置 Profiler UI，例如：https://<my_probe_system>:45000。

注意：安装代理的位置变为 Diagnostics < 探测器安装目录 >。默认情况下，此位置在 Windows 上是 C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent，在 UNIX 上是 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent。

要为传入的 HTTPS 连接配置 Java 代理，请执行以下操作：

1 使用以下命令在 **<probe_install_dir>/etc** 目录中生成密钥库：

```
/opt/MercuryDiagnostics/JavaAgent/_jvm/bin/keytool -genkey -keystore
<probe_install_dir>/etc/keystore -storepass <password> -alias PROBE -keyalg RSA
-keypass <password> -dname "CN=<probe_hostname>, OU=Diagnostics,
O=Hewlett-Packard, L=Palo Alto, S=CA, C=USA" -validity 3650
```

要使用此命令示例，请执行以下操作：

- 使用 Java 代理的安装目录路径替换 **<probe_install_dir>**。

- 使用 Java 代理主机的计算机名替换 **<probe_hostname>**。此值不能是服务器的 IP 地址。应当为证书中的对象 (CN) 使用完全限定域名。
- 使用相同的密码字符串替换每次出现的 **<password>**。可以将不同的密码分配给 **storepass** 和 **keypass**。

运行此命令之后，将在 **<probe_install_dir>/etc/keystore** 中创建一个密钥库，其中包含一个名为 **PROBE** 的条目，用于表示 Java 代理的主机。

2 使用以下命令导出密钥库中 PROBE 条目的证书。

```
/opt/MercuryDiagnostics/JavaAgent/_jvm/bin/keytool -export -keystore
<probe_install_dir>/etc/keystore -storepass <password> -alias PROBE -rfc -file
<probe_install_dir>/etc/<probe_certificate_name>.cer
```

要使用此命令，请执行以下操作：

- 使用 Java 代理的安装目录路径替换 **<probe_install_dir>**。
- 使用在创建密钥库时分配为 **storepass** 密码的字符串替换 **<password>**。
- 使用要分配给证书文件的名称替换 **<probe_certificate_name>**。对于已创建证书的组件，建议您分配一个可帮助识别该组件的证书名称。

  包括探测器的类型及其主机名，以便于识别为其创建证书的组件；例如：**Java_probe_<probe_hostname>。**

运行此命令之后，将在 Java 代理 的 **<probe_install_dir>/etc** 目录中创建一个名为 **Java_probe_<probe_hostname>.cer** 的目录。

注意：必须为将要与 Java 代理通信的每个 Diagnostics 组件将证书文件导入主机计算机中。以下内容说明如何将证书文件导入每个 Diagnostics 组件。

3 使用以下示例中的命令，生成在创建密钥库时分配的 **storepass** 和 **keypass** 密码的模糊版本。

a 使用 Java 代理的安装目录路径替换 **<probe_install_dir>**。

b 使用创建密钥库时分配为密码的字符串替换 **<password>**。

```
/opt/MercuryDiagnostics/JavaAgent/_jvm/bin/java -cp
<probe_install_dir>/lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

以下示例将显示模糊操作的输出。在此示例中，密码字符串是“testpass”。输出包含三行内容。其中的一行是要模糊的原始字符串，另外的两行是对模糊后的密码的描述。在本过程的后续步骤中，将仅使用以“OBF”开头的行来设置属性。

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

注意：如果未对 **keypass** 和 **storepass** 使用相同的密码，则必须运行此命令两次，才能创建每个密码的模糊版本。

4 在文件 **<probe_install_dir>/etc/security.properties** 中更改以下属性。

a 设置 **enableSSL=true**。

b 设置 **keyStorePassword=<obfuscated_password>**。

c 设置 **keyPassword=<obfuscated_password>**。

注意：为 **<obfuscated_password>** 输入的值必须包含上一步骤中命令输出的完整 “OBF” 行；例如：

keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta

为传入的 HTTPS 连接配置采集器

本节说明如何配置采集器以接收传入的 HTTPS 连接。

要为传入的 HTTPS 连接配置采集器，请执行以下操作：

1 使用以下命令在 **<collector_install_dir>/etc** 目录中生成密钥库：

```
<collector_install_dir>/_jvm/bin/keytool -genkey -keystore <collector_install_dir>/etc/
keystore -storepass <password> -alias COLLECTOR -keyalg RSA -keypass
<password> -dname "CN=<collector_hostname>, OU=Diagnostics,
O=Hewlett-Packard, L=Palo Alto, S=CA, C=USA" -validity 3650
```

要使用此命令示例，请执行以下操作：

- 使用采集器的安装目录路径替换 **<collector_install_dir>**。
- 使用采集器主机的计算机名替换 **<collector_hostname>**。此值不能是服务器的 IP 地址。应当为证书中的对象 (CN) 使用完全限定域名。

➤ 使用相同的密码字符串替换每次出现的 **<password>**。可以将不同的密码分配给 **storepass** 和 **keypass**。

运行此命令之后，将在 **<collector_install_dir>/etc/keystore** 中创建一个密钥库，其中有一个名为 **COLLECTOR** 的条目，用于表示采集器的主机。

2 使用以下命令导出密钥库中 COLLECTOR 条目的证书。

```
<collector_install_dir>/_jvm/bin/keytool -export -keystore <collector_install_dir>/etc/
keystore -storepass <password> -alias COLLECTOR -rfc -file
<collector_install_dir>/etc/<collector_certificate_name>.cer
```

要使用此命令，请执行以下操作：

➤ 使用采集器的安装目录路径替换 **<collector_install_dir>**。

➤ 使用在创建密钥库时分配为 **storepass** 密码的字符串替换 **<password>**。

➤ 使用要分配给证书文件的名称替换 **<collector_certificate_name>**。对于已创建证书的组件，建议您分配一个可帮助识别该组件的证书名称。

包括采集器的类型及其主机名，以便于识别为其创建证书的组件；例如：**collector_<collector_hostname>**。

运行此命令之后，将在采集器的 **<collector_install_dir>/etc** 目录中创建一个名为 **collector_<collector_hostname>.cer** 的目录。

注意：必须为将要与采集器通信的每个 Diagnostics 组件将证书文件导入主机计算机中。以下内容说明如何将证书文件导入每个 Diagnostics 组件。

---

3 使用以下示例中的命令，生成在创建密钥库时分配的 **storepass** 和 **keypass** 密码的模糊版本。

a 使用采集器的安装目录路径替换 **<collector_install_dir>**。

b 使用创建密钥库时分配为密码的字符串替换 **<password>**。

```
<collector_install_dir>/_jvm/bin/java -cp
<collector_install_dir>/lib/ThirdPartyLibs.jar org.mortbay.util.Password <password>
```

以下示例将显示模糊操作的输出。在此示例中，密码字符串是 testpass。输出包含三行内容。其中的一行是要模糊的原始字符串，另外的两行是对模糊后的密码的描述。在本过程的后续步骤中，将仅使用以 “OBF”开头的行来设置属性。

```
testpass
OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta
MD5:179ad45c6ce2cb97cf1029e212046e81
```

注意：如果未对 **keypass** 和 **storepass** 使用相同的密码，则必须运行此命令两次，才能创建每个密码的模糊版本。

---

4 在文件 **<collector_install_dir>/etc/security.properties** 中更改以下属性。

a 设置 **enableSSL=true**。

b 设置 **keyStorePassword=<obfuscated_password>**。

c 设置 **keyPassword=<obfuscated_password>**。

注意：为 **<obfuscated_password>** 输入的值必须包含上一步骤中命令输出的完整 “OBF”行；例如：

keyStorePassword=OBF:1ytc1vu91v2p1y831y7v1v1p1vv11yta

为 Diagnostics 组件启用传出 HTTPS 通信

以下内容说明将 Diagnostics 组件配置为向其他 Diagnostics 组件发送传出 HTTPS 通信所需执行的步骤。

注意：安装代理的位置变为 Diagnostics < 探测器安装目录 >。默认情况下，此位置在 Windows 上是 C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent，在 UNIX 上是 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent。

要启用 Diagnostics Commander 服务器以通过 HTTPS 向 Diagnostics Mediator 服务器发送传出通信，请执行以下操作：

1. 将证书文件从 Diagnostics 服务器上的 **<Dagnostics 服务器安装目录 >/etc/diag_server_mediator.cer** 复制到 Diagnostics Commander 服务器上的 **<Dagnostics 服务器安装目录 >/etc/diag_server_mediator.cer**。
2. 在 Diagnostics Commander 服务器的 **<Dagnostics 服务器安装目录 >/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。

3 设置 **trusted.certificate=diag_server_mediator.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

4 对于传入的 Diagnostics 服务器通信，通过在 Diagnostics Commander 服务器上更新 **<Diagnostics 服务器安装目录 >/etc/server.properties** 文件中的以下属性，来指示 Diagnostics 服务器的 URL。

将 **commander.url** 设置为
https://<diagserver_commander_hostname>:8443

要启用 Diagnostics Mediator 服务器，以通过 HTTPS 向 Diagnostics Commander 服务器发送传出通信，请执行以下操作：

1 将证书文件从 Diagnostics Commander 服务器上的 **<Diagnostics 服务器安装目录 >/etc/diag_server_commander.cer** 复制到 Diagnostics Mediator 服务器上的 **<Diagnostics 服务器安装目录 >/etc/diag_server_commander.cer**。

2 在 Diagnostics Mediator 服务器的 **<diagnostics_server_install_dir>/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。

3 设置 **trusted.certificate=diag_server_commander.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

4 对于传入的 Diagnostics 服务器通信，通过在 Mediator 模式下更新 Diagnostics 服务器的 **<diagnostics_server_inst_dir>/etc/server.properties** 文件中的以下属性，来指示 Diagnostics 服务器的 URL。

将 **commander.url** 设置为
https://<diagserver_commander_hostname>:8443。

注意：在服务器上启用 HTTPS 时，必须在 URL 中使用端口 8443 才能运行 Diagnostics UI。

要在 Commander 或 Mediator 模式下启用 Diagnostics 服务器，以通过 HTTPS 向探测器发送传出通信，请执行以下操作：

1 将证书文件从每个探测器的 **<probe_install_dir>/etc/java_probe_<probe_host>.cer** 复制到 Diagnostics 服务器上的 **<diagnostics_server_install_dir>/etc/Java_probe_<probe_host>.cer**。

2 在 Diagnostics 服务器的 **<diagnostics_server_install_dir>/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。

设置 **trusted.certificate=Java_probe_<probe_host>.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

要在 Mediator 模式下启用 Diagnostics 服务器，以通过 HTTPS 向采集器发送传出通信，请执行以下操作：

1 将证书文件从每个探测器的 **<collector_install_dir>/etc/collector_<collector_host>.cer** 复制到 Diagnostics 服务器上的 **<diagnostics_server_install_dir>/etc/collector_<collector_host>.cer**。

2 在 Diagnostics 服务器的 **<Diagnostics 服务器安装目录 >/etc/security.properties** 文件中更改 trusted.certificate 属性的值。

设置 **trusted.certificate=collector_<collector_host>.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

要启用 Java 代理，以通过 HTTPS 向 Diagnostics Mediator 服务器发送传出通信，请执行以下操作：

1 将证书文件从 Diagnostics Mediator 服务器上的 **<diagnostics_server_install_dir>/etc/diag_server_mediator.cer** 复制到 Java 代理上的 **<probe_install_dir>/etc/diag_server_mediator.cer**。

2 在 Java 代理的 **<probe_install_dir>/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。

设置 **trusted.certificate=diag_server_mediator.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

3 对于传入的 Java 代理通信，通过更新 **<probe_inst_dir>/etc/dispatcher.properties** 文件中的以下属性来指明 Diagnostics Mediator 服务器的 URL。

将 **registrar.url** 设置为 **https://<diagserv_mediatormode_hostname>:8443/commander/registrar/**

注意：在服务器上启用 HTTPS 时，必须在 URL 中使用端口 8443 才能运行 Diagnostics UI。

要启用服务器 / 采集器的嵌入 Java 探测器，以通过 HTTPS 将传出通信发送到 Mediator 模式下的 Diagnostics 服务器，请执行以下操作：

1 将证书文件从 Mediator 模式下的 Diagnostics 服务器上的 **<Diagnostics 服务器安装目录>/etc/diag_server_mediator.cer** 复制到嵌入的 Java 探测器上的 **<服务器 / 采集器安装目录 >/probe/etc/diag_server_mediator.cer**。

2 在嵌入的 Java 探测器的 **<server/collector_install_dir>/probe/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。设置 **trusted.certificate=diag_server_mediator.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

3 对于传入的嵌入 Java 探测器通信，通过更新 **< 服务器 / 采集器安装目录 >/probe/etc/dispatcher.properties** 文件中的以下属性来指明 Mediator 模式 Diagnostics 服务器的 URL。将 **registrar.url** 设置为 **https://<diagserv_mediatormode_hostname>:8443/commander/registrar/**。

要启用采集器，以通过 HTTPS 将传出通信发送到 Diagnostics Mediator 服务器，请执行以下操作：

1 将证书文件从 Diagnostics Mediator 服务器上的 **<diagnostics_server_install_dir>/etc/diag_server_mediator.cer** 复制到采集器上的 **<collector_install_dir>/etc/diag_server_mediator.cer**。

2 在采集器的 **<collector_install_dir>/etc/security.properties** 文件中更改 **trusted.certificate** 属性的值。

设置 **trusted.certificate=diag_server_mediator.cer**。如果在此属性值中已包含其他证书文件，请将证书文件添加到列表的结尾，并用逗号将其与前面的值分开。

3 对于传入的采集器通信，通过更新 **<collector_install_dir>/etc/collector.properties** 文件中的下属性来指明 Diagnostics Mediator 服务器的 URL。

将 **registrar.url** 设置为 **https://<diagserv_mediatormode_hostname>:8443/commander/registrar/**

注意：在服务器上启用 HTTPS 时，必须在 URL 中使用端口 8443 才能运行 Diagnostics UI。

要启用 .NET 代理，以通过 HTTPS 向 Diagnostics Commander 服务器发送传出通信，请执行以下操作：

1 将 Diagnostics Mediator 服务器的证书复制到 .NET 代理的主机。已在将 Diagnostics Mediator 服务器配置为接收 HTTPS 时生成该证书。有关配置 Diagnostics Mediator 服务器以接收 HTTPS 的说明，请参阅“为 Diagnostics 组件启用传入 HTTPS 通信”（第 801 页）。如果遵循了所引用部分中的说明，则可以在 **<diagnostics_server_install_dir>/etc/diag_server_mediator.cer** 中找到证书。

2 在 Windows 任务栏中选择“开始”>“运行”。

3 键入 mmc 并单击“确定”，以运行 Microsoft 管理控制台。

4 在 Microsoft 管理控制台菜单上，选择“文件”>“添加 / 删除管理单元”，以显示“添加 / 删除管理单元”对话框。

5 在“添加 / 删除管理单元”对话框中单击“添加”。

6 从“可用的独立管理单元”列表中选择“证书”，然后单击“添加”。

7 在“证书管理单元”对话框中选择“计算机帐户”，然后单击“下一步”。

8 在“选择计算机”对话框中选择“本地计算机（运行此控制台的计算机）”，然后单击“完成”。

9 单击“添加独立管理单元”中的“关闭”。

10 在“添加 / 删除管理单元”对话框中单击“确定”。

11 在 Microsoft 管理控制台的“控制台根目录”对话框的左窗格中，展开“证书 (本地计算机)”的列表。

12 在“证书 (本地计算机)”下，展开“受信任的根证书颁发机构”。

13 在“受信任的根证书颁发机构”下右键单击“证书”，并选择“所有任务”>“导入”以启动“证书导入向导”。

14 单击“下一步”跳过“证书导入向导”的“欢迎”对话框。

15 单击“浏览”导航到 Diagnostics Mediator 服务器的公用密钥库。

   a 在“文件”类型中选择“所有文件 (*.*)”。

   b 导航到在步骤 1 中复制 Diagnostics Commander 服务器密钥库的目录，然后单击“打开”。该目录应当是：**<diagnostics_server_install_dir>/etc/diag_server_mediator.cer**

16 单击“下一步”导入文件。

17 单击“下一步”接受“受信任的根证书颁发机构”的默认证书存储位置。

18 单击“完成”，以完成“证书导入向导”。

19 在“证书导入向导”的确认对话框中单击“确定”。

20 在 “受信任的根证书颁发机构”下选择 “证书”，以查找您刚添加的证书 （应当是 Mediator 服务器的主机名）。在“发布到”列中记录此值。此值将用于修改探测器配置文件。

21 编辑 **<probe_install_dir>/etc/probe_config.xml**，并且更改 **diagnosticsserver url** 属性，以使用 HTTPS URL：**<diagnosticsserver url="https://<diagnostics_mediator_server_host>:8443/commander" />**

22 更改 Mediator 主机和端口，并添加 ssl= “True”：**<mediator host="<diagnostics_mediator_server_host>" port="2612" metricport="8443" ssl="true"/>**

23 编辑 **<probe_install_dir>/etc/metrics.config**。更改 **metrics.server.uri** 值，以指定 HTTPS URL：**metrics.server.uri = https://<diagnostics_mediator_server_host>:8443/metricdata/**

注意：对于 **probe_config.xml** 和 **metrics.config** 文件，**<diagnostics_mediator_server_host>** 值必须与证书中的名称匹配。例如，如果证书中的主机名是完全限定主机名，则配置文件中的主机名也应是完全限定主机名。

---

24 重新启动 IIS。有关重新启动 IIS 的说明，请参见“搜寻和标准插桩”（第 281 页）。

要验证是否已成功配置 .NET 探测器以与 Diagnostics Commander 服务器进行 HTTPS 通信，请执行以下操作：

1 浏览到 .NET 应用程序以激活 .NET 代理。

2 通过在 Diagnostics UI 中检查“系统运行状况”视图，来验证 .NET 代理是否可用。

为 Business Service Management 服务器启用 HTTPS 通信

以下部分说明如何将 Business Service Management 配置为与 Diagnostics 进行 HTTPS 通信。

要启用 Diagnostics Commander 服务器和 Business Service Management 之间的 HTTPS 通信，请执行以下操作：

1 将 Diagnostics 证书文件 **diag_server_commander.cer** 从 Diagnostics Commander 服务器安装目录 **<diagnostics_server_install_dir>/etc/** 复制到 Business Service Management 主机。

2 在 Business Service Management 主机上运行以下命令，将复制的证书 **diag_server_commander.cer** 导入到 Business Service Management 服务器 cacert 密钥库中：

```
<BAC_server_install_dir>/_jvm/bin/keytool -import -file
<copied _diag_certificate_directory>/diag_server_commander.cer -keystore
<BAC_server_install_dir>/jre/lib/security/cacerts -alias SERVER
```

- 使用 Business Service Management 的安装目录路径替换 **<BAC_server_install_dir>**。
- 使用复制的 Diagnostics 证书文件路径替换 **<copied_diag_certificate_directory>**。

在提示输入密钥库密码时键入 changeit。

在询问您是否应信任证书时，键入 yes 而不是默认的 no。

3 将 Business Service Management 证书文件 **<BAC_certificate_file.cer>** 复制到 Diagnostics 服务器主机。

4 在 Diagnostics 服务器主机上运行以下命令，将复制的证书导入到 Diagnostics 服务器 cacert 密钥库中。

```
<diagnostics_server_install_dir>/_jvm/bin/keytool -import -file
<copied_BAC_certificate_directory>/<BAC_certificate_file.cer> -keystore
<diagnostics_server_install_dir>/JRE/lib/security/cacerts
```

- 使用 Diagnostics Commander 服务器的安装目录路径替换 **<Diagnostics 服务器安装目录>**。
- 使用复制的 Business Service Management 证书文件路径替换 **<copied_BAC_certificate_directory>**。

在提示输入密钥库密码时，输入在创建密钥库时分配为 **storepass** 密码的字符串。

在询问您是否应信任证书时，键入 yes 而不是默认的 no。

5 使 Business Service Management 服务器指向 Diagnostics Commander 服务器上的 HTTPS 端口。

a 选择 “管理” > “Diagnostics”，在 Business Service Management 中打开 “Diagnostics 管理”。

b 单击 “注册” 选项卡。

c 找到 Diagnostics 服务器详细信息部分。

d 在相应的字段中提供以下信息：

- 输入 Diagnostics Commander 服务器的主机名，它应当是您为 Diagnostics Commander 服务器创建密钥库时在 CN 参数中指定的名称。应当为证书中的对象 (CN) 使用 “完全限定域名” 。请参阅 “为 Diagnostics 组件启用传入 HTTPS 通信”（第 801 页）。
- 对协议输入 HTTPS。
- 对 Diagnostics 服务器的 Web 端口输入 8443。

e 单击 “提交配置”。

D

使用管理员的系统视图

您可以使用 Diagnostics 系统视图监控 Diagnostics 组件的运行状况，并验证它们是否工作正常。

本章包括：

Diagnostics 管理员的系统视图

在大型 Diagnostics 部署中，您可以使用系统视图，而不是系统运行状况监控器。这种特殊的系统视图允许您基于各种系统属性快速查找系统或系统组。允许您更加轻松地监控系统运行状况并标识系统达到容量限制的时间。

在很多情况下，当您需要知道 Diagnostics 部署中的组件以及组件所在主机的信息时，系统视图将是您首选且唯一的查询途径。您只需一眼即可以确定存在问题的组件。

要访问系统视图，请执行以下操作：

1. 以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。
2. 在查询页面的列表中查找 Mercury 系统用户，并选择此链接打开 Diagnostics。

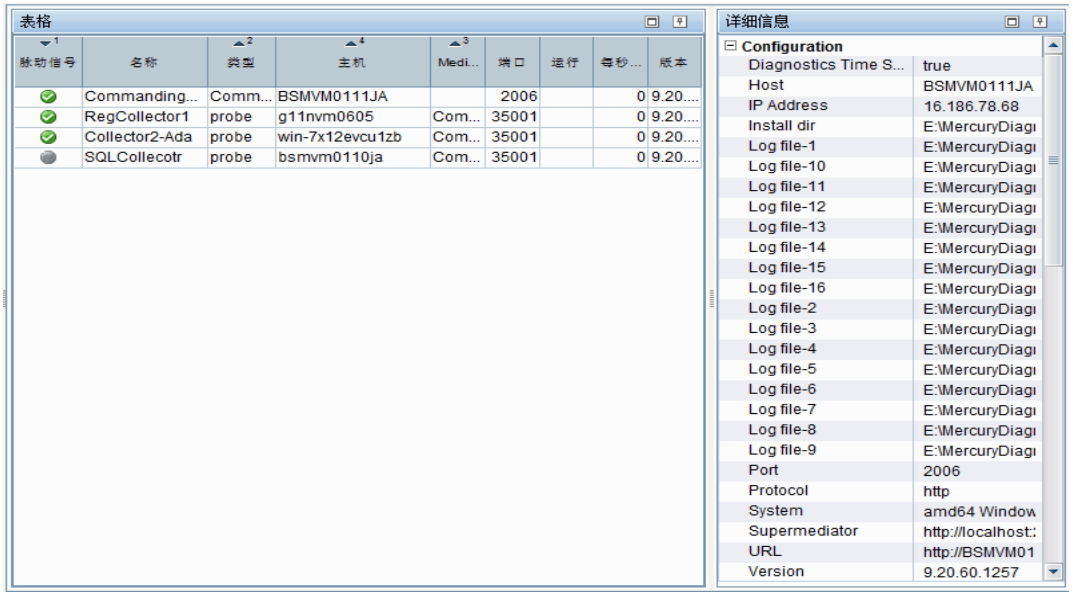
3 登录 Diagnostics 并在 “应用程序”窗口中选择 “整个企业”，然后选择任何链接打开 Diagnostics 视图。

4 在 “视图”窗格中，将显示 “系统视图”视图组。打开视图组，并选择 “系统运行状况”视图或 “系统容量”视图。

注意：系统运行状况监控器仍然可兼容使用，可从 Performance Center、LoadRunner Controller 或 Business Service Management 访问此监控器。访问系统运行状况监控器需要系统权限。在 Diagnostics 中，您可以从 http://<Diagnostics 命令服务器主机 >:<Diagnostics 命令服务器端口 >/registrar/health 访问此监控器。

系统运行状况视图描述

Diagnostics UI 中的 “系统运行状况” 视图提供 Diagnostics 环境中安装的组件的整体运行状况信息。此信息与系统运行状况监控器所提供的信息类似，但它允许您筛选、分类和选择感兴趣的度量和属性。



系统容量视图描述

Diagnostics UI 中的 “系统容量” 视图提供在 Diagnostics 环境中管理容量的信息。此视图显示了为每个 Diagnostics Mediator 分配的探测器组数和探测器数。

名称	主机	端口	探测器组计数	探测器计数
⊟ server-ovresx1-vm5	ovresx1-vm5.ovrtest.adapps.hp.com	2612	4	8
探测器组名称			探测器计数	
Sanity_LR_9_1_ovresx1-vm5				3
Sanity_CallChain_WebService_ovresx1-vm5				3
Sanity_Collectors_ovresx1-vm5				1
Sanity_RUM_Engines				1
⊞ server-HPSWROS018	HPSWROS018.ovrtest.adapps.hp.com	2612	4	14
⊞ server-ovrntt100	ovrntt100.ovrtest.adapps.hp.com	2612	3	10
⊞ server-ovrsun28.rose.hp.com	ovrsun28.rose.hp.com	2612	2	10
⊞ CommandingServer	ovrntt150.ovrtest.adapps.hp.com	2006	3	3
⊞ server-ovrlxd14.ovrtest.adapps.h...	ovrlxd14.ovrtest.adapps.hp.com	2612	1	9
⊞ server-OVRNTT154	OVRNTT154.ovrtest.adapps.hp.com	2612	2	4
⊞ server-hpsw-vm118.ovrtest.ada...	hpsw-vm118.ovrtest.adapps.hp.com	2612	1	10
⊞ server-ovresx1-vm4.ovrtest.adap...	ovresx1-vm4.ovrtest.adapps.hp.com	2612	4	12

E

Diagnostics 数据管理

本章提供有关如何管理和存储 Diagnostics 数据的详细信息。

本章包括：

关于 Diagnostics 数据

Diagnostics 数据主要有两种：

- 每个用户已创建的自定义视图。
- 由探测器收集并由 Diagnostics 服务器聚合的数据。此类数据存储在 Diagnostics 服务器上的时序数据库中。

每个 Diagnostics 服务器均存储由向其报告数据的探测器收集的数据。此外，Diagnostics 命令服务器还会存储 LoadRunner/Performance Center 运行和 Business Service Management 的虚拟事务数据，并存储应用程序度量数据。本章将讨论数据库中数据文件的组织和维护。

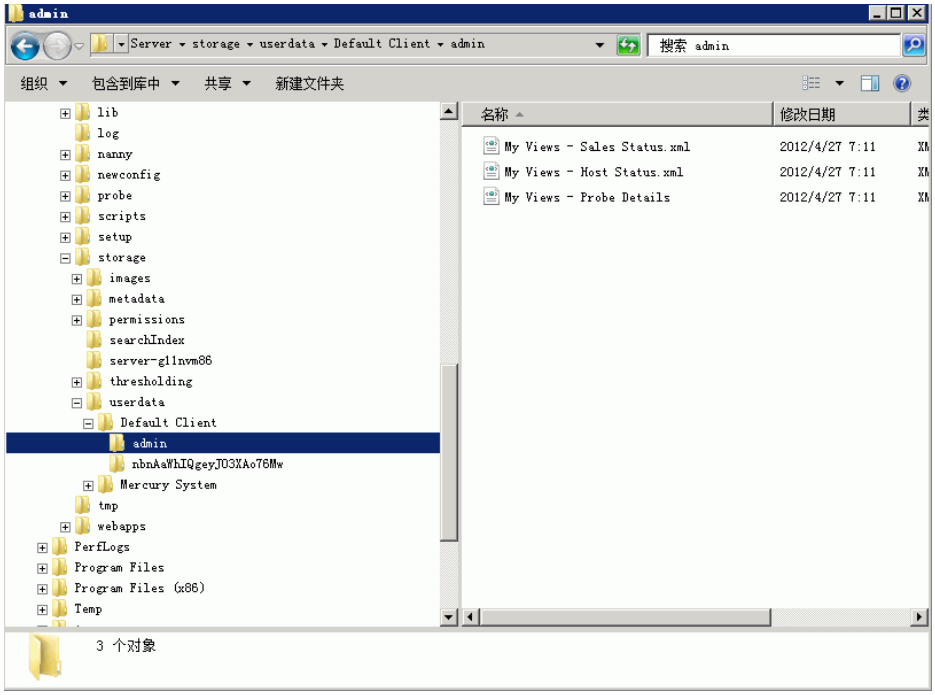
自定义视图数据

Diagnostics 用户可以创建和保存自定义视图，如《HP Diagnostics User's Guide》中的“Customizing Diagnostics Views”一章所述。 Diagnostics 会将自定义视图作为 XML 文件存储在 Diagnostics 命令服务器的主机上。

自定义视图数据构成

用户定义的自定义视图将作为 XML 文件存储在 Diagnostics 命令服务器主机上的 **<Diagnostics 服务器安装目录 >/storage/userdata** 目录中。自定义视图文件相对较小。

定义了自定义视图的每位用户在 **userdata** 目录中均有一个自己的自定义视图子目录。例如，如果管理员用户创建了两个自定义视图（“Sales Status”和“Host Status”），则这两个视图应以单独的 .xml 文件存储在 Diagnostics 命令服务器上的 **<diagnostics 服务器安装目录 >/storage/userdata/Default Client/admin** 目录中，如以下示例所示。



性能历史记录数据

Diagnostics 将历史性能数据存储在 Diagnostics Mediator 服务器上的时序数据库 (TSDB) 中。如果有大量探测器向 Diagnostics 服务器报告数据，则所存储的历史性能数据可能会增长到数 GB 大小。尽管为每个应用程序收集的数据量大小有所不同，但建议为要监控的每个虚拟机准备大约 3 GB 的数据空间。有关详细信息，请参阅 “数据保留”（第 834 页）。

本节包括：

性能历史记录数据的组织方式

Diagnostics 性能历史记录数据位于 Diagnostics Mediator 服务器的 **<Diagnostics 服务器安装目录 >/archive/mediator-< 主机名 >/persistence/< 客户名 >_** 目录中，其中：

➤ **< 主机名 >** 是 Diagnostics Mediator 服务器的主机名。

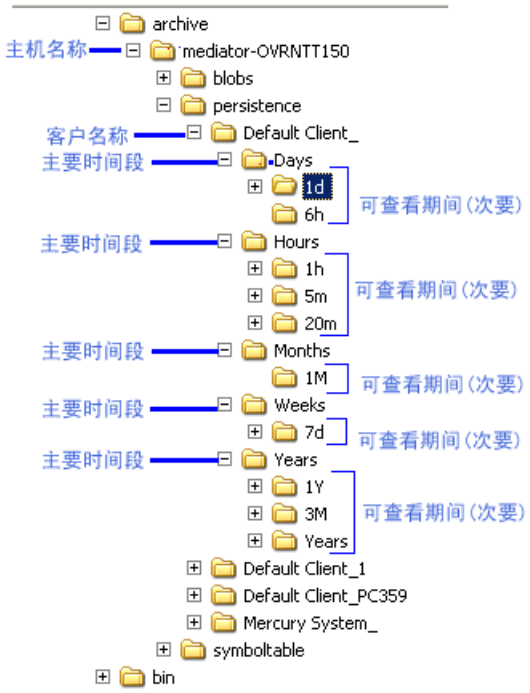
➤ **< 客户名 >** 是安装 Diagnostics Mediator 服务器时输入的客户名称。此目录的名称是客户名后加一条下划线。

etc/server.properties 中的归档 **archive.dirname** 属性可标识归档文件应存储的位置 （< 服务器安装目录 > 的绝对或相对路径）。如果要在 NAS 驱动器（如 //< 主机名 >/< 共享名 >）上移动或存储归档文件，则用以运行 Diagnostics 的用户需对 **archive.dirname** 中配置的共享具有读取 / 写入权限。

注意：

- 除非您是 HP 软件即服务 (SaaS) 客户，否则客户名应当始终是 **Default Client**。
- Performance Center 或 LoadRunner 运行的 Diagnostics 性能历史记录数据存储在 **../persistence/< 客户名 >_< 运行标识符 >** 目录中。

在 **/persistence** 目录中，性能数据将以目录形式组织，如下所示：



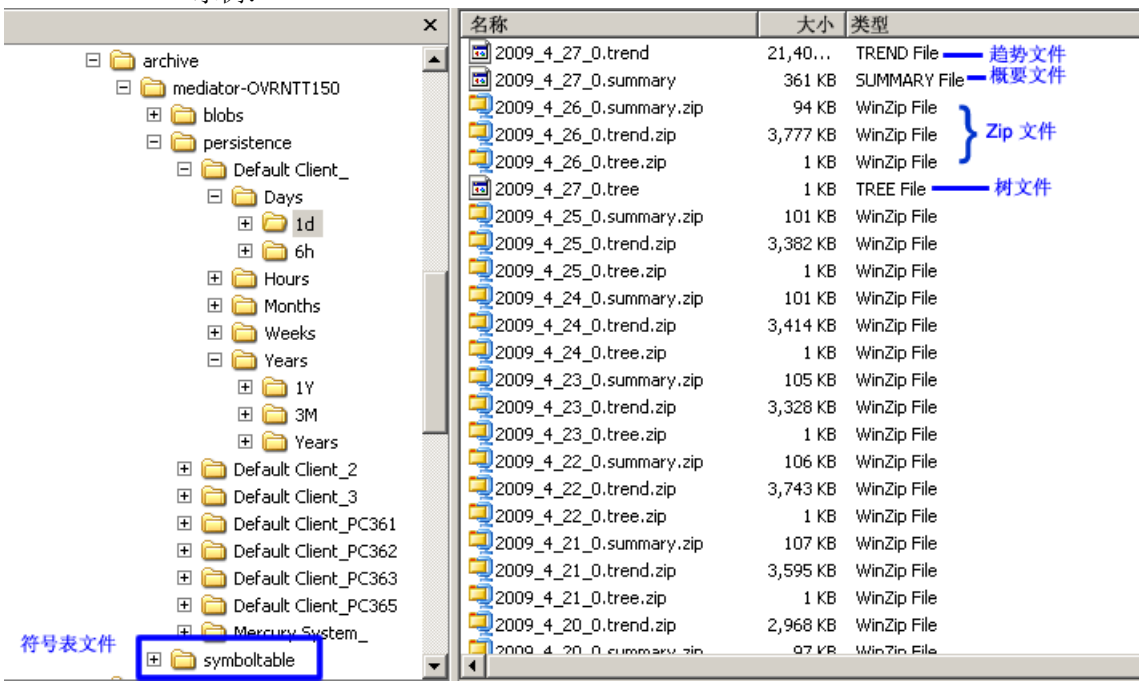
目录级别称为“主要”时间段（Days、Hours、Months、Weeks 和 Years），子目录称为“次要”时间段（1d、6h、1h、20m、5m、1M、7d、1Y、3M、Years）。这些时间段也称为数据粒度。

如以上目录示例所示， Hours 目录表示主要时间段，并有三个子目录，分别代表次要时间段 5m、 20m、 1h。

这些相同的次要时间段可充当 UI 中的查看时间段。下面是 Diagnostics 中“查看筛选器”的一个示例：

性能历史记录数据文件类型

Diagnostics 性能历史记录数据以多种文件类型存储。下面是含有这些文件的目录示例：



符号表文件

符号表包含 “字符串到整数”映射，以对其他数据文件进行小型快速数据编码。例如， /login.do 可能会编码为 1347854。

符号表存储在 **<Diagnostics 服务器安装目录 >/archive/mediator-< 主机名 >/symboltable** 目录中。

概要文件

概要文件可用于在 Diagnostics 视图的实体表和详细信息窗格中显示数据。Diagnostics 视图中显示的 “状态”基于概要文件。每个可查看的时间段都存储在单独的 “概要”文件中。

每个概要文件都按 Diagnostics 开始在其中存储概要数据时的时间 (GMT) 命名。例如，包含开始于 2009 年 4 月 27 日午夜 (GMT) 的数据概要文件将命名为 **2009_4_27_0.summary**。

趋势文件

趋势文件可用于在 Diagnostics 视图中显示图形 （图表）数据。要检索次要时间段 （如 5 分钟）的趋势，系统只会读取主要时间段 （此处为 1 小时）趋势文件中的一小部分数据。Diagnostics 会在“趋势”文件中存储每个主要时间段的图表趋势数据。

趋势文件将按照其包含的趋势数据的捕获时间 (GMT) 命名。例如，包含开始于 2009 年 4 月 27 日午夜的小时趋势数据的文件将命名为：**2009_4_27_0.trend**。

Diagnostics 在各 Diagnostics 视图中显示趋势度量时，会尝试为每个可查看期间显示大约 60 个数据点，以便呈现的趋势有意义且易于理解。为达到每个可查看期间所需的数据点， Diagnostics 服务器将在数据文件中整合适当层的数据。例如，在查看最近一小时的趋势数据时，会在图中为每一分钟显示一个数据点。这些数据点是通过对 12 个 5 秒时间段的原始数据点进行整合而创建的。

实例树文件

在向下搜索到 Diagnostics 服务器请求视图中的实例调用树时，将访问实例树文件。

“实例树”文件类似于趋势文件。将为每个主要时间段对收集的实例调用树进行相应转储，例如 **2009_4_27_0.tree**。

压缩 Zip 文件

实例树、趋势和概要文件中的数据都是当前时间段的数据。这些文件中的数据不会进行压缩。数据文件相当大，因此会在每个时间段后自动压缩这些文件以节省磁盘空间。压缩文件与未压缩文件具有相同的文件名，不同之处是扩展名为 .zip；例如 **2009_4_26_0.summary.zip**。

数据保留

Diagnostics 使用数据保留策略优化对磁盘存储的使用。数据保留的默认设置已经过设置，可直接使用。应监控系统以检查可用的磁盘空间，并相应更改数据保留设置。

确定清除数据时间时应考虑数据保留设置，因此，如果发现磁盘空间使用率意外较高时，应检查数据保留设置以确定是否需要修改此设置。请参阅“符号表清除”（第 837 页）。

本节包括：

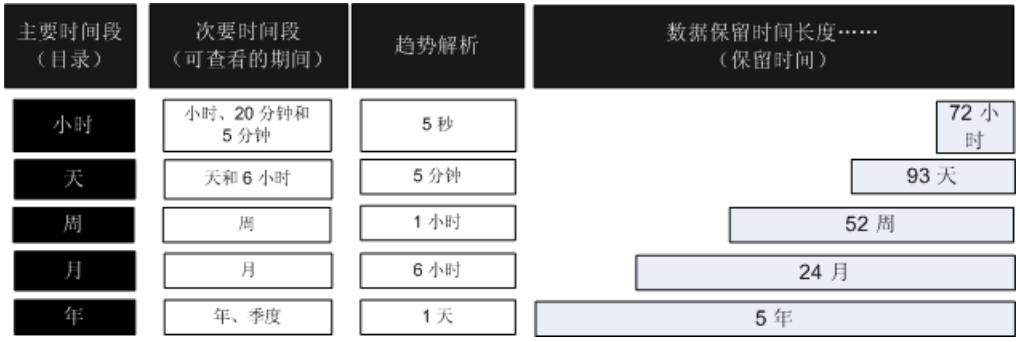
Mediator 上的数据保留

为最大限度地利用磁盘空间，历史性能数据将存储在主要和次要时间段中，并根据 Diagnostics 数据保留策略保留每个时间段中的数据。

数据点解析期较短的时间段的数据会保存较长时间，以帮助容量计划之类的活动。而数据点解析期较长的时间段的数据则不会保存很长时间，以帮助性能诊断之类的活动。对于此保留策略，测量结果显示，Diagnostics 会为每个探测的虚拟机使用大约 3 GB 空间。

下表显示了主要时间段 （“性能历史记录数据的组织方式”（第 828 页）中描述的目录）和次要时间段 （可查看期间或子目录） 。每个目录下的可查看期间均分组在一起，因为它们有相同的解析期。例如在 Days 目录中，1 天和 6 小时数据都有 5 分钟的解析期。

下表显示了常规 Diagnostics 数据保留策略。



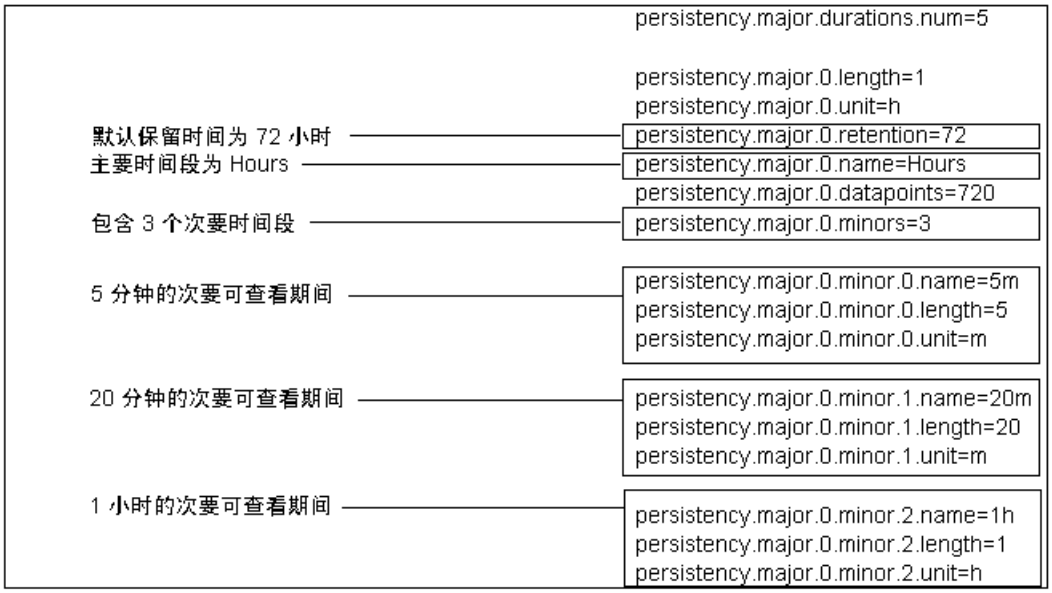
上表仅适用于实体在指定时间段内没有发生更改且一直可用的情况。请参阅 “符号表清除”（后续）。

如上表所示，最近 1 小时、20 分钟和 5 分钟查看期的数据保存 72 小时，而最近一季度的数据则保存 5 年。

数据保留配置

可以使用每个 Mediator 服务器上的 **server.properties** 文件配置数据保留行为。数据保留的默认设置已经过设置，可直接使用。应监控系统以检查可用的磁盘空间，并相应更改数据保留设置。

下面显示了 server.properties 文件的 persistence 部分的一个示例：



次要时间段会继承主要时间段的保留级别。

要将次要时间段的保留期设置为其他值，请为次要时间段 1h 添加一行内容，如下所示。此操作可以将小时数据的保留期设置为 90 小时：

```
persistency.major.0.minor.2.name=1h
persistency.major.0.minor.2.length=1
persistency.major.0.minor.2.unit=h
persistency.major.0.minor.2.retention=90
```

要将小时数据保留一周，请将保留期设置为 168 小时。单位为小时 (unit=h)，因此将一周按小时计算为 [7 （天） x 24 （小时） = 168]。

```
persistency.major.0.minor.2.name=1h
persistency.major.0.minor.2.length=1
persistency.major.0.minor.2.unit=h
persistency.major.0.minor.2.retention=168
```

要将每日数据保留 6 个月，请将保留期设置为 186 天。单位为天 (unit=d)，因此将 6 个月按天计算为 [6 （月） x 31 （天 / 月） = 186]。

```
persistency.major.1.minor.1.name=1d
persistency.major.1.minor.1.length=24
persistency.major.1.minor.1.unit=d
persistency.major.1.minor.1.retention=186
```

符号表清除

在确定清除数据的时间和方式时，清除机制必须考虑多个设置和其他因素。

默认情况下，设置为每 6 个月（4320 小时）运行一次清除。可在 **server.properties** 中更改 **persistency.major.4.total.length** 参数的小时数，来修改清除时间间隔（需要重新启动服务器）。

重要信息：增加清除时间间隔需要消耗更多的服务器内存。

不会清除属于快照的数据，因为清除这些数据将导致快照无效。

如果重命名某探测器或 6 个月（默认情况）均未从此探测器接收到数据，Diagnostics 将自动从数据库中清除此探测器及其数据。

可指定一个属性来控制 TSDB 需要使用的空间大小，通常将会删除数据以使数据量小于所指定的阈值。可在 **<Diagnostics 服务器安装目录 >/etc/server.properties** 文件中设置 **persistence.purging.threshold** 属性。但是请注意，多个其他设置的优先级高于此阈值，从而导致保留过多的数据。

如果发现磁盘空间用尽，这不意味着清除过程没有运行，可能是以下其中一种因素影响了清除机制。例如，为 Diagnostics 分配了 10GB 服务器磁盘空间，但却看到归档文件有 20 GB，系统磁盘空间快耗尽了，这种情况可能是以下任意原因造成的：

- 尚未达到清除间隔时间。可将间隔时间缩短。
- 系统上存在根据设置不会清除的大量快照。
- 数据保留设置可能要求保留过多数据。可能需要调整数据保留，以节省磁盘空间（请参阅 “数据保留”（第 834 页））。

不会清除其中包含快照数据的数据文件。由于 TSDB 是分布式的并且快照仅存在于命令服务器上，因此需要通过一个机制来确定应当保留的时间范围。

创建快照时，命令服务器会将事件的时间范围添加到保留时间的全局列表中。在删除快照后，会一并删除此时间范围。要同时允许多个快照，应为每个快照创建一个新的保留条目。由于不能执行删除处理，因此不会合并完全相同的条目。 UI 将通知服务器需要单独保留快照的时间范围。这样，可将数据保留起来以用于非快照目的。

服务器启动清除进程时，它会从命令服务器中检索保留的时间范围。如果检索列表失败，将取消清除进程，并随后重新运行该进程。如果在一周内未访问命令服务器，则将不考虑保留列表而直接运行清除进程，以避免在命令服务器永久离线时分布式服务器上的数据无限增长。

只有当归档大小超过清除阈值（**persistence.purging.threshold** 属性）后，才会开始删除数据。之后，将使用下文中定义的策略删除文件，直到归档大小小于此阈值为止（假设其他因素不会影响清除机制）。此阈值默认设置为 5G，但可以在 **server.properties** 文件中更改 **persistence.purging.threshold** 的值。

清除进程将扫描数据文件，并确定所有要删除的项。此进程将在文件集上进行操作。由于趋势文件消耗的磁盘空间最大，且需要概要文件，因此数据清除将基于趋势文件进行。对于现存的每个趋势文件 （未包含保留的时间范围），将创建一个文件集，该文件集将包含所有与此趋势相关的文件，包括含有趋势文件的主要概要的概要文件和树文件 （仅主概要包含趋势文件），以及索引到此趋势文件的所有次要概要。

每个文件集将有多个关联的值，用于确定要清除的项目。文件集的 “结束时间”是指集合内所包含的最近一个数据点的时间。“清除大小”是指概要中可删除的所有文件的大小。

服务器上的磁盘空间问题

可以设置电子邮件警报，以针对服务器上的磁盘空间问题向 Diagnostics 管理员发送警报。可在服务器安装期间输入管理员电子邮件地址，或在之后使用 “警报属性”页面进行设置。

如果服务器存档目录的可用磁盘空间少于 100 MB，则发出警报。如果服务器空间少于 50 MB，则将停止收集数据并退出。如果服务器可用磁盘空间少于 50 MB，则也将不会启动。这些预防措施可帮助确保服务器在磁盘空间用完之前停止收集数据，以维持服务器稳定性。

为 Diagnostics 管理员确定这些警报类型的阈值是在服务器 **server.properties** 文件中出厂配置的。有关详细信息，请参阅此文件中的注释，了解各种 **watchdog** 属性。

预安装数据管理注意事项

准备安装和配置大型 Diagnostics 服务器时，应当考虑以下性能调整建议：

- 为实现最佳性能，应当将 Diagnostics 服务器安装到空磁盘或近期经过碎片整理的磁盘上。此外，应将归档目录存储在相同磁盘上。或者，可以将归档目录安装在空磁盘或近期进经过碎片整理的磁盘上。

  注意：建议不要将用于时序数据库 （存储在 **<Diagnostics 服务器安装目录 >/archive** 中）的磁盘用于其他磁盘活动 （不要将 **<Diagnostics 服务器安装目录 >/archive** 目录安装到系统文件或临时文件等使用的相同磁盘上）。

- 为减少长时间使用后产生的碎片并提高系统性能，建议为归档目录使用单独的磁盘 （或分区）。

- 应在存储归档目录的磁盘上禁用密集型后台磁盘进程（例如磁盘碎片整理或病毒扫描）。
- 不应使用 NFS 或 Samba 等网络文件系统。

注意：专用于 Diagnostics 服务器归档目录的磁盘原始性能越高， Diagnostics 服务器可处理的负载就越多。请联系系统管理员，以确保为归档目录安装的磁盘是高性能磁盘或磁盘阵列。

备份 Diagnostics 数据

建议您定期备份 Diagnostics 数据，以便在磁盘或系统出现故障时进行恢复。

如果使用自己的备份方法，则必须关闭服务器（由于锁定的 PathSymbolTable.pst）。因此，建议使用 Diagnostics 提供的备份脚本进行备份。

注意：如果 Diagnostics 部署要求 Diagnostics 服务器具备高可用性，您可以为每个 Diagnostics 服务器创建备用 Diagnostics 服务器。这样便可在出现硬件故障或其他 Diagnostics 服务器主机问题时，使用备用 Diagnostics 服务器。请参阅“准备高可用性 Diagnostics 服务器”（第 445 页）

本节包括：

远程备份数据

通过以 HTTP 方式将 Diagnostics 数据文件下载到本地目录，然后使用常规备份过程对该目录进行备份，可实现远程备份。

Diagnostics 服务器还支持 HTTP 的 If-Modified-Since 和 Request-Range 标头（“re-get”），允许标准 HTTP 镜像软件下载或增量更新这些文件。如果选择使用自己的 HTTP 镜像软件，请联系 HP 支持代表，确保按正确顺序备份文件以保证数据完整性。

安装 Diagnostics 时会一起安装远程备份脚本，该脚本位于 **<diagnostics 安装目录 >/server/bin/remote-backup.sh** 中。 UNIX 脚本使用 wget 实用程序 (http://www.gnu.org/software/wget/wget.html)，以 HTTP 方式进行增量下载。在 Windows 上， Cygwin (http://www.cygwin.com/) 可用于运行此脚本。

或者在 Windows 上使用 **remote-backup.cmd**。此 .cmd 脚本需要 **<diagnostics 安装目录 >/server/bin/wget** 目录中的 **wget.exe** （.sh 脚本仅要求 wget 存在于此路径中）。

备份脚本可远程备份数据，您可按意愿从此目录执行传统备份。此脚本还可将数据备份到本地目录 （理想情况是同一个主机上的其他驱动器）。

重要信息：Diagnostics 服务器提供的备份脚本将按特定顺序备份数据。如果不按正确顺序备份文件会导致所恢复的备份不可用。因此，建议始终使用提供的脚本来创建数据备份。

下表列出了 **remote-backup.sh** 参数：

参数	描述
-h	从其进行下载的源主机 （或 IP 地址）
-o	存储备份的目录
-u	使用的 HTTP 用户名 默认：admin
-p	使用的 HTTP 密码 默认：admin
-P	使用的 HTTP 端口号 （可选） 默认：2006
-r	从其进行读取的 Mediator 模式 Diagnostics 服务器的 ID （对于 rhttp 备份）（可选）。例如，如果有“Commander”和“Mediator”两个服务器，则可以使用以下命令通过 rhttp 备份 Mediator：-h commander -r mediatorId。
-c	清除选项。指定后，将删除存在于输出目录中而不存在于服务器中的文件 （为获得较高性能，需保留下载时带有时间戳功能的其他文件）。
-v	指定该参数可输出更多详细信息。

例如，要将运行在 dragonfly 计算机上的 Diagnostics 服务器备份到 **dragonfly-backup** 目录，请执行以下命令：

```
% mkdir dragonfly-backup
% bin/remote-backup.sh -u admin -p secret -h dragonfly -o dragonfly-backup
```

数据将存储在以下目录中：

数据	备份目录
服务器配置	**etc/**
用户自定义视图	**storage/userdata**
原始性能历史记录数据	**archive/.../persistence/**
符号表	**archive/.../symboltable/**

配置符号表备份

有时，当存在大量符号文件时， symboltable 下 jdb 文件的备份文件夹会消耗大量磁盘空间。因此，可以按以下方法配置符号表备份：

- 通过将 **server.properties** 文件中的 **symboltable.backup** 属性设置为 true 或 false，可以启用或禁用备份符号表。
- 通过在 **server.properties** 文件中设置 **symboltable.backup.majors** 属性，可以配置符号表备份频率。

  使用逗号分隔列表将 symboltable.backup.majors 属性设置为所需的备份频率（Days、 Weeks、 Months）。这些频率值与 **persistency.major.<n>.name** 属性所定义的值相同 （请参阅 “数据保留配置”（第 835 页））。例如，要每周备份符号表，请使用值为 Weeks 的 **persistency.major.2.name**。

  server.properties 中定义的默认符号表备份配置如下：

```
 # Should the server backup the symboltable?
symboltable.backup = true
# Which majors should be backed up?
symboltable.backup.majors = Days,Weeks,Months
```

发生故障后恢复数据

备份目录中的文件存储在由 Diagnostics 服务器使用的结构中。

要从备份中恢复时序数据库，请执行以下操作：

1. 安装干净的 Diagnostics 服务器。安装完成之后， Diagnostics 服务器会自动启动。
2. 关闭 Diagnostics 服务器。
3. 通过验证进程列表中不存在任何 Java /Javaw 进程，来确保 Diagnostics 服务器已经关闭。在 Windows 系统上，可以使用任务管理器进行此操作；在 UNIX 系统上，可以使用 ps 进行此操作。
4. 在 Diagnostics 服务器中删除 **<Diagnostics 服务器安装目录 >/archive** 目录。
5. 复制数据库备份以替换 **<Diagnostics 服务器安装目录 >/archive**。
6. 如果备份之后 Diagnostics 服务器的主机名发生更改，则必须更新基于 Diagnostics 服务器主机名的目录名称，以反映新主机名。

   重命名 **<Diagnostics 服务器安装目录 >/archive/mediator-< 主机名 >**，使 **< 主机名 >** 反映新 Diagnostics 服务器主机名。例如，如果备份中的主机名为 oldhost，而新主机名为 newhost，则可以将 **<Diagnostics 服务器安装目录 >/archive/mediator-oldhost** 更改为 **<Diagnostics 服务器安装目录 >/archive/mediator-newhost**。

重新生成索引

首次启动恢复后的 Diagnostics 服务器时，必须重新生成索引数据 （未备份的数据） 。会在后台自动启动用于重新生成索引的过程，此过程需要几个小时才能完成。重新生成索引后，Diagnostics 服务器便可从探测器接收事件，但是，在恢复完成前，将无法在 Diagnostics 视图中显示某些历史数据。

已知限制

在 Diagnostics 中，会以本机字节顺序写入二进制数据。这意味着无法在 Little Endian 计算机上恢复和使用 Big Endian 计算机的 Diagnostics 数据。

升级 Diagnostics 时处理 Diagnostics 数据

有关在升级时处理 Diagnostics 数据的详细信息，请参阅附录 G，“升级和修补程序安装说明”。

F

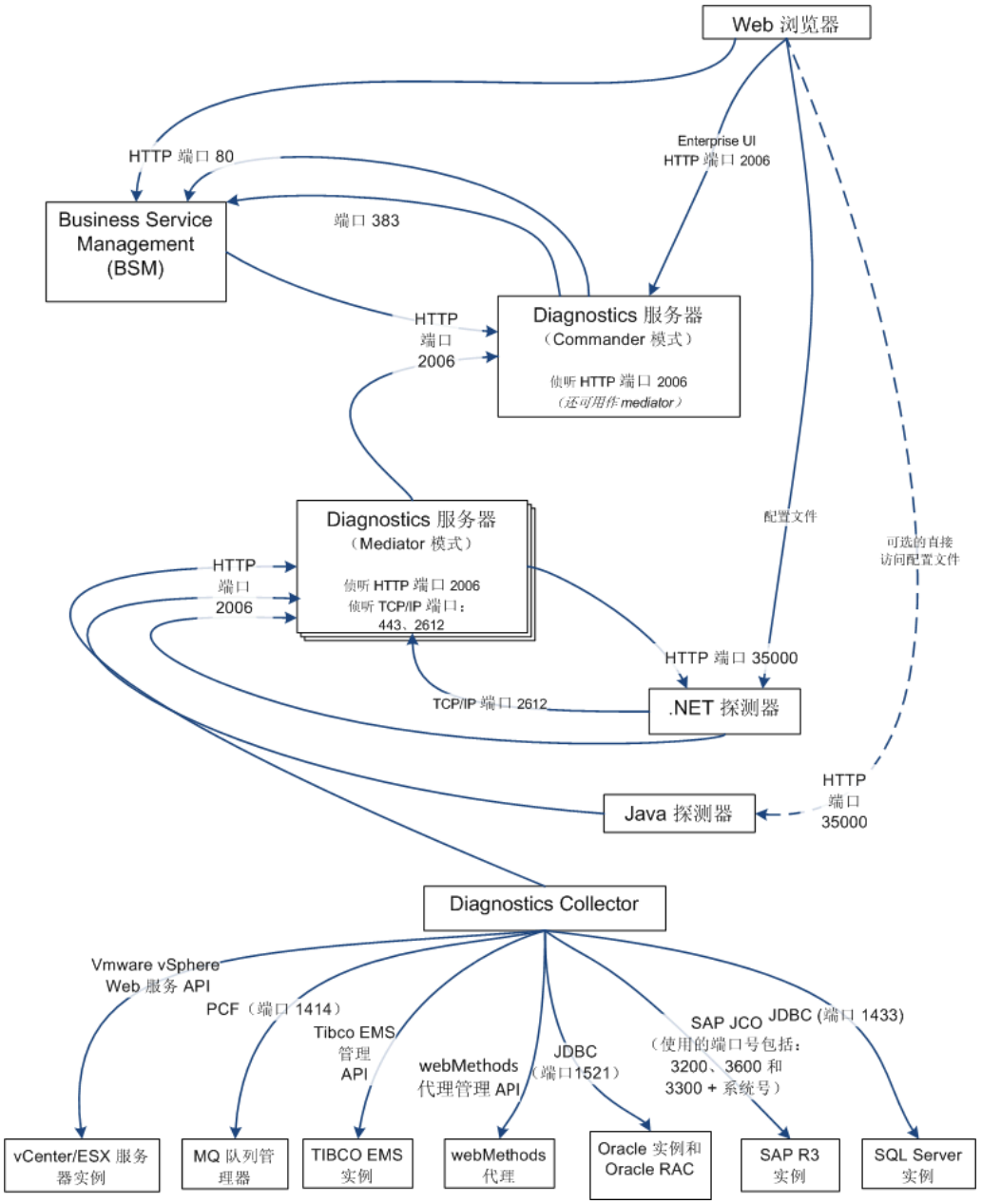
Diagnostics 技术图

本章提供数据流图和通信图，以帮助您部署 Diagnostics 组件，以及将 Diagnostics 与其他 HP 软件产品集成。

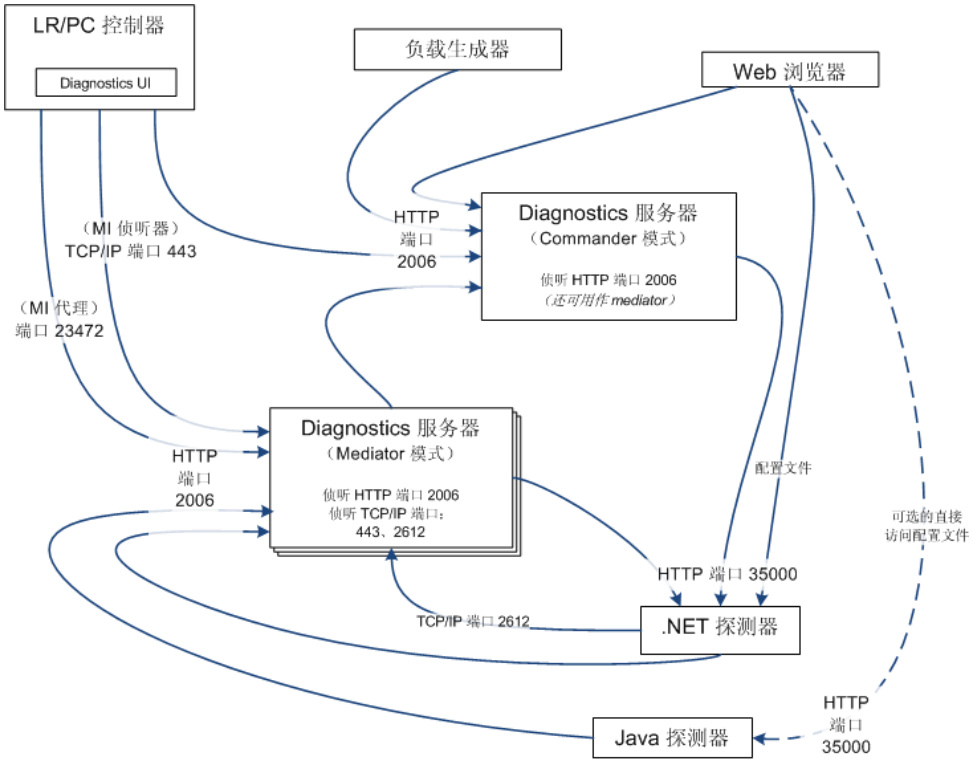
本章包括：

注意：这些图表旨在提供各种组件的高级别视图，而非其详细工作原理。

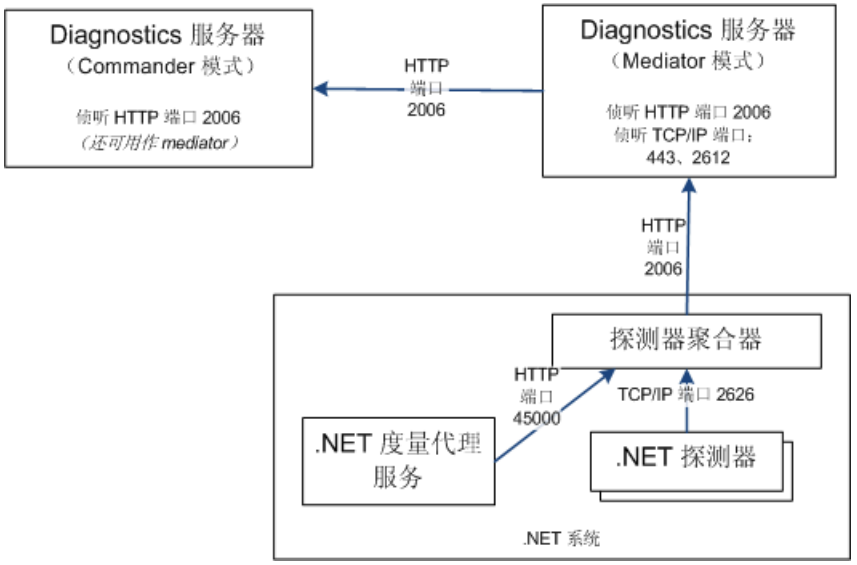
与 Business Service Management 的通信



与 LoadRunner 和 Performance Center 的通信



.NET Probe Aggregator 数据流



G

升级和修补程序安装说明

本部分提供有关 Diagnostics 主要版本之间的升级 （例如从 8.0 升级 9.0）的说明。在安装修补程序版本时，也请按照这些说明进行操作。修补程序版本包含对 Diagnostics 产品组件的完整替换项，因此需要遵循与升级过程相同的说明。

注意：Java 代理和 .NET 代理说明适用于 Diagnostics Agent 以及 TransactionVision 代理修补程序的升级或安装。

本章包括：

开始前准备

从较早版本 Diagnostics 升级或安装修补程序时，下面所述的建议在一般情况下都适用。

- 在使用较早版本组件的主机上升级到该组件的新版本之前，请确认该主机符合新版本组件的系统要求。请参阅自述文件或《HP Diagnostics 安装和配置指南》，以了解系统要求。
- 在升级代理之前，必须升级 Diagnostics 服务器。
- 如果您需要从较早版本的 Business Service Management 或 Performance Center 升级，请联系 HP 软件客户支持，并参考这些产品的升级文档，了解与 Diagnostics 集成相关的重要说明。

Diagnostics 与较早版本的兼容性

支持 Diagnostics 服务器与以下较早版本的代理和采集器配合使用：

- Java 代理 8.x、9.0x
- .NET 代理 8.x、9.0x
- 采集器 8.x、9.0x

Diagnostics 组件的升级或修补程序安装说明

以下说明用于指导您完成对现有 Diagnostics 组件的升级，或从修补程序版本中安装组件。

本节包括针对以下各组件的说明：

- “Diagnostics 服务器”（第 853 页）
- “Java 代理”（第 857 页）

➤ “.NET 代理”（第 861 页）

➤ “Diagnostics Collector”（第 862 页）

Diagnostics 服务器

本节包含有关将 Diagnostics 服务器从较早版本进行升级的说明。此说明同样适用于安装修补程序。

注意：

➤ 如果要更新 Diagnostics 服务器，必须升级部署的所有 Diagnostics 服务器。部署的所有 Diagnostics 服务器必须运行相同版本的 Diagnostics。

➤ 如果您是 HP 软件即服务 (SaaS) 客户，请联系 SaaS 支持以获取有关升级方面的指导。

在安装期间，将*覆盖* 密钥库和 JRE。因此，您的受信任证书会在升级之后失效。

对于每个新版本的 Diagnostics，应该先*重新记录* Diagnostics 服务器的静默安装响应文件，然后在多台计算机上执行静默安装。

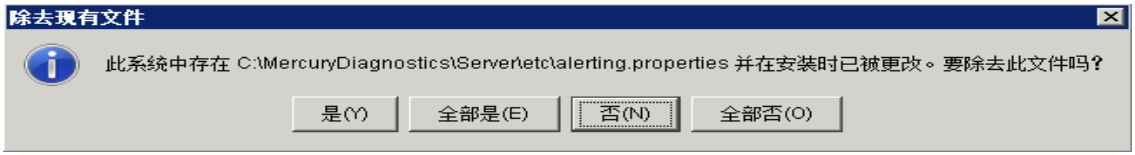
要升级 Diagnostics 服务器，请执行以下操作：

1 关闭当前的 Diagnostics 服务器。

2 为当前 Diagnostics 服务器目录生成**备份副本**。默认情况下，该目录在 Windows 上是 C:\MercuryDiagnostics\Server，
在 UNIX 上是 /opt/MercuryDiagnostics/Server，但可能会在安装服务器时指定为了其他目录。

由于升级过程会要求您卸载当前 Diagnostics 服务器，因此可以在需要重新使用 Diagnostics 服务器时使用该备份副本。

3 卸载当前 Diagnostics 服务器，**但在出现提示时保留经过修改的文件**。例如，在 Windows 中，在出现以下提示时单击 “全部否”：



有关卸载和删除 Diagnostics 服务器的信息，请完整地参阅第 2 章，“安装 Diagnostics 服务器”。

升级过程中，现有 **etc** 目录将重命名为 **etc.old_< 时间戳 >**。

4 将新 Diagnostics 服务器安装到先前版本 Diagnostics 服务器的**安装目录**中。为新 Diagnostics 服务器指定先前版本 Diagnostics 服务器所使用的**主机和端口**。

在显示以下消息时选择 “是”：

5 如果是在 Windows 中进行安装，则停止 Diagnostics 服务器。

在 Windows 中，会在安装程序完成后自动启动 Diagnostics 服务器。而在 UNIX 中则不会自动启动，因此无需停止它。

6 比较 **etc** 目录和 **etc.old_< 时间戳 >** 目录，以确定两者之间的差异。在进行此比较时，使用差异 / 合并工具可能会有所帮助。

重要信息：在从 9.20 之前的版本进行升级时，不论您之前是否已自定义以下两个属性，都可能需要对其进行修改：

- 如果 etc.old_< 时间戳 > 中包含 *server.properties* 的副本，请将 **thresholding.evaluation.status.red.for.availability** 属性及其值从旧文件复制到新文件。
- 如果 etc.old_< 时间戳 > 中包含 *thresholds.configuration* 的副本，请将 **com.mercury.diagnostics.common.data.graph.node.ProbeData.Availability** 属性及其值从旧文件复制到新文件；否则，请在新文件中将该值设置为“-95”，而非 “,-95”。

这些更改将保留之前设置的可用性阈值的行为。如果在首次升级时未执行这些更改，可以稍后再更改。

向 **etc** 目录应用由于已执行的自定义 （在 **etc.old_< 时间戳 >** 目录中）而导致的任何差异，以免丢失这些自定义内容。以下是一些常见的更改：

属性文件	要复制到新 Diagnostics 服务器的配置属性
alerting.properties	SNMP 和 SMTP 服务器，邮件地址。
security.properties	如果系统设置为 SSL 模式，则应更新所有参数，并将证书手动复制到新的 /etc 文件夹。
server.properties	超时 / 剪裁设置， Commander 的 URL。 请参阅上述有关 thresholding.evaluation.status.red.for.availability 属性的重要注意事项。
thresholds.configuration	复制所有已做出的自定义。 请参阅上述有关 com.mercury.diagnostics.common.data.graph.node.ProbeData.Availability 属性的重要注意事项。

属性文件	要复制到新 Diagnostics 服务器的配置属性
webserver.properties	默认端口信息。
.htaccess	.htaccess 文件用于安全性，需要将其复制到新的 /etc 文件夹，以保留用户角色的原始设置。

7 如果系统与 LoadRunner 或 Performance Center 集成，请将 **run_id.xml** 从 **etc.old_< 时间戳 >** 目录复制到新 **etc** 目录，以确保在将来运行时能正确增大运行 ID。

8 如果要升级 Diagnostics 命令服务器，请将 **DiagnosticsLicFile.txt** 或 **DiagnosticsServer.lic** 文件从 **etc.old< 时间戳 >** 目录复制到新的 **etc** 目录。

9 启动 Diagnostics 服务器。

10 在尝试访问 Diagnostics UI 前，清除浏览器的缓存并重新启动浏览器。

11 可通过在 Diagnostics UI 的“系统运行状况”视图中查看版本信息，来验证升级后的 Diagnostics 服务器是否正在运行。如果升级已成功，并且已重新启动 Diagnostics 服务器，则版本应该是最新版本。要访问“系统运行状况”视图，必须以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。然后，可以在“视图”窗格中选择“系统视图”视图组。

12 如果 Diagnostics 与 **Business Service Management** 集成，则需要执行以下操作：

a 查看 Diagnostics 自述文件，了解在将 Diagnostics 与 BSM 9.01/9.00 集成时需注意的所有安装事项。

b Diagnostics Commander 服务器升级完成后，必须将 **RegistrarPersistence.xml** 文件从 **etc.old<时间戳>** 文件夹复制到新的 **etc** 文件。然后检查“BSM”>“管理”>“Diagnostics”页面内的 Diagnostics 集成，如果无法正常工作，则重新在 BSM 内注册 Diagnostics 服务器。请参阅第 21 章，“设置 Business Service Management 和 Diagnostics 之间的集成”。

13 成功升级 Diagnostics 服务器之后，请删除步骤 2 中创建的备份副本。

注意：初次在较新版本中打开在较早版本 Diagnostics 创建的自定义视图时，Diagnostics 会升级视图以执行因为 Diagnostics 功能更改而导致的必要更改。当 Diagnostics 更改自定义视图时，会显示一条信息，告诉您自定义视图已被修改。

Java 代理

本节包含有关将 Diagnostics Java 代理从较早版本进行升级的说明。此说明同样适用于安装修补程序。

注意：在升级与 Diagnostics 服务器连接的代理之前，必须首先升级 Diagnostics 服务器，因为 Diagnostics 服务器不能向上兼容。

注意：对于每个新版本的 Diagnostics，应该先重新记录 Java 代理的静默安装响应文件，然后在多台计算机上执行静默安装。

要升级 Java 代理，请执行以下操作：

注意：在按照这些说明更新启动脚本以启动新代理，并重新启动应用程序之后，新安装的代理才会开始监控您的应用程序。

1 将 Diagnostics Agent for Java 安装到**不同**于当前代理安装目录的**目录中**。

在安装期间，请确保为 Diagnostics 执行下列操作。这样才能确保应用程序的持久性数据能与新代理捕获的度量相匹配。

- 将 Java 代理配置为与 Diagnostics 服务器一起使用，或配置为独立 Diagnostics Profiler。如果需要，也可以将 Java 代理配置为与 TransactionVision Server 配合使用。
- 对于代理名称，请使用先前代理所使用的探测器名称
- 对于代理组名称，请使用先前代理所使用的探测器组名称
- 对于 Mediator 服务器的名称和端口，请使用先前代理使用的信息

2 安装程序将在新安装目录中创建 **< 探测器安装目录 >\etc** 目录。

在 7.50 或更高版本中，< 探测器安装目录 > 的默认目录已更改。默认位置在 Windows 上是 C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent，在 UNIX 上是 /opt/MercuryDiagnostics/JavaAgent/DiagnosticsAgent。

3 比较新代理和先前代理的 **\etc** 目录，以确定两者之间的差异。

HP 建议您运行 Java 代理附带的**属性扫描器**实用程序，该实用程序可指示两个不同 Java 代理安装之间的差异（属性和点）。要运行 “属性扫描器”实用程序，请将当前目录更改为 **< 探测器安装目录 >/contrib/JASMUtilities/Snapins**，然后按如下所示执行 **runPropertyScanner.cmd –onsole**（对于 Unix，则为 .sh）命令：

```
runPropertyScanner –onsole –diffOnly yes –Source1 ..\..\..\etc –Source2
OtherEtc
```

示例输入：

```
C:\MercuryDiagnostics\JavaAgent8\DiagnosticsAgent\contrib\JASMUtilities\Sna
pins>runPropertyScanner -console -diffOnly yes -Source1
C:\MercuryDiagnostics\JavaAgent\DiagnosticsAgent\etc -Source2
C:\MercuryDiagnostics\JavaAgent8\DiagnosticsAgent\etc
```

示例输出：

```
****** Property dispatcher.properties:stack.trace.method.calls.max
PropertyFile=dispatcher.properties
Property=stack.trace.method.calls.max
Source1=
Source2=1000
```

向新代理的 **etc** 目录应用因对先前代理 **etc.old** 目录进行自定义而导致的任何差异，以免丢失这些自定义。应注意以下更改：

属性文件	要复制到新 Diagnostics Agent 的配置属性
auto_detect.points	将已创建的子定义点和已修改的点从旧 **etc** 目录中的 **auto_detect.points** 文件复制到新 **etc** 目录。在查找可能已修改的点时，请务必检查 RMI、 LWMD 和 args_by_class 的点。
capture.properties	深度剪裁和延迟剪裁。

属性文件	要复制到新 Diagnostics Agent 的配置属性
inst.properties	define.pre.process
dispatcher.properties	minimum.sql.latency sql.parsing.mode
dynamic.properties	cpu.timestamp.collection.method
metrics.config	确保在之前版本中注释掉的所有度量在新版本中也被注释掉，以继续使用您熟悉的度量。
security.properties	如果系统设置为 SSL 模式，则设置所有属性，并将证书从旧属性文件复制到新属性文件。

4 使用 《HP Diagnostics 安装和配置指南》的 “准备应用程序服务器以使用 Java 代理进行监控”章节所描述的 JRE 插桩方法，准备要监控的应用程序服务器。特别是，您必须更新应用程序启动脚本或 JVM 参数以指向升级的代理安装。这些参数包括 -javaagent 和 / 或 -Xbootclasspath。

5 在许可的时间内，关闭受旧代理监控的应用程序。

6 重新启动应用程序，以便新版本的代理开始监控应用程序。

7 在尝试访问 Java Diagnostics Profiler 用户界面之前，清除浏览器的缓存并重新启动浏览器。如果不执行此操作，则会显示说明大小不匹配的错误消息。

8 可通过在 Diagnostics UI 的 “系统运行状况” 视图中查看版本信息，来验证升级后的 Diagnostics Agent 是否正在运行。如果升级成功，则版本应该为最新版本。要访问 “系统运行状况” 视图，必须以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。然后，可以在 “视图” 窗格中选择 “系统视图” 视图组。

9 如果所有应用程序均已迁移到最新版本，并且一切运行正常，则可以删除旧版本的目录。请不要尝试卸载旧版本，因为这样实际上会卸载新版本。

.NET 代理

本节包含有关将 Diagnostics .NET 代理从较早版本进行升级的说明。此说明同样适用于安装修补程序。

注意：在升级与 Diagnostics 服务器连接的 .NET 代理之前，必须首先升级 Diagnostics 服务器，因为 Diagnostics 服务器不能向上兼容。

要升级 .NET 代理，请执行以下操作：

1 安装新 Diagnostics .NET 代理 （选择 “升级”）。

升级操作将在探测到的应用程序重新启动时生效。

要强行使升级生效，请执行以下操作：

a 关闭当前 .NET 探测器监控的所有应用程序。

b 重新启动 IIS。

c 重新启动受旧探测器监控的应用程序。

2 可通过在 Diagnostics UI 的“系统运行状况”视图中查看版本信息，来验证升级后的 Diagnostics Agent 是否正在运行。如果升级成功，则版本应该为最新版本。要访问 “系统运行状况”视图，必须以 Mercury 系统用户身份从 http://<Diagnostics 命令服务器名称 >:2006/query/ 打开 Diagnostics UI。然后，可以在 “视图”窗格中选择 “系统视图”视图组。

Diagnostics Collector

本节包含有关将 Diagnostics Collector 从较早版本进行升级的说明。此说明也适用于安装修补程序。

注意：在升级与 Diagnostics 服务器连接的采集器之前，必须首先升级 Diagnostics 服务器，因为 Diagnostics 服务器不能向上兼容。

重要信息：对于每个新版本的 Diagnostics，应该先重新记录 Diagnostics Collector 的静默安装响应文件，然后在多台计算机上执行静默安装。

要升级 Diagnostics Collector，请执行以下操作：

1 停止或关闭要升级的 Diagnostics Collector。

2 备份当前采集器的目录。

默认情况下，此位置在 Windows 上是 **C:\MercuryDiagnostics\Collector**，在 UNIX 上是 **/opt/MercuryDiagnostics/Collector**。

由于升级过程会要求您卸载当前的 Diagnostics Collector，因此可以在需要重新使用 Diagnostics Collector 时使用该备份副本。

3 卸载当前 Diagnostics Collector，**但在出现提示时保留经过修改的文件**。例如，在 Windows 中，在出现以下提示时单击 “全部否”：

有关完全卸载和删除采集器的信息，请参阅 “卸载 Diagnostics Collector”（第 128 页）。

升级过程中，现有 **etc** 目录将重命名为 **etc.old_< 时间戳 >**。

4 将新的采集器安装到旧版本采集器的安装目录中。

请务必**使用相同的采集器名称和 Mediator 主机**，以确保应用程序的持久数据与新采集器所捕获的度量相匹配。

可以通过查看备份的 **<采集器安装目录>\etc\collector.properties** 文件确定旧采集器的名称。

5 如果是在 Windows 上进行安装，则停止采集器。

采集器会在安装程序完成后自动启动。

而在 UNIX 中则不会自动启动采集器，因此无需停止它。

6 比较新 **etc** 目录和 **etc.old_< 时间戳 >** 目录，以确定两者之间的差异。（在进行比较时，使用差异 / 合并工具可能会有所帮助。）

对 **etc** 目录应用因对目录进行自定义（在 **etc.old_< 时间戳 >** 内找到）而导致的任何差异，以免丢失这些自定义。

属性文件	要复制到新 Diagnostics 服务器的配置属性
mq-config.xml	如果采集器要监控 MQ 环境。
oracle-config.xml	如果采集器要监控 Oracle 环境。
sqlserver-config.xml	如果采集器要监控 SQL Server 环境。 如果要从 7.x 升级，则必须从 sqlserver-config.xml 文件中删除 databaseName 属性，这是因为更高版本的 Collector 会自动搜寻 databaseName。
r3config.xml	如果采集器要监控 SAP ABAP 环境。
vmware-config.xml	如果采集器要监控 VMware 环境。
tibco-ems-config.xml	如果采集器要监控 TIBCO EMS 环境。
wm-broker-config.xml	如果采集器要监控 webMethods 代理环境。
security.properties	如果系统设置为 SSL 模式，则设置所有属性，并将证书从旧属性文件复制到新属性文件。

7 启动 Diagnostics Collector。

8 可通过查看版本信息，来验证已升级的 Collector。

9 成功升级 Diagnostic Collector 之后，请删除步骤 2 中创建的备份副本。

Diagnostics 与其他 HP 软件产品的兼容性

有关版本兼容性的最新信息，请参阅位于以下网址的 Diagnostics Support Matrix：http://support.openview.hp.com/sc/support_matrices.jsp。

H

HP Diagnostics 疑难解答

此处提供的疑难解答提示用于解决在使用 HP Diagnostics 时可能遇到的问题。

本章包括：

Solaris 计算机上的组件安装被中断

如果 Solaris 计算机上的组件安装程序在完成组件安装之前中断，将没有可用于自动卸载或重新安装组件的选项。您必须手动清理已完成的部分组件安装，然后才能再次开始安装。

要在安装被中断后手动进行清理，请执行以下操作：

1. 清理安装目录。
2. 删除 **~/vpd.properties** 和 **~/vpd.patches**。
3. 删除以下 Solaris 目录：**/var/sadm/pkg/IS*** 和 **/var/sadm/pkg/MERQ**。

Java 代理无法正常运行

如果 Java 代理无法正常运行，请检查安装过程是否在 **< 探测器安装目录 > \classes\boot\java\lang** 文件夹中创建了 **ClassLoader.class** 文件。

如果未创建此文件，则确保已插桩 JRE，因为它由 JRE 创建。请参见第 6 章，“准备应用程序服务器以使用 Java 代理进行监控”。

使用 Diagnostics Profiler for Java 启动 WAS 时出错

症状：

使用 Diagnostics Profiler for Java 启动 WAS 时，类加载程序出错。

原因：

需要从插桩中排除其他类。

解决方案：

1 打开属性文件 < **探测器安装目录** >**\etc\inst.properties**

2 通过将 “!com\.ibm\..*” 附加到现有 “classes.to.exclude” 属性值的末尾，来更新该属性，以排除相应类。

```
classes.to.exclude=!iaik\.security\..*,!c8e\..*,!org\.jboss\.net\.protocol\.file\.Handler,!
org\.jboss\.net\.protocol\.file\.FileURLConnection,!.*ByCGLIB.*,!com\.ibm\..*
```

服务器端事务丢失

症状：

在 Diagnostics 中显示每个探测器的服务器请求，但是不显示与服务器请求关联的 BPM 事务。

可在 **server.log** 文件中查找以下两种症状：

“not dropping at least one transaction that timed out” – 这表示事务在一段时间内（默认为 10 分钟）未收到任何数据，并且未收到 ELT。此警告很少出现，只会在事务超时的情况下才会出现。出现此警告后，应当在 UI 中查看事务数据。有关 ELT 的详细信息，请参见 “降低 Diagnostics 服务器内存使用量 “降低 Diagnostics 服务器内存使用量”（第 440 页）。

“Late data received for time period that was already persisted. Adjusting data by...” – 这表示服务器收到无故延迟的 ELT，但是事务尚未超时。仍然会报告数据，但是不会与 BSM 或 SaaS 同时报告。

原因：

如果没有显示上面列出的任意一条日志消息，也没有显示任何事务数据，最可能的原因是 BPM 未运行脚本。

解决方案：

1. 验证 Business Process Monitor 是否在 Business Service Management 或 HP 软件即服务 (SaaS) 中运行，以及监控器是否在运行。
2. 验证 Business Process Monitor 控制台中的配置文件的状态。

事件捕获缓冲区用尽警告

症状：

某些 Diagnostics 数据丢失，并且探测器日志文件中出现以下错误：

“The event capture buffer is full, at least one event dropped.”

原因：

该日志条目表示应用程序负载过高，或者被过度使用。

解决方案：

在某些情况下，增大 etc/capture.properties 文件中的 **event_buffer.size** 属性值可以防止事件丢失，但是通常情况下需要减少应用程序插桩。

Java 代理支持采集器

runSupportSnapshot 实用程序会创建一个 .zip 文件，其中包含有关对 Diagnostics 或 TransactionVision 部署环境中的一个或多个 Java 代理实例进行疑难解答的完整文件。

.zip 文件包含以下内容：

- **<Diagnostics 探测器安装目录>\etc** 目录下的文件
- **<Diagnostics 探测器安装目录>\log** 目录下的文件
- **<TransactionVision 代理安装目录>\config** 目录下的文件
- **<TransactionVision 代理安装目录>\logs** 目录下的文件
- 属性扫描程序报告，此报告将比较两个代理目录，并报告属性文件、点文件和 XML 文件之间的差异 （仅限 TransactionVision 代理）。
- 探测器实例信息，包括属性设置。对于在 1.5 JVM 中运行的代理，还包括环境变量、堆栈转储和类加载程序信息。

要运行 runSupportSnapshot，请执行以下操作：

1 导航到 **<Diagnostics 探测器安装目录>\contrib\JASMUtilities\Snapins**。

请注意，在 < 探测器安装目录 >\bin 目录中也提供了此实用程序

2 在 Windows 中执行 **.\runSupportSnapshot.cmd -console**，
或者在 UNIX 或 Linux 上执行 **./runSupportSnapshot.sh -console**。

3 将创建一个 .zip 文件。此 zip 文件的默认保存位置为 .../DiagnosticsAgent/ 文件夹。

基于事件的运行状况指标状态排疑解难流程

与 Business Service Management 9.0 或更高版本集成时，Diagnostics 会将运行状况指标状态事件发送到 Business Service Management 网关服务器。

重要信息：要使用事件通道集成在 BSM 网关服务器和 BSM 处理服务器之间通信，则这两个计算机之间必须存在信任关系。如果需要设置此关系，请参见 “用于 DPS 和网关的单独 BSM 服务器的配置 “用于 DPS 和网关的单独 BSM 服务器的配置”（第 716 页）。

如果发送到 Business Service Management 的运行状况指标状态事件中存在问题，可以使用以下排疑解难流程：

要解决 HI 状态事件问题，请执行以下操作：

1 验证 Diagnostics 是否将探测器度量的运行状况指标状态更改写入 **bachi_data.log**：

条目：

```
C|latency|jbossas|bsavm12.ovrtest.adapps.hp.com|17b87476ac6de3938ff9898cd19c8bd8|mercury|Default Client|1|J2EE|PROBE|2010-05-17 13:40:51|latency [BpmTxJpaImpl.getBamNodeStatusKey()  (144.5s > 122.6s)]
```

2 通过 **ovc -status** 验证 **opcle** 是否运行：

```
opcle          OVO Logfile Encapsulator      AGENT,EA    (5528)   Running
opcmsga        OVO Message Agent             AGENT,EA    (5460)   Running
```

3 在代理日志文件中查找错误（例如，无法与 BSM 服务器进行通信的错误或者证书错误）：

C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\log\System.txt

4 启用跟踪 （如有必要）：

```
ovconfchg -ns eaagt -set OPC_TRACE TRUE -set OPC_TRC_PROCS opcle -set
OPC_TRACE_AREA ALL
```

跟踪将被写入到 **C:\Documents and Settings/All Users/Application Data/HP/HP BTO Software/tmp/OpC and /var/opt/OV/log/tmp** 中。

要测试 Business Service Management 网关服务器上的事件通道，请执行以下操作：

1 通过手动提交事件检查 OPR (hpbsm_opr-backend) 是否正在运行。

a 首先获取由 Diagnostics 填充的 CI 的 CMDBID （可在 BSM Diagnostics 探测器和基础结构视图中查看）。

b 然后转到 **opr\support**，并运行以下命令（使用上一步中的 CMDBID 替换粗体的 CMDBID）：

```
sendEvent.bat -s critical -t foo -eh CPU Critical -rch
UCMDB:7b75a57ee89fe6c076ce8d258be4a971
```

2 在 **log\opr-backend\opr-backend.log** 中验证流程：

```
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'PipelineEntry': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'CIResolver': got 1 events
2010-05-11 06:17:18,981 [Thread-37] ERROR
EventChannelCiResolver.resolveHost(172) - No host CI found for event
com.hp.opr.common.model.Event@4ef05e84[865b7200-5cff-71df-00eb-0f2bf8e70000,
Back to normal: Threshold violation(s) for latency on
mercury:bsavm12.ovrtest.adapps.hp.com,<null>,OPEN,NORMAL,NONE,J2EE,<null>,
<null>,UCMDB:17b87476ac6de3938ff9898cd19c8bd8,<null>,<null>,<null>,com.hp.opr.
common.model.ResolutionHints@6cd5499[<null>,ROS84604HAE.ovrtest.adapps.hp.c
om,15.43.248.231,ad2c79b2-9af7-7543-002d-ceeb548960bc],com.hp.opr.common.mo
del.ResolutionHints@126d0c4c[Diagnostics:mercury,ROS84604HAE.ovrtest.adapps.h
p.com,15.43.248.231,ad2c79b2-9af7-7543-002d-ceeb548960bc],<null>,<null>,false,-1,
-1,[],{},Tue May 11 06:17:18 PDT 2010,Tue May 11 06:17:18 PDT 2010,Tue May 11
06:17:18 PDT
2010,0,latency:Normal,<null>,<null>,Diagnostics,latency,N:17b87476ac6de3938ff9898
cd19c8bd8:latency,^<*>:17b87476ac6de3938ff9898cd19c8bd8:latency$,N|latency|jbos
sas|bsavm12.ovrtest.adapps.hp.com|17b87476ac6de3938ff9898cd19c8bd8|mercury|D
efault Client|1|J2EE|PROBE|2010-05-11
06:16:48|latency,false,com.hp.opr.common.model.MatchInfo@35425b07,<null>,<null>]
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'CiVariableReplacer': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'DowntimeProvider': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'EtiResolverByHint': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'EtiResolverByRule': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'HIUpdater': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  MarbleHealthSubmitter.submit(129) -
submitting 1 health updates for customer 1
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'ResolutionCompleted': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'IMDBStore': got 1 events
2010-05-11 06:17:18,981 [Thread-37] INFO  Step.process(45) - Pipeline Step
'PairwiseCorrelation': got 1 events
```

3 启用更多跟踪 （如有必要）：

```
HPBSM\conf\core\Tools\log4j\opr-backend\opr.backend.properties
```

有关其他疑难解答步骤，请转至以下部分的 OM 代理疑难解答。您还可以查看“用于 DPS 和网关的单独 BSM 服务器的配置 “用于 DPS 和网关的单独 BSM 服务器的配置”（第 716 页）。

OM 代理疑难解答

将 Diagnostics 与 Business Service Management 9.0 或更高版本集成时，必须在 Diagnostics Commander 服务器上安装 OM 代理和 IAPA 组件。Diagnostics 将使用这些组件向 Business Service Management 网关服务器发送运行状况指标状态事件。

- **OM 代理安装已验证。**确保已安装 OM 代理组件，如果安装出现问题，则将在 **<Diagnostics 安装目录 >/server/log.txt** 文件中报告错误。
- **根访问权限要求。**要安装 OM 代理和 IAPA 组件，需要根访问权限。如果需要在没有根访问权限的情况下安装 Diagnostics 服务器，则可以选择不安装这两个组件，并在稍后手动安装它们。当看到对话框时：OM 代理和 IAPA 组件安装不选中此框，并在稍后安装（请参阅“手动安装 OM 代理和 IAPA 组件“手动安装 OM 代理和 IAPA 组件”（第 72 页））。
- **在网关服务器上授予证书。**安装 OM 代理和 IAPA 组件之后，必须完成在使用 Business Service Management 9.0 或更高版本注册 Diagnostics 时执行的其他配置。注册过程会要求您提供所需的证书，然后，可以执行另一个步骤以在网关服务器上授予证书。（请参见“手动授予证书”部分的“在 Business Service Management 中注册 Diagnostics 服务器“在 Business Service Management 中注册 Diagnostics 服务器”（第 696 页））。

要确保 Diagnostics Commander 服务器能够对 Business Service Management 网关服务器执行 Ping 操作，请执行以下操作：

1 在 Diagnostics Commander 服务器上执行以下命令：

```
bbcutil –ping <gateway server hostname>
```

预期的输出如下：

```
bsavm12.rose.hp.com: status=eServiceOK
coreID=6c852d02-9ae6-7543-1d6e-b6fab24428f0
bbcV=06.20.101 appN=ovbbccb appV=06.20.101 conn=2
time=218 ms
```

2 如果收到 eSSLError，请删除证书（除非其他 OM 服务器也在使用 Diagnostics Commander 服务器上的 OM 代理）：

3 然后再次运行 **< 服务器安装目录 >\server\bin\switch_ovo_agent.vbs**：

```
cscript switch_ovo_agent.vbs –server <GATEWAY SERVER> -cert_srv <DATA
PROCESSING SERVER>
```

要检查代理是否无法与服务器进行通信，请执行以下操作：

1 如果在 **C:\Documents and Settings\All Users\Application Data\HP\HP BTO Software\log\System.txt** 文件中收到错误，表示代理无法与服务器通信，请验证 **com.hp.ov.opc.msgr** 是否在 Business Service Management 网关服务器上运行：

```
bbcutil –reg
BasePath=/com.hp.ov.ctrl.ovcd/
            Protocol=HTTPS
            BindAddress=localhost
            Port=1057
            Authentication=REMOTE
      BasePath=/com.hp.ov.opc.msgr/
            Protocol=https
            BindAddress=ANY
            Port=2506
            Authentication=REMOTE
```

要解决因 OM 代理而导致注册超时的问题，请执行以下操作：

1 如果看到类似如下所示的错误：

```
Timed Out: cscript //NoLogo C:\MercuryDiagnostics\Server\bin\switch_ovo_agent.vbs
-server hpswros055.ovrtest.adapps.hp.com -cert_srv
hpswros055.ovrtest.adapps.hp.com
64-bit OS
Server is currently set to '' need to register 'hpswros055.ovrtest.adapps.hp.com'
Certificate server is currently set to '' need to register
'hpswros055.ovrtest.adapps.hp.com'
```

2 在 Business Service Management 的 Diagnostics 注册页面中再次选择 “保存注册”。

BSM 服务器和数据处理器之间的 OMi 注册疑难解答

完成以下步骤，以便在位于单独系统上的 BSM 网关服务器和数据处理服务器之间注册 OMi，从而在 BSM 中完全显示 Diagnostics 9.x 事件。

要进行疑难解答，请执行以下步骤：

1 首先检查 BSM 网关服务器上的证书。在网关服务器上的命令提示符中运行命令：**ovcert --check**。

如果出现问题，则将看到类似如下所示的错误：

```
C:\Documents and Settings\Admin>ovcert -check
OvCoreId set                    : OK
Private key installed           : FAILED
Certificate installed           : FAILED
Certificate valid               : FAILED
Trusted certificates installed  : FAILED
Trusted certificates valid      : FAILED
```

否则，您将看到类似如下所示的消息：

```
C:\Documents and Settings\Admin>ovcert -check
OvCoreId set                    : OK
Private key installed           : OK
Certificate installed           : OK
Certificate valid               : OK
Trusted certificates installed  : OK
Trusted certificates valid      : OK
Check succeeded.
```

2 检查是否使用 BSM 数据处理服务器正确注册了 OMi BSM 网关服务器。在 BSM 网关服务器上的命令提示符中运行命令：**bbcutil -ping < 数据处理服务器 >**。

如果出现问题，则将看到类似如下所示的错误：

(bbc-289) status=eServiceError coreID= bbcV= appN= appV=conn=0 time=109 ms

否则，您将看到类似如下所示的消息：

status=eServiceOK coreID=09139942-991a-7549-1eae-ee2cbe62289a

bbcV=11.00.044 appN=ovbbccb appV=11.00.044 conn=2 time=156 ms

3 检查是否使用 BSM 网关服务器正确注册了 OMi BSM 数据处理服务器。在 BSM 数据处理服务器上的命令提示符中运行命令：**bbcutil -ping < 网关服务器 >**。

如果出现问题，则将看到类似如下所示的错误：

(bbc-288) status=eServiceError coreID= bbcV= appN= appV=
conn=0 time=109 ms

否则，您将看到类似如下所示的消息：

status=eServiceOK

coreID=c3475f92-a584-7546-1cfb-b2612a96538f

bbcV=11.00.044 appN=ovbbccb appV=11.00.044

conn=2 time=94 ms

4 如果遇到以上任何问题，请使用以下步骤在单独的 BSM 网关服务器和数据处理服务器上设置证书 （如 BSM 部署指南所述）。

a 在 BSM 网关服务器上的命令提示符中运行以下命令：

ovconfchg -ns sec.cm.client -set CERTIFICATE_SERVER < 数据处理服务器 >

ovcert -certreq

INFO: Certificate request has been successfully triggered

b 在 BSM 数据处理服务器上的命令提示符中运行以下命令：

ovcm -listpending -l

RequestID: e0609222-7ea6-754d-0a03-e1a09dc5dd43
Context:
CN: c3475f92-a584-7546-1cfb-b2612a96538f
Nodename: bsavm9.ovrtest.adapps.hp.com
IPAddress: 16.93.25.199
PeerAddress: 16.93.25.199

```
平台：     Windows 5.2, CPU: x64
InstallType:  Manual
TimeReceived: 1/19/2011 1:49:18 PM Pacific Standard Time
```

c 授予请求 ID。

ovcm -grant e0609222-7ea6-754d-0a03-e1a09dc5dd43

d 在 BSM 网关服务器上的命令提示符中运行命令：

ovcert --list

您应当看到证书和受信任的证书，如下例所示：

```
+-------------------------------------------------------+
| 密钥库内容                                            |
+-------------------------------------------------------+
| 证书：                                                |
|    c3475f92-a584-7546-1cfb-b2612a96538f (*)           |
+-------------------------------------------------------+
| 信任证书：                                            |
|    CA_09139942-991a-7549-1eae-ee2cbe62289a            |
+-------------------------------------------------------+
```

e 在 BSM 网关服务器上的命令提示符中运行命令：

bbcutil -ping < 数据处理服务器 >

Ping 应立即生效。

```
status=eServiceOK coreID=09139942-991a-7549-1eae-ee2cbe62289a

bbcV=11.00.044 appN=ovbbccb appV=11.00.044 conn=2 time=156 ms
```

5 现在应正常了。

一般参考信息

本节包括一般参考主题。

本章包括：

使用 UNIX 命令

在 UNIX 平台上运行安装时，您通常可以按照屏幕上显示的说明进行操作。有时，屏幕上的说明比较含糊不清。

如果说明不清楚，请遵循以下准则进行操作：

- 要从选项列表中选择一个选项，请键入该选项所对应的数字，并按 **Enter** 键。然后键入 0，并再次按 **Enter** 键以确认选择。
- 如果要选择多个选项，请键入每个选项所对应的数字，并按 **Enter** 键。选择完所有选项之后，键入 0，并再次按 **Enter** 键以确认选择。
- 如果要清除已选择的选项，请重新键入对应的数字，或键入其他选项的数字，并按 **Enter** 键。然后键入 0，并再次按 **Enter** 键以确认选择。
- 在提示处输入信息时：

- 要接受提示处显示的默认值，请按 **Enter** 键。
- 键入信息，并按 **Enter** 键继续。

- 要继续执行安装的下一个步骤，请键入 1 选择“Next”，然后按 **Enter** 键。
- 要返回先前的提示处以执行更改，请键入 2 选择 “Previous”，然后按 **Enter** 键。
- 要取消安装，请键入 3 选择 “Cancel”，然后按 **Enter** 键。
- 要重新显示提示，请键入 4 选择 “Redisplay”，然后按 **Enter** 键。

使用正则表达式

为捕获点文件中的每个探测器指定设备定义时，可以对点中的大多数参数使用正则表达式。

正则表达式是用于指定复杂搜索短语的字符串。可以使用句点 (.)、星号 (*)、插入符 (^) 和方括号 ([ ]) 等特殊字符，来定义搜索的条件。

注意：Diagnostics 中的正则表达式必须以感叹号开头。

默认情况下， Diagnostics 将以文本形式处理正则表达式中的所有字符，但句点 (.)、连字符 (-)、星号 (*)、插入符 (^)、方括号 ([ ])、括号 (())、美元符号 ($)、竖线 (|)、加号 (+)、问号 (?) 和反斜杠 (\) 除外。如果将反斜杠 (\) 放在这些特殊字符的前面时， Diagnostics 会将特殊字符视为文本字符。

常用正则表达式运算符

本节介绍一些可用于创建正则表达式的常用运算符。

注意：有关受支持的正则表达式字符的完整列表和说明，请参阅 Microsoft VBScript 文档中的 “正则表达式”部分。

运算符	用途
(\ )	将特殊字符呈现为文本字符 创建文本字符以外的特殊字符
(.)	匹配任何单个字符
([xy])	匹配列表中的任何单个字符
([^xy])	匹配不在列表中的任何单个字符
([x-y])	匹配范围内的任何单个字符
(*)	匹配零或更多特定字符
(+)	匹配一个或更多特定字符
(?)	匹配零或一个特定字符
(( ))	正则表达式分组
(|)	匹配多个正则表达式中的一个正则表达式
(^)	匹配行的开头
($)	匹配行结尾
(\w)	匹配任何包括下划线的字母数字字符
(\W)	匹配任何非字母数字字符

使用反斜杠字符

反斜杠 (\) 有两种用途。它可以与特殊字符一起使用，用于指示将下一个字符视为文本字符。例如，\. 将被视为句点 (.) 而不是通配符（请参阅 " 匹配任何单个字符”（第 885 页））。

此外，如果将反斜杠 (\) 与某些其他情况下会被视为文本字符的字符（如字母 n、t、 w 或 d）一起使用，则该组合将表示特殊字符。例如， \n 代表换行符。

以下是一个示例：

- w 匹配字符 w
- \w 则是一个特殊字符，可以匹配包括下划线在内的任何单词字符
- \\ 匹配文本字符 \
- \(匹配文本字符 (

例如，如果要查找一个具有以下名称的文件：

filename.ext

此处的句点将会被错误地视为一种正则表达式标记。要表明该句点不是正则表达式的一部分，可按照以下方式输入该文件名：

filename\.ext

注意：如果在没有特殊含义的字符之前使用反斜杠字符，将忽略反斜杠。例如，\z 匹配 z。

匹配任何单个字符

句点 (.) 指示 Diagnostics 搜索任何单个字符 （不包括 \n），例如：

```
welcome.
```

匹配 **welcomes**、**welcomed** 或后跟空格或任何其他单个字符的 **welcome**。一组句点表示相同数量的未指定字符。

要匹配包括 \n 在内的任何单个字符，请输入：

```
(.|\n)
```

有关 **( )** 正则表达式字符的详细信息，请参阅 " 正则表达式分组”（第 887 页）。有关 **|** 正则表达式字符的详细信息，请参阅 " 匹配多个正则表达式中的一个正则表达式”（第 888 页）。

匹配列表中的任何单个字符

方括号指示 Diagnostics 搜索字符列表中的任何单个字符。例如，要搜索日期 1967、 1968 或 1969，请输入：

```
196[789]
```

匹配不在列表中的任何单个字符

当插入符 (^) 是方括号内的第一个字符时，它会指示 Diagnostics 匹配列表中的所有字符，但不包括在字符串中指定的字符，例如：

[^ab]

将匹配除 **a** 或 **b** 外的所有字符。

注意：仅当插入符是方括号内显示的第一个字符时，才有此特殊含义。

匹配范围内的任何单个字符

要匹配某个范围内的单个字符，可以使用带有连字符 (-) 的方括号 ([ ])。例如，要匹配 20 世纪 60 年代中的任一年，请输入：

196[0-9]

如果连字符是方括号内的第一个或最后一个字符，或在插入符之后，则不表示范围。

例如， [-a-z] 将匹配连字符或任何小写字母。

注意：在方括号内，“.”、“*”、“[”和“\”是文本字符。例如，**[.*]** 匹配 **.** 或 *****。如果右方括号是范围内的第一个字符，则它还是文本字符。

匹配零或更多特定字符

星号 (*) 指示 Diagnostics 匹配零或更多个前一字符，例如：

```
ca*r
```

匹配 **car**、 **caaaaaar** 和 **cr**。

匹配一个或更多特定字符

加号 (+) 指示 Diagnostics 匹配一个或更多个前导字符，例如：

```
ca+r
```

匹配 **car** 和 **caaaaaar**，但不匹配 **cr**。

匹配零或一个特定字符

问号 (?) 指示 Diagnostics 匹配零或一个前一字符，例如：

```
ca?r
```

仅匹配 **car** 和 **cr**。

正则表达式分组

括号 (()) 会指示 Diagnostics 将其中包含的字符序列视为单位，就如同数学和编程语言一样。

用交替运算符 (|) 或重复运算符 （*、 +、 ?、 {}）。

匹配多个正则表达式中的一个正则表达式

竖线 (|) 指示 Diagnostics 匹配所选表达式中的其中一个，例如：

```
foo|bar
```

使 Diagnostics 匹配 **foo** 或 **bar**。

```
fo(o|b)ar
```

使 Diagnostics 匹配 **fooar** 或 **fobar**。

匹配行的开头

插入符 (^) 指示 Diagnostics 仅匹配行开头处或者换行符之后的表达式。

以下是一个示例：

```
book
```

匹配 **book**、 **my book** 和 **book list** 行内的 **book**，而

```
^book
```

仅匹配 **lines book** 和 **book list** 中的 **book**。

匹配行结尾

美元符号 ($) 指示 Diagnostics 仅匹配位于行结尾处或者换行符之前的表达式，例如：

```
book
```

匹配 **my book** 和 **book list** 行内的 **book**，而后跟 ($) 的字符串仅匹配以该字符串结尾的行，例如：

```
book$
```

仅匹配 **my book** 行内的 **book**。

匹配任何包括下划线的字母数字字符

\w 指示 Diagnostics 匹配任何字母数字字符和下划线 （A-Z、 a-z、 0-9、 _）。

以下是一个示例：

\w* 使 Diagnostics 匹配零个或多个字母数字字符：**A-Z**、**a-z**、**0-9** 和下划线 (_)。它匹配 **Ab**、 **r9Cj** 或 **12_uYLgeu_435**。

以下是一个示例：

\w{3} 指示 Diagnostics 匹配 3 个字母数字字符：**A-Z**、**a-z**、**0-9** 和下划线 (_)。它匹配 **Ab4**、 **r9_** 或 **z_M**。

匹配任何非字母数字字符

\W 指示 Diagnostics 匹配除字母数字字符和下划线以外的任何字符。

以下是一个示例：

\W 匹配 **&**、 *****、 **^**、 **%**、 **$** 和 **#** 。

组合表达式运算符

您可以在一个表达式中组合多个正则表达式运算符，以实现所需的精确搜索条件。

例如，可以组合 ‘.’和 ‘*’字符，以查找零或多个任意字符 （\n 除外）。

例如，

start.*

匹配 **start**、 **started**、 **starting**、 **starter** 等。

您可以组合使用方括号和星号以限制对非数字字符组合的搜索，例如：

[a-zA-Z]*

要匹配 0 和 1200 之间的任一数字，必须匹配 1000 - 1200 之间的 1 位数、 2 位数、 3 位数或 4 位数。

下列正则表达式可匹配 0 和 1200 之间的任一数字。

([0-9]?[0-9]?[0-9]|1[01][0-9][0-9]|1200)

多语言用户界面支持

可以在 Web 浏览器上查看多种语言的 Diagnostics 用户界面 (UI)。当 Diagnostics 与 Business Service Management 集成或以独立模式 （非集成）运行时，可在 Windows 环境中使用此功能。

如果 Diagnostics 与 LoadRunner 或 Performance Center 集成，则 UI 的显示语言将由客户端的区域设置 （在操作系统的 “区域设置”中定义）决定。

注意：Diagnostics 不支持代理名称的本地化。

本附录说明如何查看特定语言的 Diagnostics 用户界面。您可以在 Web 浏览器中查看以下语言的 Diagnostics UI：

语言	Web 浏览器中的语言首选项
英语	英语
简体中文	中文（中国） [zh-cn]、 中文（新加坡） [zh-sg]
朝鲜语	朝鲜语 [ko]
日语	日语 [ja]

您可以使用浏览器中的语言首选项来选择 Diagnostics 的查看方式。所选语言首选项仅会影响用户的本地计算机，而不会影响 Diagnostics 服务器或任何访问相同 Diagnostics 服务器的其他用户。

要查看某特定语言的 Diagnostics UI，请执行以下操作：

1. 在本地计算机上安装相应的字体（如果尚未安装）。如果在 Web 浏览器上选择尚未安装相应字体的语言，则 Diagnostics 用户界面将使用本地计算机的默认语言。

   例如，假设本地计算机上的默认语言是英语，并且将 Web 浏览器配置为使用日语。如果没有在本地计算机上安装日语字体，则 Diagnostics 用户界面将用英语显示。

2. 如果使用 Internet Explorer，则应在本地计算机上配置 Web 浏览器，以选择要用于查看 Diagnostics 用户界面的语言。有关详细信息，请参阅：http://support.microsoft.com/kb/306872/zh-cn。

   继续步骤 4。

3. 如果使用 FireFox，请按照以下方法在本地计算机上配置 Web 浏览器：

   a. 选择“工具”>“选项”>“高级”。单击“编辑语言”。此时将打开“语言”对话框。

   b. 突出显示要用于查看 Diagnostics 的语言。

      如果对话框中没有列出您需要的语言，则展开“选择要添加的语言”列表，在其中选择语言，然后单击“添加”。

   c. 单击“上移”，将所选语言移到第一行。

   d. 单击“确定”保存设置。单击“确定”关闭“语言”对话框。

4. 关闭当前使用的浏览器，然后在新的浏览器中打开 Diagnostics。此时，将以所选语言显示 Diagnostics 用户界面。

J

数据导出

Diagnostics 收集的度量数据可以直接归档到第三方数据库，您可以保留这些数据，也可以将其转换成数据库支持的报告。

通过使用类似 XPath 的查询从 Diagnostics 时间序列数据库 (TSDB) 中提取出所需的度量，即可完成数据导出，TSDB 是存储所有持久 Diagnostics 数据的库。有关 TSDB 的信息，请参阅 “Diagnostics 数据管理”（第 825 页）。

本章包括：

任务 1：准备目标数据库

导出数据的目标数据库可以是 SQL Server 或 Oracle 数据库，供 Diagnostics 命令服务器访问。有关受支持的环境的最新信息，请参阅 Diagnostics Support Matrix，网址为：http://support.openview.hp.com/sc/support_matrices.jsp。

由 Diagnostics 服务器执行的数据导出会自动在目标数据库中创建架构和表。目标数据库至少应当有 1 GB 的可用空间；在最初的几次导出操作中，应当监控数据库的大小，查看是否需要更多空间。

要连接目标数据库，必须为具有数据库读 / 写权限和表定义权限的用户指定登录凭据。

注意：如果要升级到 Diagnostics 9.10 或更高版本，但想保留 9.10 之前的旧有数据库内容，则可以更改数据库以获得以下新功能：在 9.10 或更高版本中，最小值和最大值使用双精度值，而非整数，以便允许导出的数据显示小数点位置。在升级后更改数据库，如下所示：

Oracle：

```
ALTER TABLE RECORD MODIFY (
  REC_COUNT   NUMBER(38),
  TOTAL       FLOAT,
  MINIMUM     FLOAT,
  MAXIMUM     FLOAT)
```

SQL Server：

```
ALTER TABLE RECORD ALTER COLUMN REC_COUNT   DECIMAL(19)
ALTER TABLE RECORD ALTER COLUMN TOTAL       FLOAT
ALTER TABLE RECORD ALTER COLUMN MINIMUM     FLOAT
ALTER TABLE RECORD ALTER COLUMN MAXIMUM     FLOAT
```

任务 2：确定要导出的度量

有多重方式可以控制导出哪些度量。可以指定获取基于特定实体类型的所有度量，可以从分组中排除度量，或者指定仅包括特定度量。有关 Diagnostics 数据模式的详细信息，请参见 DVD 和帮助中提供的 “Diagnostics Data Model and Query API” 文档。

重要信息：数据导出操作仅导出度量数据，即计数、延迟和平均值，而不导出实例数据或状态数据。同样地，您不能导出调用配置文件数据。

度量按实体类型分组，此外，还可对实体类型应用其他条件。以下是使用最广泛的实体类型分组：

- /probegroup/probe

  跨所有探测器组的所有探测器的度量。
- /probegroup/probe/fragment

  跨所有探测器组和探测器的所有服务器请求的度量。
- /probegroup/index[name='rollup_fragment']/fragment

  探测器跨所有探测器组累积的服务器请求的度量。
- /probegroup/probe/index[name='services']/service

  跨所有探测器组和探测器的 Web 服务 （操作服务除外）的度量。
- /index [equals(name,'apps')]/app/app_metrics

  特定应用程序的度量。
- /probegroup/probe[equals （probeType , 'Oracle'）]

  所有 Oracle 采集器的度量。
- /probegroup/probe[equals(probeType, 'SqlServer')]

  所有 SqlServer 采集器的度量。
- /host -- 所有主机的度量 （各种系统度量）。
- /txn -- 所有 BPM 事务的度量。

下表提供度量类型示例以及它们所属的类别：

类别	度量
Classes	Classes Currently Loaded Classes Loaded Classes Unloaded
Dynamic Caching	Caching Current Cache Size Caching Max Cache Size
EJB	EJB Activates EJB Activation Time EJB Committed Transactions / sec EJB Concurrent Active Methods EJB Concurrent Live Beans EJB Create Time EJB Creates EJB Drain Size EJB Drains From Pool EJB Frees EJB Gets Found EJB Gets From Pool EJB Instantiates EJB Load Time EJB Loads EJB Passivates EJB Passivation Time EJB Passive Beans EJB Pools Size EJB Ready Beans EJB Remote Time EJB Removes EJB Response Time EJB Returns Discarded EJB Returns To Pool EJB Rolled Back Transactions / sec EJB Store Time EJB Stores

类别	度量
EJB （续）	EJB Timed Out Transactions / sec EJB Total Method Calls EJB-Cache Access / sec EJB-Cache Beans Cached EJB-Cache Get Failures / sec EJB-Pool Access / sec EJB-Pool Available Beans EJB-Pool Beans in Use EJB-Pool Current Waiters EJB-Pool Get Failures / sec EJB-Pool Get Timeouts / sec
Execute Queues	Execute Queues Idle Threads Execute Queues Pending Requests Execute Queues Requests / sec Execute Queues Total Threads
GC	GC Collections/sec GC Time Spent in Collections
Http Status	5xx-6xx
J2C Connections	J2C Connection Handles J2C Connection Released J2C Connections Allocated J2C Connections Closed J2C Connections Created
JDBC	JDBC Connections Created/sec JDBC Create Connection Delay JDBC Current Capacity JDBC Execute Statement JDBC Leaked Connections JDBC Reconnect Failures JDBC Requests Waiting for Connection JDBC Statement Cache Accesses / sec JDBC Statement Cache Hits / sec JDBC Statement Cache Size JDBC Total Connections Opened JDBC Wait Seconds High

类别	度量
Latency	latency total_cpu exception_count timeout_count throughput

按任务 4 中所述内容指定要导出的组或单个度量。

任务 3：确定频率和恢复期

每个导出操作都有指定的频率，它控制操作的发生频率，从而控制返回度量的粒度。建议频率是 1h（每小时一次），这意味着导出操作每小时运行一次。频率的其他选项为：5m 和 1d。

注意：数据导出操作可以按您要求的频率运行，但是该操作可能会影响 Diagnostics 服务器的性能。因为频率越高，服务器上的负载越大。可以批量配置 Mediator 的导出进程以减少命令服务器的负载（在服务器上的 **etc/data-export-config.xml** 文件中设置的 **servers-per-query** 属性）。

您还可以指定一个频率恢复期。只有在命令 Diagnostics 服务器关闭或不可用时，才会使用恢复期。该值告诉命令 Diagnostics 服务器：当它恢复操作后，要多久才能恢复运行数据导出操作。

频率恢复期公式为：

（当前时间）-（恢复期 * 频率）

例如，假设命令 Diagnostics 服务器停止活动的时间为 24 小时，则需每小时执行一次的数据导出操作至少已经有 23 次未执行。默认情况下，数据导出操作会从发生中断的时间开始查询 （24 小时前），而每小时的数据度量会聚合成更大的存储区，因此，返回的度量是无意义的。

但是，如果将恢复期指定为 6h，则每小时执行的任务会返回到 6 小时（而不是 24）之前，启动对 TSDB 的查询；这些度量就是有意义的。

按任务 4 中所述内容设置 <frequency> 和 <recovery-periods> 元素。

任务 4：修改数据导出配置文件

导出 Diagnostics 数据的查询是在 Diagnostic Commander 服务器上的 **<Diagnostics 服务器安装目录 >/etc/data-export-config.xml** 文件中定义的。

请遵循以下步骤，以设置此文件：

1. 如果需要，可备份 <diagnostics_server_install_dir>/etc/data-export-config.xml 文件。
2. 打开 <diagnostics_server_install_dir>/etc/data-export-config.xml 文件进行编辑。
3. 找到 <enabled> 元素，然后将它设置为 true：

```
<enabled>true</enabled>
```

此元素用于打开或关闭数据导出操作。应当在不需要执行数据导出操作时禁用该操作，以避免不必要的系统开销。默认情况下，禁用数据导出操作。

4. 找到 <customer name> 元素，然后将它设置为客户名：

除非您是 SaaS 客户，否则客户名应当始终是 Default Client。

```
<customer name='Default Client">
```

5 找到 <db-target> 元素，输入目标数据库的驱动程序名称、连接 URL、用户名和密码 （加密或纯文本）。

例如，对于具有加密密码的 SQL Server，输入内容为：

```
<db-target>
  <driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver>
  <connection-url>jdbc:sqlserver://testapps.hp.com:1433;databaseName=DIAG</
connection-url>
  <username>sa</username>
  <encrypted-password>OBF:1ym51y0s1uo71z0f1unr1y0y1ym9</encrypted-password>
  <batchsize>200</batchsize>
</db-target>
```

例如，对于具有未加密密码的 Oracle，输入内容为：

```
<db-target>
  <driver>oracle.jdbc.driver.OracleDriver</driver>
  <connection-url>jdbc:oracle:thin:@testapps.hp.com:1521:ORCL</connection-url>
  <username>diagfan</username>
  <password>tiger2</password>
  <connection-property  name="oracle.jdbc.defaultNChar" value="true"/>
</db-target>
```

对于 Oracle 数据库，在需要 UTF8/UTF15 字符支持时，必须将 **oracle.jdbc.defaultNChar** 属性设置为 true。

要加密数据库密码，请使用 Diagnostics 密码加密程序。请参阅 “密码模糊” （第 123 页）。

对于 <batchsize> 元素，请用最佳 JDBC PreparedStatement 执行中使用的单位指定批大小。默认情况下，该值设置为 100，有效荷载较大的大型实现则需要调整默认值。

6 对于要导出的每组度量，请在 <query> 中指定以下内容：

id= 标识正在定义的查询的名称，且对 data-export-config.xml 文件必须是唯一的。 ID 值中不允许使用空格。

frequency= 指定运行查询频率的字符串值，选项为：1h、 5m 和 1d。

recovery-periods= 指定在发生中断之后需要多长时间才能开始查询。

<entity-path> 任务 2 中描述的实体路径分组中的一种。

<init-query-time> 或 <init-query-periods> 先前查询开始的时间。<init-query-time> 是以标准 XSD 时间格式指定的时间值。<init-query-periods> 是一个整数，用于与频率相乘以确定查询时间。如果省略，则查询会在下一个频率边界运行。

例如，以下条目将创建一个查询，该查询每小时运行一次，并返回所有探测器组中所有探测器的所有度量。如果发生中断，则仅恢复到当前时间的 2 小时之前：

```
<query id="Probes" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/probe</entity-path>
</query>
```

以下条目创建一个查询，该查询每小时运行一次，并返回所有探测器组每小时的累积片段延迟度量：

```
<query id="Aggregate-SRs" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/index[name='rollup_fragment']/fragment</entity-path>
</query>
```

以下条目创建一个查询，该查询每小时运行一次，并返回自 4 月 22 日下午 3 点开始所有探测器组每小时的 Web 服务延迟度量：

```
<query id="Web-Services" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/probe/index[name='services']/service</entity-path>
   <init-query-time>2009-04-22T15:00:00</init-query-time>
</query>
```

7 （可选）您可以使用查询中的 servers-per-query 属性做为减少服务器上负载的方式。

servers-per-query= 指定以批量方式处理导出查询。

```
<query id="Probes" frequency="1h" recovery-periods="2" servers-per-query="10">
   <entity-path>/probegroup/probe</entity-path>
</query>
```

例如，如果有 30 个 Mediator 服务器，并且设置 servers-per-query=10，则导出操作将获取 10 个 Mediator 的结果，导出这些结果，然后处理后面的 10 个 Mediator，以此类推。

只有在使用 10 个以上的 Mediator 时，才应设置此属性。

8 另外，每个查询都可以有一个 “包含”筛选器或 “排除”筛选器，它们应用于在 <entity-path> 元素中指定的查询； “包含”筛选器元素必须指定在 “排除”筛选器元素之前。

对于任何一种筛选器，须指定以下内容：

name= 用于匹配要筛选的度量名称的正则表达式。值为 "" 表示匹配所有度量。

category= 用于匹配要筛选的类别名称匹配的正则表达式。值为 "" 表示匹配所有类别。

<order>：在存在多个 “包含”或 “排除”筛选器的情况下，该元素表示处理这些筛选器的顺序。

例如，以下条目仅返回数据库度量的度量：

```
<query id="Probes" frequency="5m" recovery-periods="2">
   <entity-path>/probegroup/probe</entity-path>
   <metric-include-filter order="1" name="" category="Database" />
   <metric-exclude-filter order="1" category="" />
</query>
```

以下示例返回除 EJB-Poll 度量之外的所有 EJB 度量。

```
<query id="EJBStats" frequency="5m" recovery-periods="2">
  <entity-path>/probegroup/probe</entity-path>
  <metric-include-filter order="1" name="" category="EJB" />
  <metric-exclude-filter order="1" name="EJB-Pool" />
</query>
```

有关正则表达式的详细信息，请参见 “使用正则表达式”（第 882 页）。

9 另外，通过指定 <purge> 元素，可以为提取的数据指定数据保留规则，以防数据库耗尽存储空间。

例如，以下条目将从目标数据库中删除存在时间超过 24 小时 (retention="1d") 的数据。清除操作每小时执行一次 (frequency="1h")，并且任何需要的清除操作均可使用 1hr 增量 (purgeInterval="1h") 来清除数据，从而减少系统的总体负载：

```
<purge id="Default.Client.Purger" frequency="1h" retention="1d" purgeInterval="1h"/>
```

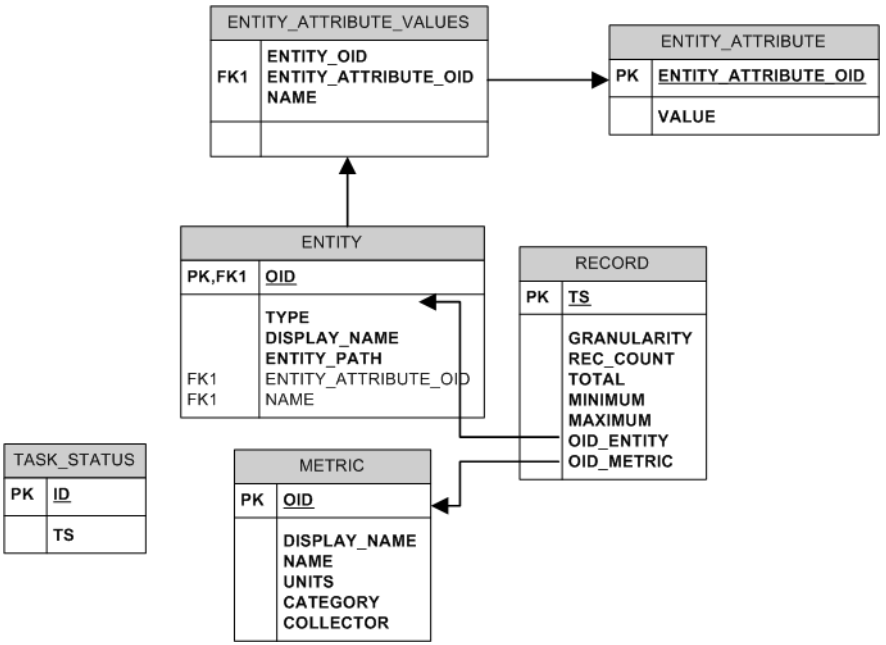
10 将更改保存到 **data-export-config.xml** 文件中。

任务 5：监控数据导出操作

假设相应的命令 Diagnostics 服务器已启动，则已保存的配置文件中的条目将立即生效，无需再重新启动命令 Diagnostics 服务器。

对 **data-export-config.xml** 文件的分析和验证方式如下：

1 通过连接并验证数据库表是否可用，来验证 XML 中指定的每个 [db-target]。如果这些数据库不可用，则是通过以下数据关系创建的：



2 Diagnostics 服务器根据指定的 <frequency>，计划每个查询运行的时间。例如，每日报告 (frequency = 1d) 要在每天的最后一个小时聚合到每日摘要后，才开始每天一次的运行。这意味着，查询会自动与现有的粒度边界保持一致。

3 执行计划的查询时，查询的结果按如下方式存储在数据库表中：

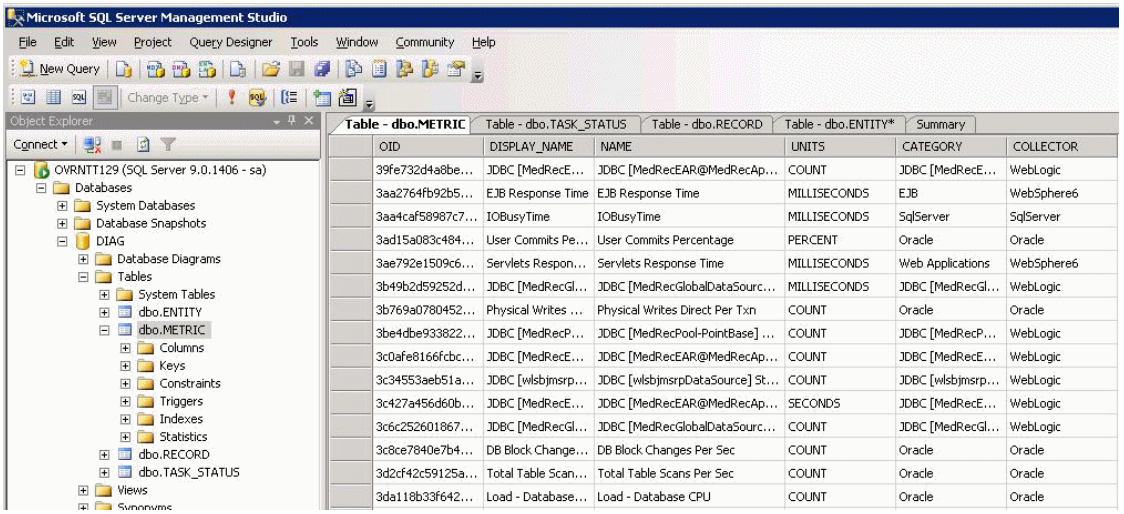
- 实体描述（例如探测器、分段或主机）都存储在 ENTITY 表中，唯一键是 TSDB 实体键和分发源值的 MD5 哈希，以保证它在联合环境中的唯一性。

- 度量描述存储在 METRIC 表中，唯一键是度量数据名称、度量数据采集器名称和分发源的 MD5 哈希，以保证它在联合环境中的唯一性。
- 度量值存储在 RECORD 表中，并且有一个指向 ENTITY 和 METRIC 表唯一键的外键值。此举可减少数据存储的总体大小，因为对实体和度量的描述会复制到每个 RECORD 表行中。
- 其他两个表 ENTITY_ATTRIBUTE_VALUES 和 ENTITY_ATTRIBUTE 将充当查找表，以进一步描述 ENTITY 维度。

任务 6：验证结果

可以使用目标数据库的数据库工具来验证预期结果。例如，下图所示为存储在 SQL Server 数据库的 METRIC 表中的结果，这组度量由下面显示的查询语句返回：

```
<query id="Probes" frequency="1h" recovery-periods="2">
   <entity-path>/probegroup/probe</entity-path>
</query>
```



任务 7：从目标数据库选择数据

当导出的数据存储在目标数据库中后，即可根据需要查询这些数据。但是，需要透视操作才能将数据转换为有用的报告格式。透视使用外键引用，将立体数据表数据合并到平面的行中，同时将描述数据加入到事实数据表中。

< 服务器安装目录 >/contrib/dataexport/ 中包含样本 SQL 脚本、查询和报告，如下所示：

- **sql_server_sample_view.sql**。SQL Server 视图，用于使导出的数据非规范化并将其转换为更友好的报告格式。
- **oracle_server_sample_view.sql**。Oracle DB 服务器视图，用于使导出的数据非规范化并将其转换为更友好的报告格式。
- **oracle_view_query_samples.sql**。各种查询 Oracle 视图的示例。
- **sql_server_reports 目录**。一个目录，包含带有各种样本报告的 SQL Server 报告项目。通过使用 SQL Server 报告工具，可打开 sql_server_reports 目录和“Diagnostic Fragments.sln”文件。

样本查询

此部分包括处理时间、查询总数和查询平均值的一些查询示例。

处理时间

要查询从上午 8 点到下午 5 点的数据，应将开始时间指定为 8:00，将结束时间指定为 16:59。如果将结束时间指定为 17:00，则会包括下一时间存储区的数据，也就是从 17:00 到 18:00 的数据，后面的示例中反映了这种情况。

查询总数

使用 sum() 函数计算度量的总数。

示例：

此查询将计算服务器请求在下午 6 点至 8 点之间的阈值冲突总数，其实体路径为：Default Client / Default / ROS54770TST_Diag80_JDK_15。

```
select entity_display_name as Server_Request, sum(total) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'threshold_violations'
  and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
  and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
  group by entity_path, entity_display_name , metric_name,units, name
  order by entity_path, entity_display_name
```

查询平均值

要计算平均度量值，请用总度量值除以计数。 rec_count 字段包含计数。

注意：“Soap 错误”度量的计数设置为其总数，因此无法计算此度量的平均值，只能计算其总数。在 Diagnostics 版本 7.5 中，“阈值冲突”度量的计数设置也同样如此。从 Diagnostics 8.0 开始，可以使用“阈值冲突”的计数，这样就能计算此度量的平均值。

示例：

此查询将计算服务器请求在下午 6 点至 8 点之间的平均延迟，其实体路径为：Default Client / Default / ROS54770TST_Diag80_JDK_15。

```
select entity_display_name as Server_Request, sum(total)/sum(rec_count) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'latency'
  and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
  and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
  group by entity_path, entity_display_name , metric_name,units, name
  order by entity_path, entity_display_name
```

示例：

此查询将计算服务器请求在下午 6 点至 8 点之间的阈值冲突平均数，其实体路径为：Default Client / Default / ROS54770TST_Diag80_JDK_15。

```
select entity_display_name as Server_Request, sum(total)/sum(rec_count) as
Avg_Latency,sum(total) as Tot_Latency, sum(rec_count) as count, metric_name,
units, name, entity_path
from DIAG.DBO.REG_FRAGMENT_TYPE_METRICS_VIEW
where metric_name = 'threshold_violations'
  and ts between '2009-06-11 18:00:00.000' and '2009-06-11 19:59:00.000'
  and entity_path = 'Default Client / Default / ROS54770TST_Diag80_JDK_15 / '
  group by entity_path, entity_display_name , metric_name,units, name
  order by entity_path, entity_display_name
```

索引

B

C

D

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z