HP Server Automation

适用于 HP-UX、 IBM AIX、 Red Hat Enterprise LinuxSolaris、 SUSE Linux Enterprise Server、 VMware 和 Windows® 操作系统

软件版本：10.0

平台开发人员指南

文档发布日期：2013 年 6 月
软件发布日期：2013 年 6 月

invent

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。必须有 HP 授予的有效许可证，方可拥有、使用或复制本软件。按照 FAR12.211 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

Intel® 和 Itanium® 是 Intel Corporation 在美国和其他国家 / 地区的商标。

Java™ 是 Sun Microsystems, Inc 在美国的商标。

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

Oracle 是 Oracle Corporation 和 / 或其附属公司的注册商标。

UNIX® 是 The Open Group 的注册商标。

文档更新

本文档的标题页包含以下标识信息：

- 软件版本号，指示软件版本。
- 文档发布日期，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。

要检查最新更新或要确定您是否在使用最新版本的文档，请访问：

http://h20230.www2.hp.com/selfsolve/manuals

此站点需要您注册 HP Passport 并登录。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

访问 HP 软件支持联机网站：

www.hp.com/go/hpsoftwaresupport

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://h20230.www2.hp.com/new_access_levels.jsp

目录

1 概述

Server Automation 平台概述

Server Automation 平台中包含一组 API 和一个运行时环境，用于帮助集成和扩展 SA。 Server Automation 平台 API 提供了各种核心服务，例如审核符合性、Windows 修补程序管理和操作系统配置。运行时环境将执行可用于访问全局文件系统 (OGFS) 的全局 Shell 脚本。

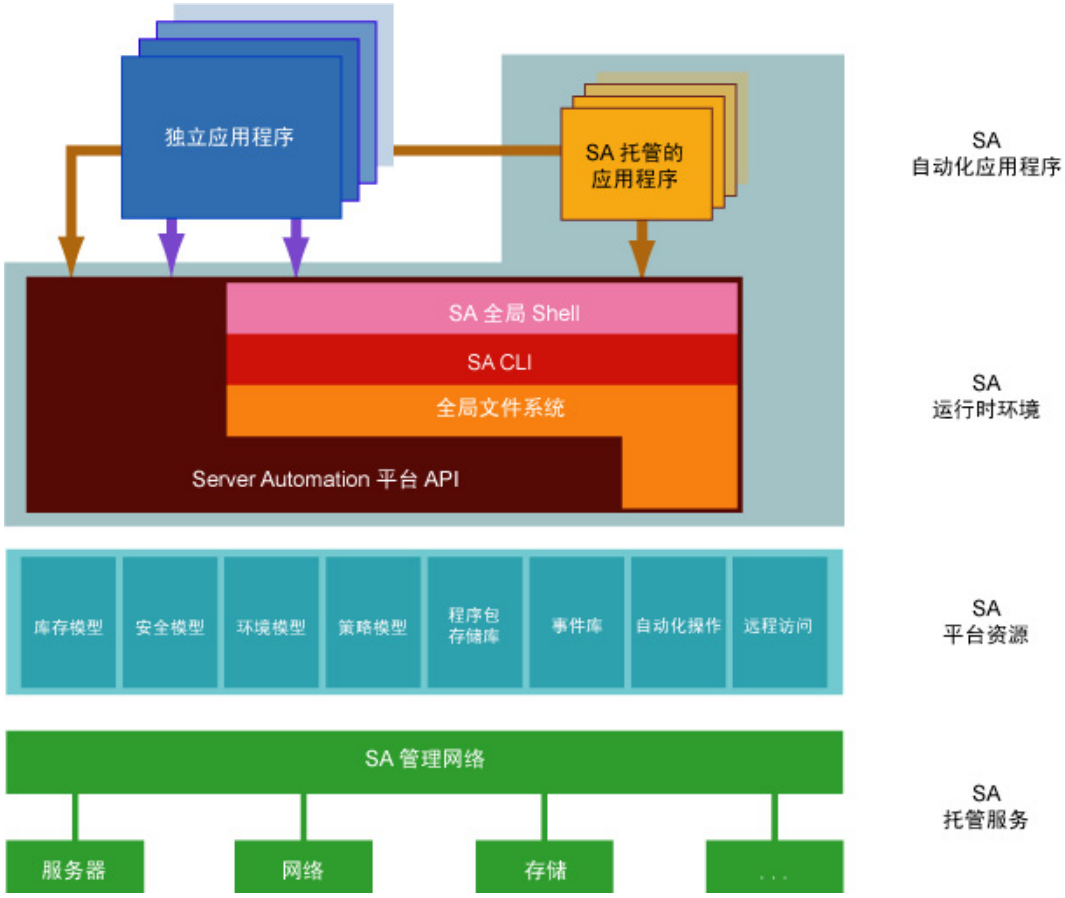
使用 Server Automation 平台，您可以执行以下任务：

- 构建新自动化应用程序和扩展 SA 以提高 IT 生产力，并契合您的 IT 策略。
- 与其他 IT 系统交换信息，例如与现有的监控、问题工单处理、计费和虚拟化技术系统交换信息。
- 使用 SA 模型库存储和组织关于运营、环境和资产的关键 IT 信息。
- 自动管理各种应用程序和操作系统。
- 将现有 Unix 和 Windows 脚本包含到 SA 中，使脚本能够在经审核的安全环境中运行。

Server Automation 平台的组件

图 1 显示了 Server Automation 平台的主要元素。

图 1　Server Automation 平台组件



如图 1 所示，该平台包括以下五个关键元素。将在随后的章节中更详细地讨论每一个元素。

- **SA 自动化应用程序：**应用程序用户在平台的基础上进行编写。这些应用程序可以是运行于正在运行的 SA 环境中的 SA 托管应用程序，也可以是运行于现有业务和管理系统环境中的独立应用程序。
- **SA 运行时环境：**提供了一系列预置的强大运行时服务，以及一个独立于语言的相应编程模型。该编程模型经过专门设计，可供各种程序员轻松访问，从脚本编写人员到 Web 开发人员再到经验丰富的企业 Java 程序员都能够轻松访问它。
- **SA 平台资源：**使开发人员能够轻松访问平台的丰富数据对象和自动化操作（如修补程序、配置和审核）以及各种功能（例如，远程访问每个托管服务器的运行时环境）。
- **SA 管理网络：**一套强大的连接、安全和缓存技术，无论设备的位置、 IP 地址空间、带宽可用性等特性如何，平台都能够访问设备。
- **SA 托管设备：**通过 SA 管理网络连接到平台的托管服务器和网络设备。

SA 自动化应用程序

如图 1 所示，自动化应用程序位于堆栈的顶部。它们是用户在平台的基础上编写的应用程序。

自动化应用程序可以是在 SA 运行时环境中运行的 SA 托管应用程序，也可以是在完全独立的环境中运行的独立应用程序。独立应用程序可通过 Web 服务调用远程访问平台。

用户可以在几分钟内编写像简单 Unix shell 脚本一样的简单应用程序。较复杂应用程序（例如与现有源代码控制或工单处理系统集成）的编写时间可能要长一点，并且可能涉及 Python、 Microsoft .NET 或 Java 编码。在这些情况下，平台被设计为独立于语言的系统，可供各类开发人员轻松采用。

SA 运行时环境

平台堆栈接下来的是 SA 运行时环境，它提供了一系列预置的强大运行时服务，以及一个独立于语言的相应编程模型。 SA 托管的应用程序在 SA 运行时环境中运行。

运行时环境核心包含两个组件：全局 Shell 和全局文件系统。这两个组件一起工作，对所有托管设备进行组织，并提供人们熟悉的 Linux/Unix shell “文件和目录”范例的设备访问方式。

SA 全局 Shell

全局 Shell 是全局文件系统 (OGFS) 的命令行界面。此命令行界面通过在终端窗口中运行的 Linux shell （例如 `bash`）显示。 OGFS 将 SA 数据模型以及托管服务器的内容 （包括文件）统一放到一个虚拟文件系统中。

全局文件系统

OGFS 以文件目录和文本文件的层次结构表示平台数据模型中的对象 （如设施、客户和设备组）以及平台托管设备上的可用信息 （如托管网络设备的配置设置或托管服务器的文件系统）。例如，在 OGFS 中，`/opsw/Customer` 目录包含客户对象的详细信息，`/opsw/Server` 目录包含托管服务器的信息。 `/opsw/Server` 目录还包含反映托管服务器内容 （如文件系统和注册表）的子目录。

该“文件和目录”范例使管理员能够熟悉 shell 脚本以轻松编写脚本，这些脚本通过遍历表示服务器的目录，可在不同服务器上执行相同任务。在后台，全局文件系统安全地向每个托管服务器提供和执行脚本中的任何逻辑。

设备的内容可以通过全局文件系统访问。全局文件系统是一个虚拟文件系统，表示 SA 和 Network Automation (NA) 管理的所有设备。在得到必要安全授权的情况下，最终用户和自动化应用程序可以通过 OGFS 浏览到远程服务器的文件系统。在 Windows 服务器上，管理员还可以访问注册表、 II 元数据库和 COM+ 对象。

SA 命令行界面

SA 命令行界面 (CLI) 为系统管理员和平台自动化应用程序提供了从命令行调用自动化任务（例如，配置软件、为设备安装修补程序或运行审核）的方法。通过丰富的语法，用户可将大量对象类型表示为输入内容，或将它们作为 CLI 调用操作的输出内容接收。

CLI 实际上是在平台 API 的基础上以编程方式生成的。将在下一节讨论平台 API。这样做的好处是，一旦开发人员向平台 API 添加新 API，将会自动为其提供相应的 CLI 方法。换句话说，在产品中提供新功能之后，平台中将立即出现相应的 CLI 方法。

SA 平台 API

SA 平台 API 是 SA 的 Win32 API：它定义了一系列应用程序编程接口，用于获取和设置值，以及执行操作。 SA 用户界面 （包括 SA 客户端和 SA 命令行界面 (CLI)）都建立在 SA 平台 API 基础之上。该 API 中包括 Java RM 客户端的库，以及基于 SOAP 的 Web 服务客户端的 WSDL。在具有 Web 服务支持的情况下，程序员可以使用 Perl、 C# 和 Python 等流行语言创建客户端。

SA 平台资源

SA 平台资源位于 SA 运行时环境下，开发人员可访问其中丰富的对象和操作，并在自己的应用程序中重用和操作它们。

库存模型

库存模型提供了 SA 收集的关于每个托管设备的所有信息，例如品牌、制造商、 CPU、操作系统、已安装的软件等。库存信息通过 SA API 提供，也作为全局文件系统中的文件提供 （在 `attr` 子目录中）。库存模型中包括服务器和网络设备等对象。

管理员可以扩展与库存对象关联的数据。例如，如果用户希望存储设备图片、租赁过期日期以及设备所插入的 UPS 的 ID，该平台可以很容易地将这些特性添加到每个设备记录。然后，用户可以添加、删除和处理这些特性，就像处理预置特性一样。

安全模型

安全模型使开发人员能够利用内置的 SA 身份验证和授权安全系统。

平台的所有客户端（SA 提供的管理应用程序、脚本以及最终用户界面）都由相同的安全框架控制。

安全管理员 （而非开发人员）将创建用户角色并授予权限。开发人员可以在自己的应用程序环境中重用所有这些用户角色和权限。例如，网络管理员可以编写 shell 脚本，将其与其他网络管理员共享，并确保这些网络管理员只能在他们有权管理的网络设备上运行该脚本。

授权机制将访问权限控制在多个级别：用户可以执行的任务类型、任务访问的服务器和网络设备，以及 SA 对象 （如软件策略）。

环境模型

环境模型定义了设备所在的整体业务环境。在通常情况下，设备属于一个或多个客户，位于特定设施中，并属于一个或多个组。平台使应用程序开发人员可以访问其中的每个对象 （客户设施、设备组和其他对象）。

与库存对象一样，环境对象也很容易扩展。这使某些操作变得简单，例如，定义在特定数据中心使用的 SNMP 陷阱接收器等特性、仅在特定设施中可用的打印机，或仅由特定业务单位使用的 Apache 配置。

策略模型

策略模型使开发人员能够访问在 SA 中定义的所有最佳实践。策略描述了服务器或网络设备上的期望状态。例如，修补程序策略描述了应在服务器上安装的修补程序，而软件策略描述了应在服务器上安装的软件，等等。

主题内容专家将定义这些策略。任何授权的系统管理员都可使用这些策略来审核设备，以便发现设备上的实际对象是否与需要的对象不同。程序员可以访问这些完整的策略库，以便在自己的应用程序中使用。

软件策略被组织到一些文件夹中，这些文件夹可以定义安全边界。也就是说，应用程序将只能访问他们的用户有权访问的软件策略。

程序包存储库

包存储库使开发人员能够访问 SA 中存储的所有软件和修补程序。其中包括操作系统内部版本、操作系统修补程序、中间件、代理以及用户已上载到 SA 的任何其他软件。

事件库

事件库中包含在执行操作 （通过用户界面或通过平台以编程方式执行）时， SA 生成的数字签名的审核跟踪。与其他平台对象一样，这些事件通过编程方式提供。

自动化操作

自动化操作允许开发人员以编程方式启动 SA 可对托管设备执行的任何操作，从运行审核到配置软件再到应用最新操作系统修补程序等。

通过平台可访问 SA 客户端中的最终用户可用的相同功能。这些功能包括一些任务，例如安装修补程序、配置操作系统，以及安装和删除软件策略。实际上，SA 客户端将调用通过 SA 运行时环境以编程方式显示的相同 API。

远程访问

远程访问功能允许开发人员以编程方式访问托管设备的文件系统（对于服务器）和执行环境（对于所有设备）。开发人员可以轻松编写应用程序来检查文件或特定软件包是否存在、运行操作系统命令以检查磁盘使用情况，或者运行系统脚本来执行例行维护任务。

SA 管理网络

管理网络是一个强大的技术组合，使开发人员能够安全地访问所管理的任何设备。管理网络提供了几个重要服务：

- **连接**：允许平台（以及自动化应用程序）访问任何托管设备。
- **安全**：包括基于 SSL/TLS 的加密、身份验证和消息完整性。
- **地址空间虚拟化**：使平台能够在多个重叠的 IP 地址空间中定位服务器。大多数复杂企业网络都具有多个专用 IP 地址空间。
- **可用性**：允许系统体系结构定义任何给定托管设备的冗余路径，即使任何给定网络路径发生故障，也仍然能够访问设备。
- **缓存**：允许服务器从较近的服务器下载软件和修补程序，而不是从较远服务器下载，从而节省时间和网络连接费用。
- **带宽限制**：让系统体系结构确定在 SA 通过网络向特定设备传输数据时， SA 及其任何应用程序可使用的带宽。
- **最低成本路由**：允许系统设计人员设置规则以管理要使用哪个路径访问特定设备，从而最大限度降低网络连接成本。

SA 托管设备

平台堆栈的底部是管理的实际设备。平台可管理超过 65 个服务器操作系统版本，以及超过 35 个不同的网络设备供应商，出厂支持数千个设备型号 / 版本。

支持的设备列表在不断更新。平台开发人员和脚本编写人员将从此设备列表受益，因为他们的自动化应用程序可以在熟悉的相同编程环境中持续访问内容不断丰富的托管设备库存。

SA 平台的优势

SA 平台具有以下主要优势。

强大的安全性

平台提供了下列全面的安全机制，因此开发人员无需在自己的应用程序中提供这些机制。

- **安全通信通道**：从自动化应用程序到托管设备的端到端通信将进行加密和身份验证。
- **基于角色的访问控制**：平台采用 SA 中内置的基于角色的访问控制，开发人员能够轻松共享其应用程序，并确保它们只在管理员已授予访问权限的设备上运行。
- **数字签名的审核跟踪**：在自动化应用程序运行后，平台将生成数字签名的审核跟踪，捕获运行应用程序的用户、应用程序的执行时间，以及应用程序所在的设备。
- **全面访问**：平台提供对所有设备的全面访问，使系统管理员和开发人员能够轻松访问设备。
- **市场领先的平台覆盖范围**：支持的设备包括 65 个以上的服务器操作系统版本，以及 1000 多种网络设备。
- **位于任何物理位置**：设备可以位于世界上的任何位置，无论是在主要的数据中心、零售商店还是分支办事处。
- **位于任何 IP 地址空间中**：因为平台支持多个重叠的 IP 地址空间，设备可以属于任何 IP 地址空间。
- **在 DMZ 中**：设备可以位于 DMZ 或其他难以访问的网络空间中，因此开发人员或系统管理员无需担心设备访问权限的具体细节 （例如，通过防御主机）。

丰富的服务

平台提供了基础自动化系统中的所有相关数据和操作：

- **丰富的预置数据**：开发人员可以很方便地访问丰富的数据，这些数据一部分由平台自身生成（例如，设备库存数据和设施信息），一部分由与平台交互的用户生成（例如，设备组客户、最佳实践策略以及上载的软件、修补程序和脚本）。开发人员可以轻松编写用于读取和写入这些数据的应用程序。
- **可扩展的数据存储**：开发人员可以轻松扩展本地平台对象，以包含自己的数据。可以扩展设备库存模型，以包含平台本身不包含的特性。可以扩展客户和设施对象，以包含用于引导对该客户相关设备进行配置或审核的特性。
- **自动化任务**：平台几乎向开发人员公开了基础自动化系统的所有功能：安装修补程序、配置、审核及其他。这使得开发人员能够编写跨多个系统的复杂工作流，以便轻松从自动化应用程序环境中调用这些操作。

可供各种程序员轻松访问

平台经过专门设计，适用于各种开发人员，包括 Unix shell 和 Visual Basic 脚本编写人员、Perl 和 Python 程序员以及企业 .NET 或 Java 程序员等。平台的运行时服务层支持以“文件和目录”范例提供大多数平台对象，并从命令行界面 (SA CLI) 获取大多数平台服务。这允许习惯编写 shell 脚本的系统管理员可以立即使用平台，而无需学习使用新的编程语言和工具。他们可以使用自己喜欢的文本编辑器 （熟悉的 Unix shell 程序）开始工作，然后快速开发脚本。

对于更复杂的应用程序以及与现有系统的集成，系统程序员可以使用任何具有 Web 服务绑定的编程工具和语言。

SA 平台 API 设计

平台 API 由 Java 接口定义，并被组织到 Java 包中。为了支持多种客户端语言和远程访问协议，API 采用面向函数的 “按值调用”模型。

服务

在平台 API 中，一个服务封装一组相关函数。每个服务通过 Java 接口指定，其名称以 `Service` 结尾，例如 `ServerService`、 `FolderService` 和 `JobService`。

服务是 API 的入口点。要访问 API，客户端将调用由服务器接口定义的方法。例如，要检索托管服务器上已安装的软件的列表，客户端将调用 `ServerService` 接口的 `getInstalledSoftware` 方法。其他 `ServerService` 方法示例包括 `checkDuplex`、 `setPrimaryInterface` 和 `changeCustomer`。

SA 平台 API 包含 70 多个服务 – 数量太多，无法在这里一一描述。表 1 列出了一小部分您可能需首先使用到的服务。有关完整服务列表，请在浏览器中转到 API 文档和 Twister （第 21 页）中所示的 URL。

表 1　SA API 的部分服务列表

服务名称	该服务提供的部分操作
`AuditTaskService`	创建、获取和运行审核任务。
`ConfigurationService`	创建应用程序配置，使用应用程序配置获取软件策略。
`DeviceGroupService`	创建设备组，向组分配设备，获取组成员，设置动态规则。
`EventCacheService`	触发操作，例如更新值对象的客户端缓存。请参见事件缓存 （第 20 页）。

表 1　SA API 的部分服务列表 （续）

服务名称	该服务提供的部分操作
FolderService	创建文件夹，获取文件夹的子级，设置文件夹的客户，移动文件夹。
InstallProfileService	创建、获取和更新操作系统安装配置文件。
JobService	获取作业进度和结果，取消作业，更新作业计划。
NasConnectionService	获取 NA 服务器的主机名，在 NA 服务器上运行命令。
NetworkDeviceService	根据指定的搜索筛选器获取系列、名称、型号和类型等信息。
SequenceService	创建、获取和运行操作系统序列，在服务器上安装操作系统。
ServerService	获取服务器信息，协调（修正）服务器上的策略（安装软件），获取和设置自定义字段和特性，执行操作系统序列 （安装操作系统）。
SoftwarePolicyService	创建软件策略，向服务器分配策略，获取策略内容，修正 （协调）服务器策略。
SolPatchService	安装和卸载 Solaris 修补程序，添加策略覆盖。
VirtualColumnService	管理自定义字段和自定义特性。
WindowsPatchService	安装和卸载 Windows 修补程序，添加策略覆盖。

API 中的对象

虽然 SA 平台 API 是面向函数的，但是它的设计允许客户端创建面向对象的库。 SA 数据模型包括服务器、文件夹和客户等对象。这是对象是永久对象，也就是说，它们存储在模型库中。在 API 中，这些对象具有以下项：

- 一个用于定义对象行为的服务。例如， ServerService 的方法将指定托管服务器对象的行为。
- 一个表示永久对象实例的对象 （身份）引用。例如， ServerRef 是用于唯一标识托管服务器的引用。在 ServerService 中，大多数方法的第一个参数是 ServerRef，它标识由该方法操作的托管服务器。 ServerRef 的 Id 特性是存储在模型库中的服务器对象的主键。
- 一个或多个表示永久对象数据成员 （特性、字段）的值对象 (VO)。例如， ServerVO 包含 agentVersion 和 loopbackIP 等特性。 ServerHardwareVO 的特性包括 manufacturer、model 和 assetTag。大多数特性不能由客户端应用程序更改。如果特性可以更改，则 setter 方法的 API 文档中将包含 “Field can be set by clients”。

出于性能方面的考虑，对永久对象的更新操作是粗粒度的。例如， ServerService 的 update 方法接受完整 ServerVO 而非单个特性作为参数。

异常

特定于 SA 的所有 API 异常都派生自以下异常之一：

- OpswareException - 在发生应用程序级错误的情况下引发，例如，当最终用户输入将传递给方法的无效值时。通常情况下，客户端应用程序可以从这类异常中恢复。派生自 OpswareException 的异常示例包括 NotFoundException、 NotInFolderException 和 JobNotScheduledException。
- OpswareSystemException - SA 中发生错误时引发。通常，SA 管理员必须首先解决问题，然后客户端应用程序才能运行。

以下是与安全相关的异常：

- AuthenticationException - 在指定了无效的 SA 用户名或密码时引发。
- AuthorizationException - 在用户无权执行操作或访问对象时引发。有关权限的详细信息，请参见 《SA 管理指南》。

事件缓存

某些客户端应用程序需要保留 SA 对象的本地副本。缓存由客户端通过 EventCacheService 访问，其中包含用于描述对 SA 对象执行的最近更改的事件。客户端可以定期轮询缓存，以便检查是否已创建、更新或删除对象。该缓存将保留配置的滑动时间窗口内的事件。默认情况下，会保留最近两小时内的事件。要更改此滑动窗口的大小，请按照 《SA 管理指南》中所述编辑 Web 服务数据访问引擎配置文件。

搜索

SA 平台 API 的搜索机制将根据值对象的的特性（字段）检索对象引用。例如， getServerRefs 方法按 ServerVO 值对象的特性执行搜索。 getServerRefs 方法具有以下签名：

```
public ServerRef[] getServerRefs(Filter filter)...
```

每个 get*Refs 方法都接受 filter 参数，即用于指定搜索条件的对象。使用简单表达式的 filter 参数具有以下语法：

value-object.attribute operator value

（该语法已简化。有关完整的定义，请参见筛选器语法 （第 157 页）。）

下面的示例是 getServerRefs 方法的 filter 参数：

```
ServerVO.hostName = "d04.example.com"
ServerVO.model BEGINS_WITH "POWER"
ServerVO.use IN "UNKNOWN" "PRODUCTION"
```

允许使用复杂表达式，例如：

```
(ServerVO.model BEGINS_WITH "POWER") AND (ServerVO.use = "UNKNOWN")
```

并不是值对象的每个特性都可以在 filter 参数中指定。例如，允许在 filter 参数中指定 ServerVO.state，但不允许指定 ServerVO.OsFlavor。要找出允许的特性，请在 API 文档中找到值对象，并查找注释 “Field can be used in a filter query”。

安全

SA 平台的用户必须进行身份验证和授权，以在 SA 自动化平台 API 上调用方法。要连接到 SA，客户端需提供 SA 用户名和密码（身份验证）。要调用方法，SA 用户必须属于具有必要权限（授权）的用户组。这些权限不仅会限制用户可执行的操作类型，还会限制对在操作中使用的服务器和网络设备的访问权限。

SA 管理员必须首先在命令中心中指定所需的用户和权限，然后应用程序客户端才能在平台上运行。有关说明，请参见《SA 管理指南》的 “用户组和设置”一章。有关与安全相关的异常的信息，请参见异常 （第 20 页）。

客户端与 SA 之间的通信已加密。对于 Web 服务客户端，将使用通过 HTTP over SSL (HTTPS) 加密请求和响应 SOAP 消息 （执行操作调用）。

API 文档和 Twister

SA 包括用于描述 SA 平台 API 的 API 文档 (Javadoc)。要访问 API 文档，请在浏览器中指定以下 URL：

```
https://<SA_core_host>/twister
```

其中 *<SA_core_host>* 是运行命令中心组件的 SA 核心服务器的 IP 地址或主机名。

Twister 是一个程序，允许您在浏览器中一次调用一个 API 方法。例如，要调用 ServerService.getServerVO 方法，请执行以下步骤：

1. 在浏览器中打开 API 文档。
2. 在 “All Classes”窗格中，选择 com.opsware.server。
3. 在 com.opsware.server 窗格中，选择 ServerService。
4. 在主窗格中，向下滚动到 getServerVO 方法。
5. 对 getServerVO 方法单击 “尝试”。
6. 输入您的 SA 用户名和密码。
7. 在 ServerService.getServerVO 的 “Twister”窗格中，在 oid 字段中输入托管服务器的 ID。

8 单击“运行”。“Twister”窗格将显示返回的 ServerVO 对象的特性。

常量字段值

一些 API 值对象 (VO) 的字段值被定义为常量。例如， JobInfoVO 具有 status 字段，该字段可具有由常量定义的值，如 STATUS_ACTIVE、 STATUS_PENDING 等。 API 将常量指定为 Java static final 字段，但是从 API 生成的 WSDL 不定义常量。要查看常量的定义，请在 API 文档中转到“Constant Field Values”页面：

```
https://<SA_core_host>/twister/docs/constant-values.html
```

例如，“Constant Field Values”页面将 STATUS_ACTIVE 定义为整数 1。

支持的客户端

SA 平台支持具有不同技能的程序员，从编写 shell 脚本的系统管理员到熟悉最新工具和技术的 .NET 和 Java 程序员等。所有支持的客户端都调用一组相同的方法，这些方法被组织到 SA 平台的服务中。开发人员可以创建以下类型的客户端以在 SA 平台 API 中调用方法：

- **SA 命令行界面 (CLI)**：shell 脚本从全局 Shell 会话启动，可通过调用 CLI 方法来访问 SA 平台 API，它们是 OGFS 中的可执行程序。每个 CLI 方法对应于 API 中的一个方法。
- **Web 服务**：这些客户端使用 SOAP over HTTPS 向 SA 发送请求并获得响应。Web 服务操作（在 WSDL 中定义）对应于 API 中的方法。开发人员可以使用 Perl 和 C# 等流行语言编写 Web 服务客户端。
- **Java RMI**：这些客户端从其他 Java 虚拟机调用远程 Java 对象。
- **Pytwist**：这些 Python 程序可以在 SA 核心或托管服务器上运行。

Web 服务和 Java RMI 客户端可以在不同于 SA 核心或托管服务器的服务器上运行。CLI 方法将在安装 OGFS 的核心服务器上的全局 Shell 会话中执行。

2 SA CLI 方法

SA CLI 方法概述

最终用户通过 SA 客户端访问 SA。有时，高级用户需要在命令行环境中访问 SA，以便在多个服务器上执行批量操作或重复性任务。在 SA 中，命令行环境由全局 Shell (OGSH)、全局文件系统 (OGFS) 和 SA 命令行界面 (CLI) 方法组成。

要通过命令行执行 SA 操作，可从 OGSH 会话中调用 SA CLI 方法。SA CLI 方法是 OGFS 中的可执行文件，对应于 SA API 中的方法。当运行 SA CLI 方法时，将调用基础 API 方法。

为了理解本章，您需要熟悉 OGSH 和 OGFS。有关详细信息，请参见 《SA 用户指南：Server Automation》中的 OGSH。

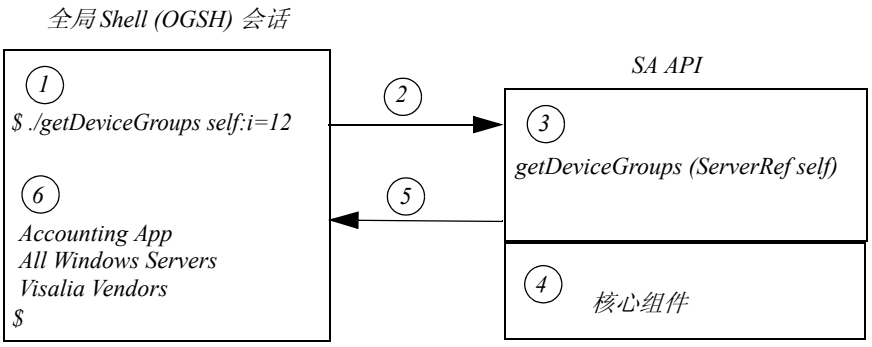
有关 `oupload` 和 `odownload` 命令的信息，请参见《SA 用户指南：Server Automation》中的 OCLI 1.0。

方法调用

如图 2 所示，当在 OGSH 会话中调用 SA CLI 方法时，将执行以下操作：

1. OGSH 解析您输入的命令和参数，以便确定 API 方法。
2. OGSH 调用基础 API 方法。
3. 授权检查过程将验证用户是否有权执行此操作。然后 SA 执行操作。
4. API 方法将结果传送回 SA CLI 方法。
5. SA CLI 方法将返回值写入 OGSH 会话的 `stdout`。如果引发异常，SA CLI 方法将返回非零值状态。

图 2　SA CLI 方法调用概述



安全

SA CLI 方法与 SA 客户端使用相同的身份验证和授权机制。当您启动 OGSH 会话时，SA 会对 SA 用户进行身份验证。当运行 SA CLI 方法时，将执行授权操作。要成功运行 SA CLI 方法，您的 SA 用户必须属于具有所需权限的组。有关安全的详细信息，请参见 《SA 管理指南》。

API 与 SA CLI 方法之间的映射

OGFS 以目录结构表示 SA 对象，以文本文件表示对象特性，以可执行文件表示 API 方法。这些可执行文件即 SA CLI 方法。每个 SA CLI 方法与一个基础 API 方法相匹配。对于这两种类型的方法，方法名称、参数和返回值都相同。

例如， `setCustomer` API 方法具有以下 Java 签名：

```
public void setCustomer(ServerRef self,
                        CustomerRef customer)...
```

在 OGFS 中，对应的 SA CLI 方法具有以下语法：

```
setCustomer self:i=server-id customer:i=customer-id
```

请注意，参数名称 （`self` 和 `customer`）在这两种语言中是相同的。（`:i` 符号称为格式说明符，将在本章后文进行讨论。 ）在该示例中，返回类型为 void，因此， SA CLI 方法不会将结果写入 `stdout`。有关 SA CLI 方法如何返回用于代表对象的字符串的信息，请参见返回值 （第 39 页）。

SA CLI 方法与 Unix 命令之间的差异

虽然在 OGSH 中可以运行 Unix 命令和 SA CLI 方法，但是 SA CLI 方法在以下几个方面存在不同：

- 与许多 Unix 命令不同， SA CLI 方法不从 `stdin` 中读取数据。因此，您不能在一组由管道 (|) 连接的命令中插入 SA CLI 方法。（但是， SA CLI 方法会将数据写入 `stdout`。）
- 大多数 Unix 命令可接受标志和值形式的参数 （例如， `ls -l /usr`）。在 SA CLI 方法中，命令行参数是以等号连接的 “名称 - 值” 对。

- Unix 命令基于文本：它们接受并返回字符串形式的数据。相比之下， SA CLI 方法可以接受和返回复杂的对象。
- 使用 SA CLI 方法时，您可以指定参数和返回值的格式。 Unix 命令则没有同等功能。

SA CLI 方法教程

本教程介绍 SA CLI 方法以及一些示例，您可以在自己的环境中试着使用这些示例。完成本教程后，您应该能够运行 SA CLI 方法、检查 SA 对象的 `self` 文件，并创建用于在多个服务器上调用 SA CLI 方法的脚本。

在开始本教程之前，您需要具有以下能力：

- 可以登录到 SA 客户端。
- 您的 SA 用户至少在一个托管服务器上具有 “读取和写入”权限。这些权限通常由安全管理员分配。《SA 管理指南》中讨论了这些权限。
- 您的 SA 用户在同一个托管服务器上拥有所有 OGSH 权限。有关这些权限的信息，请参见 《SA 用户指南：Server Automation》中的 “aaa 实用程序”一节。
- 熟悉 OGSH 和 OGFS。如果是首次使用这些功能，请先参见 《SA 用户指南：Server Automation》中的全局 Shell 部分，然后再继续本教程。

本教程中的示例命令将在名为 `abc.example.com` 的 Windows 服务器上运行。该服务器属于名为 “All Windows Servers”的服务器组。在尝试执行这些命令时，请使用您有权访问的托管服务器的主机名替换 `abc.example.com`。

1 打开 OGSH 会话。

您可以从 SA 客户端中打开全局 Shell 会话。在 “操作”菜单中，选择 “全局 Shell”。也可以从在桌面上运行的终端客户端中打开 OGSH 会话。有关说明，请参见 《SA 用户指南：Server Automation》中的 “打开全局 Shell 会话”。

2 列出服务器的 SA CLI 方法。

特定服务器的 `method` 子目录中包含一些可执行文件，也就是您可以对该服务器运行的方法。以下示例列出了 `abc.example.com` 服务器的 SA CLI 方法：

```
$ cd /opsw/Server/@/abc.example.com/method
$ ls -1
addDeviceGroups
attachPolicies
attachVirtualColumn
checkDuplex
clearCustAttrs
...
```

这些方法具有实例上下文 - 它们作用于一个特定的服务器实例（在本示例中，为 `abc.example.com`）。可以从方法的路径推断服务器实例。步骤 5 讨论了具有静态上下文的方法。

3 运行不带参数的 SA CLI 方法。

要显示 abc.example.com 所属的公共服务器组，请调用 getDeviceGroups 方法：

```
$ cd /opsw/Server/@/abc.example.com/method
$ ./getDeviceGroups
Accounting App
All Windows Servers
Visalia Vendors
```

4 运行带参数的方法。

方法的命令行参数以 “名称 - 值” 对的形式表示，各个参数之间由空白字符分隔。在 setCustomer 的以下调用中，参数名称是 customer 且值为 20039。参数名称末尾的 :i 是 ID 格式说明符，将在后面的步骤中讨论它。

以下方法调用将 abc.example.com 服务器的客户从 Opsware 更改为 C39。客户 C39 的 ID 是 20039。

```
$ cd /opsw/Server/@/abc.example.com
$ cat attr/customer ; echo
Opsware
$ method/setCustomer customer:i=20039
$ cat attr/customer ; echo
C39
```

5 列出托管服务器的静态上下文方法。

静态上下文方法驻留在 /opsw/api 目录中。这些方法未被限制到某个特定对象实例。要列出服务器的静态方法，请输入以下命令：

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ls
```

列出的方法与步骤 2 中显示的方法相同。

6 运行带 self 参数的方法。

该步骤会将 getDeviceGroups 作为静态上下文方法调用。与步骤 3 中所示的的实例上下文方法不同，静态上下文方法需要使用 self 参数来标识服务器实例。

例如，假设 abc.example.com 服务器的 ID 为 530039。要列出该服务器组，请输入以下命令：

```
$ cd /opsw/api/com/opsware/server/ServerService/method
$ ./getDeviceGroups self:i=530039
Accounting App
All Windows Servers
Visalia Vendors
```

将对 getDeviceGroups 的此调用与步骤 3 中用于演示实例上下文的调用相比较。这两个调用在 API 中运行相同的基础方法，并返回相同的结果。

7 检查服务器的 self 文件。

在 SA 中，每个托管服务器都是一个对象。但是，OGFS 是文件系统，而不是对象模型。self 文件提供了对 SA 对象的各种表示形式的访问方法。这些表示形式包括 ID、名称和结构。

服务器的默认表示形式是服务器名称。例如，要显示服务器的名称，请输入以下命令：

```
$ cd /opsw/Server/@/abc.example.com
$ cat self ; echo
abc.example.com
```

如果您知道服务器的 ID，则可以从 self 文件中获取其名称，如以下示例所示：

```
$ cat /opsw/.Server.ID/530039/self ; echo
abc.example.com
```

8 在 self 文件上指示 ID 格式说明符。

要选择 self 文件的特定表示形式，请输入一个句点，然后键入文件名，再键入格式说明符。例如，以下 cat 命令中包含格式说明符 (:i) 以显示服务器 ID：

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:i ; echo
com.opsware.server.ServerRef:530039
```

此输出显示 abc.example.com 的 ID 是 530039。 com.opsware.server.ServerRef 是服务器引用的类名，它是 SA API 中的相应对象。

前导句点在文件和方法返回值上的格式说明符中是必需的，但不通过方法参数指示。

9 指示结构格式说明符。

结构格式说明符 (:s) 用于表示复杂对象的特性。这些特性将显示为“名称 - 值”对，所有特性都包含在大括号内。结构格式用于在命令行上指定作为复杂对象的方法参数。（有关方法调用的示例，请参见复杂对象和数组作为参数 （第 38 页）。）

以下示例使用结构形式显示了 abc.example.com：

```
$ cd /opsw/Server/@/abc.example.com
$ cat .self:s ; echo
{
managementIP="192.168.8.217"
modifiedBy="spujare"
manufacturer="DELL COMPUTER CORPORATION"
use="UNKNOWN"
discoveredDate=1149012848000
origin="ASSIMILATED"
osSPVersion="SP4"
locale="English_United States.1252"
reporting=false
netBIOSName=
previousSWReg=1150673874000
osFlavor="Windows 2000 Advanced Server"
. . .
```

服务器的特性也可由 attr 目录中的文件表示，例如：

```
$ pwd
```

```
/opsw/Server/@/abc.example.com
$ cat attr/osFlavor ; echo
Windows 2000 Advanced Server
```

10 创建用于调用 SA CLI 方法的脚本。

此步骤中的示例脚本遍历了“All Windows Servers”公共服务器组中的所有服务器。在每个服务器上，脚本将运行 getCommCheckTime SA CLI 方法。

首先，返回到 OGFS 中的主目录：

```
$ cd
$ cd public/bin
```

然后，运行 vi 编辑器：

```
$ vi
```

在 vi 中，插入下列行以创建 bash 脚本：

```
#!/bin/bash
# iterate_time.sh

METHOD_DIR="/opsw/api/com/opsware/server/ServerService/method"
GROUP_NAME="All Windows Servers"
cd "/opsw/Group/Public/$GROUP_NAME/@/Server"

for SERVER_NAME in *
do
   SERVER_ID=`cat $SERVER_NAME/.self:i`
   echo $SERVER_NAME
   $METHOD_DIR/getCommCheckTime self:i=$SERVER_ID
   echo
   echo
done
```

在 vi 中保存文件，并将文件命名为 iterate_time.sh。退出 vi。

使用 chmod 更改 iterate_time.sh 的权限，然后运行它：

```
$ chmod 755 iterate_time.sh
$ ./iterate_time.sh
abc.example.com
2006/06/20 16:46:56.000
. . .
```

格式说明符

格式说明符指示值在 SA CLI 环境的显示或解释方式。可以向方法参数、方法返回类型、self 文件和对象特性应用格式说明符。要指示格式说明符，请在某个字母前加上一个冒号，如表 2 中所示。

如果为文件或方法返回值指示格式说明符，则必须在文件或方法名称前加上一个句点。对于具有格式说明符的方法返回值，前导句点将不包括在内。

表 2　格式说明符摘要

格式说明符	描述	有效对象类型	是否允许用作方法参数？
:n	**名称**：用于标识对象的字符串。建议使用唯一名称，但不是必须使用唯一名称。对于没有名称的对象，此表示形式与 ID 表示形式相同。	SA 对象	是。如果名称不明确，会发生错误。
:i	**ID**：唯一指示对象类型及其 SA ID 的格式。也成为对象参考。	SA 对象； 日期 (java.util.Calendar) 对象	是。如果已在上下文中清除类型，则可忽略类型。
:s	**结构**：用于在命令行上指定复杂值的精简表示形式。特性包含在大括号中。	任何复杂对象	是
:d	**目录**：以 OGFS 中的目录表示特性。	任何用作特性的复杂对象。该表示形式不能用于方法参数或返回值。	否

格式说明符的位置

格式说明符紧跟在其作用于的项之后。对于文件，格式说明符跟在文件名后面。在以下示例中，请注意前导句点：

```
cat .self:s
```

当应用于方法返回类型时，格式说明符跟在方法名称的后面。以下调用显示了所返回组的 ID：

```
./.getDeviceGroups:i
```

在方法参数中，格式说明符位于参数名称后、等号前，如以下示例所示：

```
./setCustomer self:i=9977 customer:i=239
```

具有格式说明符的方法参数没有前导句点。

默认格式说明符

每个值或对象都具有默认的格式说明符。例如，名称格式说明符是 osVersion 特性的默认格式说明符。以下两个 cat 命令生成相同的输出：

```
cd /opsw/Server/@/d04.example.com/attr
cat osVersion
cat .osVersion:n
```

名称格式说明符是模型库中存储的 SA 对象 （例如服务器和客户）的默认格式说明符。结构格式说明符是其他复杂对象的默认格式说明符。

ID 格式说明符示例

以下示例显示了 d04.example.com 服务器所属的设施的 ID：

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:i ; echo
```

（前面的 echo 命令为可选。它会生成一个换行字符，使输出内容更容易阅读。分号字符分隔在同一行中输入的多个 bash 语句。）

具有 ID 格式说明符的值的输出将以 Java 类名称作为前缀。例如，如果设施值的 ID 为 39，那么前面的 cat 命令会显示以下输出：

```
com.opsware.locality.FacilityRef:39
```

以下 getDeviceGroups 方法调用列出了 d04.example.com 所属的公共服务器组的 ID：

```
cd /opsw/Server/@/d04.example.com/method
./.getDeviceGroups:i
```

有关更多 ID 格式示例，请参见 self 文件第 34 页。

结构格式说明符语法

可使用结构格式表示复杂对象。复杂对象中可以包含各种特性。您可以使用此格式来指定作为复杂对象的方法参数。有关示例，请参见复杂对象和数组作为参数 （第 38 页）。

结构格式是一系列 “名称 - 值” 对，由空白字符分隔，包含在大括号中。每个 “名称 - 值” 对表示一个特性。结构格式具有以下语法：

```
{ name-1=value-1 name-2=value-2 . . . }
```

下面是一个简单的示例：

```
{ version=10.1.3 isCurrent=true }
```

可使用任何空白字符作为分隔符：

```
{
  version=10.1.3
  isCurrent=true
}
```

可以将特性指定为结构形式，从而支持表示嵌套对象。在以下示例中， versionDesc 特性以结构表示：

```
{
program=agent
versionDesc={
        version=10.1.3
        isCurrent=true
        comment="Latest version"
        }
}
```

要在一个结构中指定数组，请重复特性名称。以下结构中包含一个名为 steps 数组，该数组具有三个元素，元素值分别为 33、 14 和 28。

```
{ moduleName="Some Initiator" steps=33 steps=14 steps=28 }
```

结构格式说明符示例

以下示例为 facility 特性指定了结构格式：

```
cd /opsw/Server/@/d04.example.com/attr
cat .facility:s
```

此 cat 命令将生成以下输出。请注意，customers 是一个数组，其中将为与该设施关联的每一位客户包含一个元素。

```
{
modifiedBy="192.168.9.246"
customers="Customer Independent"
customers="Not Assigned"
customers="Opsware Inc."
customers="Acme Inc."
. . .
ontogeny="PROD"
createdBy=
status="ACTIVE"
createdDt=-1
realms="Transitional"
realms="C39"
realms="C39-agents"
modifiedDt=1146528752000
name="C39"
displayName="C39"
}
```

以下 getDeviceGroups 调用指示返回值的结构格式说明符：

```
cd /opsw/Server/@/d04.example.com/method
./.getDeviceGroups:s
```

此 getDeviceGroups 调用将显示以下输出。由于 d04.example.com 属于两个服务器组，因此输出中包括两个结构。在每个结构中， devices 数组都包含属于该组的服务器的元素。

```
{
dynamic=true
devices="m302-w2k-vm1.dev.example.com"
```

```
devices="d04.example.com"
. . .
status="ACTIVE"
public=true
fullName="Device Groups Public All Windows Servers"
description="test"
createdDt=-1
modifiedDt=1142019861000
parent="Public"
}

{
dynamic=true
devices="opsware-nibwp.build.example.com"
devices="glengarriff.snv1.dev.example.com"
devices="millstreet"
. . .
fullName="Device Groups Public z_testsrvgroup"
. . .
}
```

结构格式说明符是用于检索值对象 (VO) 的方法的默认格式说明符。例如，以下两个 `getServerVO` 调用是等效的：

```
cd /opsw/Server/@/d04.example.com/method
./.getServerVO:s
./getServerVO
```

在该示例中， `getServerVO` 将显示以下输出：

```
{
managementIP="192.168.198.93"
modifiedBy=
manufacturer="DELL COMPUTER CORPORATION"
use="UNKNOWN"
discoveredDate=1145308867000
origin="ASSIMILATED"
osSPVersion="RTM"
locale="English_United States.1252"
reporting=false
netBIOSName=
previousSWReg=1147678609000
osFlavor="Windows Server 2003, Standard Edition"
peerIP="192.168.198.93"
modifiedDt=1145308868000
. . .
serialNumber="HVKZS51"
}
```

该结构代表 SA API 的 `ServerVO` 类。该结构中的每一个特性对应于 `attr` 目录中的一个文件。在以下示例中， `getServerVO` 和 `cat` 命令都将显示服务器的 `serialNumber` 特性：

```
cd /opsw/Server/@/d04.example.com
./method/getServerVO | grep serialNumber
cat attr/serialNumber ; echo
```

目录格式说明符示例

以下命令将当前工作目录更改为与服务器 d04.example.com 关联的客户：

```
cd /opsw/Server/@/d04.example.com/attr/.customer:d
```

以下命令列出该客户的名称：

```
cat /opsw/Server/@/d04.example.com/attr/\
.customer:d/attr/name
```

只能在需使用目录名称的命令参数中使用目录说明符。以下 cat 命令因为尝试显示目录而失败：

```
cat /opsw/Server/@/d04.example.com/attr/.customer:d # WRONG!
```

但是，以下命令是合法的：

```
ls /opsw/Server/@/d04.example.com/attr/.customer:d
```

值表示形式

因为 SA CLI 方法在 shell 环境 (OGSH) 中运行，所以它们可接受和返回字符串形式的数据。但是，基础 API 方法可以接受和返回其他数据类型，例如数字、布尔值和对象。下面的各节描述了 OGFS 和 SA CLI 方法是如何表示非字符串数据类型的。

OGFS 中的 SA 对象

SA 数据模型包括服务器、服务器组、客户和设施等对象。在 OGFS 中，这些对象以目录结构表示：

```
/opsw/Customer
/opsw/Facility
/opsw/Group
/opsw/Library
/opsw/Realm
/opsw/Server
. . .
```

以上列表并非完整的列表。要查看完整列表，请输入 ls /opsw。

对象特性

SA 对象的特性以 attr 子目录中的文本文件表示。每个文件的名称与相应特性的名称匹配。文件的内容显示特性的值。

例如， /opsw/Server/@/buzz.example.com/attr 目录包含以下文件：

```
agentVersion
codeset
createdBy
createdDt
```

```
customer
defaultGw
description
discoveredDate
facility
hostName
locale
lockInfo
loopbackIP
managementIP
manufacturer
. . .
```

要显示 buzz.example.com 服务器的管理 IP 地址，请输入以下命令：

```
cd /opsw/Server/@/buzz.example.com/attr
cat managementIP ; echo
```

自定义特性

自定义特性是可由您分配给服务器等 SA 对象的 “名称 - 值”对。在 OGFS 中，自定义特性以 CustAttr 子目录中的文本文件表示。您可以在 CustAttr 下创建新文本文件，从而在 OGSH 会话中创建自定义特性。以下示例在 buzz.example.com 服务器上创建一个值为 hello there 的自定义特性 MyGreeting：

```
cd /opsw/Server/@/buzz.example.com/CustAttr
echo -n "hello there" > MyGreeting
```

有关更多示例，请参见 《SA 用户指南：Server Automation》中的 “管理自定义特性”。

self 文件

self 文件位于服务器或客户等 SA 对象的目录中。此文件提供对当前对象的各种表示形式的访问，具体取决于格式说明符。（有关详细信息，请参见格式说明符 （第 28 页）。）

要列出 buzz.example.com 服务器的 ID，请输入以下命令：

```
cd /opsw/Server/@/buzz.example.com
cat .self:i ; echo
```

对于服务器，默认格式说明符是名称。下列命令将显示相同的输出：

```
cat self ; echo
cat .self:n ; echo
```

以下命令将以结构格式列出服务器的特性：

```
cat .self:s
```

基元值

表 3 显示了基元值是如何在 API 与其 SA CLI 方法中的字符串表示形式之间转换的。除了日期，基元值不支持格式说明符。日期支持 ID 格式说明符。

表 3　基元类型和 SA CLI 方法之间的转换

基元类型	Java 等效项	SA CLI 方法的输出	SA CLI 方法的输入
String	`java.lang.String`	字符串，以当前会话的编码显示。	字符串，从当前会话编码转换为 Unicode。
数字	`byte`、`short`、`int`、`long`、`float`、`double`；以及它们的对象等效项	十进制格式，未本地化。适用于非常大或非常小的值的科学记数法。	示例 - 十进制：`101`、`512.34`、`-104` 十六进制：`0x1F32`、`0x2e40` 八进制：`0543` 科学记数法：`4.3E4`、`6.532e-9`、`1.945e+02`
布尔值	`boolean`、`Boolean`	`true` 或 `false`	字符串 “true” 和所有大小写混合变体将计算为 `true`。所有其他值将计算为 `false`。
二进制数据	`byte[]`、`Byte[]`	二进制字符串。不从会话编码进行转换。	二进制字符串。不向会话编码进行转换。
日期	java.util.日历	日期值。默认情况下，以下列格式显示：`YYYY/MM/DD HH:MM:SS.mmm` 时间以 UTC 格式显示。如果指示 ID 格式说明符，则值显示为自新纪元开始所经过的毫秒数（UTC 格式）。	与输出相同。

数组

数组对象的表示形式取决于它们是独立对象 （数组特性文件或方法返回值），还是包含在复杂对象的结构中。

首先，独立数组对象将根据基础类型显示，并由换行字符分隔。在数组元素中，换行字符由 “\n” 转义，反斜杠字符由 “\\” 转义。

可以使用基础类型所支持的任何表示形式来输入或输出数组值。例如，默认情况下，getDeviceGroups 方法会以名称形式列出组：

```
All Windows Servers
Servers in Austin
Testing Pool
```

如果指示 ID 格式说明符，(.getDeviceGroups:i) 方法将显示组的 ID：

```
com.opsware.device.DeviceGroupRef:15960039
com.opsware.device.DeviceGroupRef:10390039
com.opsware.device.DeviceGroupRef:17380039
```

其次，包含在复杂对象的结构中的数组将表示为 “名称 - 值” 对 （使用特性作为名称）。特性将出现多次，为数组中的每个元素出现一次。特性的显示顺序将确定数组中元素的顺序。以下示例显示了一个结构，该结构包含两个特性，一个是名为 subject 的字符串，一个是名为 ranks 的包含三个元素的数字数组：

```
{ subject=”my favorites” ranks=17 ranks=44 ranks=24 }
```

数组还可以目录表示。在数组目录中，每个数组元素都具有一个对应文件（基元类型）或子目录（复杂类型）。每个条目的名称是数组元素的索引号，从零开始。

对于作为复杂对象特性的数组，应通过编辑其特性文件来修改数组。此操作将使用编辑过的文件内容完全替换数组。

对于包含复杂对象作为元素的数组，应通过更改其目录表示形式来修改数组。要更改元素值，请编辑元素文件。例如，假设您具有一个包含 5 个字符串元素的数组。ls 命令将按照如下所示列出这些元素：

```
0 1 2 3 4
```

以下命令将更改第三个元素的值：

```
echo -n "My new value" > 2
```

SA CLI 方法参数和返回值

本节将详细讨论方法上下文 （实例或静态）、参数用法、返回值和退出状态。

方法上下文与 self 参数

在 OGFS 中，一个方法驻留在多个位置。方法的位置与其上下文 （实例或静态）相关。

具有实例上下文的方法驻留在特定 SA 对象的 `method` 目录中。方法调用不需要 `self` 参数。方法的位置隐含了受该方法影响的对象实例。以下示例将更改 `d04.example.com` 服务器的客户：

```
cd /opsw/Server/@/d04.example.com/method
./setCustomer customer:i=9
```

具有静态上下文的方法驻留在 `/opsw/api` 下的某个位置。方法调用需使用 self 参数来确定受该方法影响的实例。在以下静态上下文示例中， `self:i` 指定了托管服务器的 ID：

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomer self:i=230054 customer:i=9
```

在命令行上传递参数

命令行参数将指定为“名称 - 值”对的形式，由等号 (=) 连接。“名称 - 值”对由一个或多个空白字符（通常是空格）分隔。命令行上的名称与 SA API 中对应 Java 方法的参数名称相匹配。

例如，在 SA API 中， `setCustomField` 方法具有以下定义：

```
public void setCustomField(CustomFieldReference self,
   java.lang.String fieldName, java.lang.String strValue)...
```

以下 SA CLI 方法示例将为 ID 为 3670039 的服务器的自定义字段指定值：

```
cd /opsw/api/com/opsware/server/ServerService/method
./setCustomField self:i=3670039 \
fieldName="Service Agreement" strValue="Gold"
```

如前一节中所述，具有实例上下文的方法无需使用 `self` 参数。以下 `setCustomField` 示例与前面的示例等效：

```
cd /opsw/.Server.ID/3670039
./setCustomField \
fieldName="Service Agreement" strValue="Gold"
```

可以按照任意顺序指定命令行参数。以下两个 SA CLI 方法调用等效：

```
./setCustomField fieldName="My Stuff" strValue="abc"
./setCustomField strValue="abc" fieldName="My Stuff"
```

要为参数指定 Null 值，请忽略该参数，或在等号后插入一个空格。在以下示例中， myParam 的值为 Null：

```
./someMethod myField="more info" myParam= anotherParam=9834
./someMethod myField="more info"          anotherParam=9834
```

指定参数的类型

如果某个方法具有抽象类型的参数，则必须指定具体类型和值。在以下示例中，必须指定 com.opsware.folder.FolderRef 类型：

```
cd /opsw/api/com/opsware/folder/FolderService/method
./remove self:i="com.opsware.folder.FolderRef:730555"
```

如果不指定具体类型，将显示以下错误消息：

```
Object type type-name is abstract.Specify a concrete sub-type.
```

复杂对象和数组作为参数

要传递作为复杂对象的参数，请将该对象的特性括在大括号中，如结构格式说明符语法（第 30 页）中所示。

以下示例将创建公共服务器组 AllMine。create 方法具一个参数 pattern，该参数的特性 parent 及 shortName 包含在大括号中。在该示例中， getPublicRoot 返回 2340555，即顶层公共组的 ID。

```
cd /opsw/api/com/opsware/device/DeviceGroupService/method
./.getPublicRoot:i ; echo
./create "pattern={ parent:i=2340555 shortName='AllMine' }"
```

通过重复参数名称 （每个数组元素一次），指定数组参数。例如，以下 assign 方法调用将指定名为 policies 的数组参数中的前两个元素：

```
cd /opsw/api/com/opsware/swmgmt
cd SoftwarePolicyService/method
./attachPolicies self:i=4220039 \
policies:i=4400335 policies:i=4400942
```

重载方法

如果多个同类方法具有相同的名称但不同的参数列表，则 Java 方法名称被重载。在重载 SA CLI 方法中，命令行上的参数名称表示要调用的方法。例如， setCustomField 方法被重载，以支持设置不同的数据类型。下面两个命令将调用不同版本的方法：

```
./setCustomField \
fieldName="Service Agreement" strValue="Gold"
./setCustomField \
fieldName=hmp longValue=2245
```

返回值

如果位于 SA CLI 方法下的 API 方法返回一个值，则 SA CLI 方法会将该值输出到 stdout。与 Unix 命令一样，您可以将方法的 stdout 重定向到文件，或将其分配给环境变量。

要更改返回值的表示形式，请在方法名称中插入一个前导句点并附加一个格式说明符。以下示例返回 ID 而不是默认名称形式的服务器引用：

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i
```

如果所指示的格式说明符与方法的返回类型不兼容，文件系统将发出错误消息作为响应。

退出状态

与 Unix shell 命令类似， SA CLI 方法使用退出状态 ($?) 表示调用结果。退出状态为零值表示成功，非零值表示错误。 SA CLI 方法将错误消息输出到 stderr。

表 4　SA CLI 方法的退出状态代码

退出状态	类别	描述
0	成功	该方法已成功完成。
1	命令行解析错误	用于方法调用的命令行格式错误，无法解析为一组选项 (--option[=value]) 和参数值 (param=value)。
2	参数解析错误	参数值无法解析为 API 所需的对象类型。
3	API 用法错误	由于用法错误 （如无效的参数值），导致调用失败。
4	访问错误	用户无权执行操作。
5	其他错误	发生除退出状态 1-4 所表示的错误之外的其他错误。

例如，以下 bash 脚本将检查 getDeviceGroups 方法的退出状态：

```
#!/bin/bash

cd /opsw/Server/@/toro.snv1.corp.example.com/method
./getDeviceGroups
cmnd_exit_status=$?

if [ $cmnd_exit_status -eq 0 ]
then
  echo "The command was successful."
else
```

```
  echo "The command failed."
  echo "Exit status = "  $cmnd_exit_status
fi
```

SA CLI 方法将调用基础 API 方法。如果 API 方法引发异常，SA CLI 方法将返回非零的退出状态。在调试方法调用时，查看关于已引发异常的信息将有助于解决问题。 OGFS 中的 /sys/last-exception 文件包含最近的 API 调用所引发的异常的堆栈跟踪。在阅读此文件后，系统将丢弃该文件的内容。

搜索筛选器和 SA CLI 方法

SA API 中的许多方法可接受对象引用作为参数。要根据搜索条件检索对象引用，可以调用 findServerRefs 和 findJobRefs 等方法。例如，您可以调用 findServerRefs 以搜索 hostname 特性中包含 example.com 的所有服务器。

搜索语法

findServerRefs 等方法具有以下语法：

find*object*Refs filter='[*object-type*:]*expression*'

filter 参数包含一个表达式，用于指定搜索条件。可以使用圆括号或大括号将表达式括起来。简单表达式具有以下语法：

value-object.attribute operator value

（该语法已简化。有关完整的定义，请参见筛选器语法 （第 157 页）。）

搜索示例

大多数 SA 对象类型具有关联的查找工具方法。本节介绍其中一小部分查找工具方法的用法。要了解如何在其他 SA CLI 方法中使用搜索功能，请参见示例脚本 （第 43 页）。

查找服务器

查找主机名包含 example.com 的服务器：

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i \
filter=' device:{ ServerVO.hostname CONTAINS example.com }'
```

查找使用特性值为 UNKNOWN 或 PRODUCTION 的服务器：

```
cd /opsw/api/com/opsware/server/ServerService/method
./.findServerRefs:i \
filter=' { ServerVO.use IN  "UNKNOWN"  "PRODUCTION" }'
```

以下 bash 脚本显示如何搜索服务器、在临时文件中保存服务器 ID，然后将每个 ID 指定为另一个方法调用的参数。该脚本显示了每个 Linux 服务器所属的公共组。

```
#!/bin/bash

TMPFILE=/tmp/server-list.txt
rm -f $TMPFILE

cd /opsw/api/com/opsware/server/ServerService/method

./.findServerRefs:i \
filter='{ ServerVO.osVersion CONTAINS Linux }' > $TMPFILE

for ID in `cat "$TMPFILE"`
do
  echo Server ID: $ID
  ./getDeviceGroups self:i=$ID
  echo
done
```

查找作业

本节中的示例将返回服务器审核或策略修正等作业的 ID。

查找已成功完成的作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i filter='job:{ job_status = "SUCCESS" }'
```

（有关 job_status 的允许值的列表，请参见 《SA 集成指南》中的 “作业批准集成”。）

查找已成功完成或发生警告的作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ job_status IN "SUCCESS" "WARNING" }'
```

查找今天已开始的作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ JobInfoVO.startDate IS_TODAY "" }'
```

查找所有服务器审核作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./findJobRefs \
filter='job:{ JobInfoVO.description = "Server Audit" }'
```

查找已在 ID 为 280039 的服务器上运行的作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i filter='job:{ job_device_id = "280039" }'
```

查找今天已失败的作业：

```
cd /opsw/api/com/opsware/job/JobService/method
./.findJobRefs:i \
filter='job:{ (( JobInfoVO.startDate IS_TODAY "" ) \
& ( job_status = "FAILURE" )) }'
```

查找其他对象

本节包含一些用于搜索软件策略和包的示例。

查找由 SA 用户 jdoe 创建的软件策略：

```
cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method
./.findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.createdBy CONTAINS jdoe }'
```

查找用于 Windows 2003 平台的含有 ismtool 的 MSI：

```
cd /opsw/api/com/opsware/pkg/UnitService/method
./.findUnitRefs:i \
filter='software_unit:{ ((UnitVO.unitType = "MSI") \
& ( UnitVO.name contains "ismtool" ) \
& ( software_platform_name = "Windows 2003" )) }'
```

查找名为 117170-01 的 Solaris 修补程序：

```
cd /opsw/api/com/opsware/pkg/solaris/SolPatchService/method
./.findSolPatchRefs:i filter='{name = 117170-01}'
```

查找名称包含字符串 Test、父文件夹名为 My Stuff 的文件夹。

```
cd /opsw/api/com/opsware/folder/FolderService/method
./.findFolders:s \
filter='( ( FolderVO.name CONTAINS "Test" ) \
& (folder_parent_name = "My Stuff" ) )'
```

可搜索的特性和有效运算符

并不是值对象的每个特性都可以在搜索筛选器中指定。例如，您可以搜索 ServerVO.use 但不能搜索 ServerVO.OsFlavor。

要找出给定对象类型的可搜索的特性，请调用 getSearchableAttributes 方法。以下示例列出了可在搜索表达式中指定的 ServerVO 的特性：

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributes searchableType=device
```

searchableType 参数用于指示对象类型。要确定 searchableType 的允许值，请输入以下命令：

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableTypes
```

要找出特性的有效运算符，请调用 getSearchableAttributeOperators 方法。以下示例列出了 ServerVO.hostname 特性的有效运算符 （如 CONTAINS 和 IN）：

```
cd /opsw/api/com/opsware/search/SearchService/method
./getSearchableAttributeOperators searchableType=device \
searchableAttribute=ServerVO.hostname
```

示例脚本

本节包含用于调用各种 SA CLI 方法的简单 bash 脚本的代码列表。这些脚本演示了如何在命令行上传递方法参数，包括复杂对象和 self 参数。如果确定要复制和粘贴这些示例脚本，您需要更改一些硬编码对象的名称，例如 d04.example.com 服务器。有关如何在 OGFS 中创建和运行脚本的教程说明，请参见第 28 页中的步骤 10。

remediate_policy.sh 第 46 页脚本将创建一个软件策略、向该策略添加包，并通过调用 startFullRemediateNow 方法在托管服务器上安装该包。

create_custom_field.sh

该脚本将创建一个名为 TestFieldA 的自定义字段 （虚拟列）、将该字段附加到所有服务器，然后在一个服务器上设置该字段的值。只有在附加之后，自定义字段才会出现在 SA 客户端中。可以为服务器、设备组或软件策略创建自定义字段。要创建自定义字段，您的 SA 用户必须属于具有 “管理虚拟列”权限的用户组。

与自定义特性不同，自定义字段将应用于某个类型的所有实例。有关在 OGFS 中创建自定义特性的示例，请参见 《SA 用户指南：Server Automation》中的 “管理自定义特性”。

create_custom_field.sh 脚本中包含以下代码：

```
#!/bin/bash
# create_custom_field.sh

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

# Create a virtual column.
# Remember the name because you cannot search for the
# displayName.
./create vo='{ name=TestFieldA type=SHORT_STRING \
displayName="Test Field A" }'

column_id=`./.findVirtualColumn:i name=TestFieldA`

echo --- column_id = $column_id

cd /opsw/api/com/opsware/server/ServerService/method

# Attach the column to all servers.
# All servers will have this custom field.
./attachVirtualColumn virtualColumn:i=$column_id

# Get the ID of the server named d04.example.com
devices_id=`./.findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }'`

echo --- devices_id = $devices_id
```

```
# Set the value of the custom field (virtual column) for
# a specific server.
./setCustomField self:i=$devices_id fieldName=TestFieldA \
strValue="This is something."
```

create_device_group.sh

该脚本将创建一个静态设备组，并向组中添加服务器。然后，该脚本将创建一个动态组、设置该组的规则，并刷新该组的成员资格。该脚本的最后一条语句将列出属于该动态组的设备。

下面是该脚本的代码：

```
#!/bin/bash
# create_device_group.sh

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Get the ID of the public root group (top of hierarchy).
public_root=`./.getPublicRoot:i`

# Create a public static group.
./create "vo={ parent:i=$public_root shortName='Test Group A' }"

# Get the ID of the group just created.
group_id=`./.findDeviceGroupRefs:i \
filter='{ DeviceGroupVO.shortName = "Test Group A" }' `

echo --- group_id = $group_id

cd /opsw/api/com/opsware/server/ServerService/method

# Get the ID of the server named d04.example.com
devices_id=`./.findServerRefs:i \
filter=\
'device:{ ServerVO.hostname CONTAINS "d04.example.com" }'`

echo --- devices_id = $devices_id

cd /opsw/api/com/opsware/device/DeviceGroupService/method

# Add a server to the device group.
./addDevices \
self:i=$group_id devices:i=$devices_id

# Create a dynamic device group.
./create \
"vo={ parent:i=$public_root \
shortName='Test Dyn B' dynamic=true }"

# Get the ID of the device group.
dynamic_group_id=`./.findDeviceGroupRefs:i \
filter='{ DeviceGroupVO.shortName = "Test Dyn B" }' `

echo --- dynamic_group_id = $dynamic_group_id
```

```
# Set the rule so that this group contains servers with
# hostnames containing the string example.com.
# The rule parameter has the same syntax as the filter
# parameter of the find methods.
./setDynamicRule self:i=$dynamic_group_id \
rule='device:{ ServerVO.hostname CONTAINS example.com }'

# By default, membership in dynamic device groups is refreshed
# once
# an hour, so force the refresh now.
./refreshMembership selves:i=$dynamic_group_id now=true

# Display the names of the devices that belong to the group.
echo --- Devices in group:
./getDevices selves:i=$dynamic_group_id
```

create_folder.sh

该脚本将创建名为 /Test 1 的文件夹、列出根 (/) 文件夹下的文件夹，然后创建子文件夹 /Test 1/Test 2。创建这些文件夹后，您可以在 SA 客户端的导航窗格的 “库” 中查看它们。

下面是该脚本的代码：

```
#!/bin/bash
# create_folder.sh

cd /opsw/api/com/opsware/folder/FolderService/method

# Get the ID of the root (top) folder.
root_id=`./.getRoot:i`

# Create a new folder under the root folder.
./create vo="{ name='Test 1' folder:i=$root_id }"

# Display the names of the folders under the root folder.
./getChildren self:i=$root_id

# Get the ID of the folder "/Test 1"
folder_id=`./.getFolderRef:i path="Test 1"`

# Create a subfolder.
./create vo="{ name='Test 2' folder:i=$folder_id }"

# Get the ID of the folder "/Test 1/Test 2"
folder_id=`./.getFolderRef:i path="Test 1" path="Test 2"`
echo folder_id = $folder_id
```

remediate_policy.sh

该脚本将在现有文件夹 Test 2 中创建名为 TestPolicyA 的软件策略、向该策略添加包含 ismtool 的包、将该策略附加到一个（而不是一组）服务器，然后修正该服务器。修正操作会启动在服务器上安装包的作业。可以在 SA 客户端中检查作业的进度和结果。有关使用 SA CLI 方法搜索作业的示例，请参见查找作业第 41 页。

在该脚本中，SoftwarePolicyService 的 create 方法中的 platforms 参数值为硬编码。在大部分示例脚本中，已通过按名称搜索对象来避免硬编码。对于平台，很难按 name 特性执行搜索，因为该特性不同于 displayName 特性，它显示在 SA 客户端中，但是无法搜索。用于查找平台 ID 的最简单方法是，转到 Twister 并运行不带任何参数的 PlatformService.findPlatformRefs 方法。

该脚本中的 update 方法将对 softwarePolicyItems （如果软件数据库包含很多名称相似的包，则很难按名称搜索到这个对象）的 ID 进行硬编码。一种用于获取 ID 的方法是，运行 SA 客户端，按字段 （例如 “文件名” 和 “操作系统”）搜索 “软件”，打开通过搜索找到的包，并记录包的属性视图中的 SA ID。

在下面的列表中，update 方法中存在错误的换行符。如果要复制此代码，请编辑脚本，以便 vo 参数位于一行中。

下面是 remediate_policy.sh 脚本的源代码：

```
#!/bin/bash
# remediate_policy.sh

# Get the ID of the folder where the policy will reside.
cd /opsw/api/com/opsware/folder/FolderService/method
folder_id=`./.findFolders:i filter='{ FolderVO.name = "Test 2" }'`

cd /opsw/api/com/opsware/swmgmt/SoftwarePolicyService/method

# Create a software policy named TestPolicyA.
# This policy resides in the folder located in the preceding findFolders
# call.
# The platform for this policy is Windows 2008 (ID 160076)
./create vo="{ platforms:i=160076 name="TestPolicyA" \
folder:i=$folder_id lifecycle=AVAILABLE }"

policy_id=`./.findSoftwarePolicyRefs:i \
filter='{ SoftwarePolicyVO.name = "TestPolicyA" }'`

echo --- policy_id = $policy_id

# Call the update method to add a package to the software policy.
# The package ID for the "ismtool" msi installer is 4010001.
# Note that "force = true" is required.
./update self:i=$policy_id force=true \
vo='{ softwarePolicyItems:i=com.opsware.pkg.windows.MSIRef:4010001 }'

cd /opsw/api/com/opsware/server/ServerService/method

# Get the ID of the server named d04.opsware.com
```

```
devices_id=`./.findServerRefs:i \
filter='device:{ ServerVO.hostname CONTAINS "d04.opsware.com" }'`

echo --- devices_id = $devices_id

# Attach the policy to a single server (not a group).
./attachPolicies self:i=$devices_id \
policies:i=$policy_id

# Remediate the server to install the package in the policy.
job_id=`./.startFullRemediateNow:i self:i=$devices_id`

echo --- job_id = $job_id
```

remove_custom_field.sh

虽然在操作环境中不会经常执行自定义字段删除操作，但是在测试环境中有时必须执行该操作。请注意，必须首先取消附加自定义字段，然后才能将其删除。

下面是 remove_custom_field.sh 的代码：

```
#!/bin/bash
# remove_custom_field.sh

if [ !-n "$1" ]
   then
   echo "Usage:`basename $0` <name>"
   echo "Example:`basename $0` hmp"
   exit
fi

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

column_id=`./.findVirtualColumn:i name=$1`

echo --- column_id = $column_id

cd /opsw/api/com/opsware/server/ServerService/method

# Column must be detached before it can be removed.
./detachVirtualColumn virtualColumn:i=$column_id

cd /opsw/api/com/opsware/custattr/VirtualColumnService/method

# Remove the virtual column.
./remove self:i=$column_id
```

schedule_audit_task.sh

该脚本将启动审核任务，计划在将来某个日期运行。在采用 SA CLI 方法时，将使用以下语法指定日期参数：

```
YYYY/MM/DD HH:MM:SS.sss
```

用于启动任务的 `startAudit` 方法将返回执行审核的作业的 ID。有关使用 SA CLI 方法搜索作业的示例，请参见查找作业第 41 页。

下面是 `schedule_audit_task.sh` 的代码：

```
#!/bin/bash
# schedule_audit_task.sh

cd /opsw/api/com/opsware/compliance/sco/AuditTaskService/method

# Get the ID of the audit task to schedule.
audit_task_id=`./.findAuditTask:i \
filter='audit_task:{ (( AuditTaskVO.name BEGINS_WITH "HW check" ) \
& ( AuditTaskVO.createdBy = "gsmith" )) }'`

echo --- audit_task_id = $audit_task_id

# Schedule the audit task for Oct. 16, 2013.
# In the startDate parameter, note that the last delimiter for the time
# is a period, not a colon.
job_id=`./startAudit self:i=$audit_task_id \
schedule:s='{ startDate="2013/10/16 00:00:00.000" }' \
notification:s='{ onFailureOwner="sjones@opsware.com" \
onFailureRecipients="jdoe@opsware.com" \
onSuccessOwner="sjones@opsware.com" \
onSuccessRecipients="jdoe@opsware.com" }'`

echo --- job_id = $job_id
```

获取 SA CLI 方法的用法信息

在将来的发布版中，SA CLI 方法将显示用法信息。在此之前，您可以使用下列各节所述的方法从 API 文档或 OGFS 中获取必需的信息。

列出服务

SA API 方法已组织到一些服务中。要找出可用于 SA CLI 方法的服务，请在 OGSH 会话中输入以下命令：

```
cd /opsw/api/com/opsware
find .-name "*Service"
```

要列出 API 文档中的服务，请在浏览器中指定以下 URL：

```
https://occ_host:1032
```

其中 *occ_host* 是运行命令中心组件的核心服务器的 IP 地址或主机名。

查找 API 文档中的服务

OGFS 中的服务路径将映射到 API 文档中的 Java 包名称。例如，在 OGFS 中，`ServerService` 方法显示在以下目录中：

```
/opsw/api/com/opsware/server
```

在 API 文档中，以下接口定义了这些方法：

```
com.opsware.server.ServerService
```

列出服务的方法

在 OGFS 中，您可以列出某个服务的 `method` 目录的内容。例如，要显示 `ServerService` 的方法名称，请输入以下命令：

```
ls /opsw/api/com/opsware/server/ServerService/method
```

在 API 文档中，执行以下步骤查看 `ServerService` 的方法：

1. 在左上部的窗格中，选择 `com.opsware.server`。
2. 在左下部的窗格中，选择 `ServerService`。
3. 在主窗格中，向下滚动查看方法。

列出方法的参数

在 API 文档中，执行上一节中所述的步骤。在服务接口页面的方法详细信息部分中，查看参数和返回类型。（有关方法参数的详细信息，请参见在命令行上传递参数 （第 37 页）。）

获取关于值对象的信息

API 文档中介绍到，一些服务方法会传递或返回值对象 (VO)，其中包含数据成员 （特性）。例如，`ServerService.getServerVO` 方法返回 `ServerVO` 对象。要找出 `ServerVO` 所包含的特性，请执行以下步骤：

1. 在 API 文档中，选择 `ServerVO` 链接。可以在多个位置找到该链接：
   - `getServerVO` 的方法签名
   - `com.opsware.server` 的类列表 （左下部窗格）
   - 在索引页上。 API 文档的主窗格顶部提供了一个指向索引页的链接。

2 在 ServerVO 页面上，记录 getter 和 setter 方法。每个 getter-setter 对与值对象中的一个特性相对应。例如， getCustomer 和 setCustomer 指示 ServerVO 包含名为 customer 的特性。

确定特性是否可修改

只有少数对象特性可由客户端应用程序修改。要找出可修改的特性，请执行以下步骤：

1 在 API 文档中，转到值对象页面，如上一节中所述。

2 在 setter 方法的方法详细信息部分中，查找 “Field can be set by clients”。

对于 OGFS 中表示的 SA 对象 （如服务器和客户），可通过检查 attr 目录中文件的访问类型，确定可修改的特性。具有读写 (rw) 访问类型的文件对应于可修改的特性。例如，要列出某个服务器的可修改特性，请输入以下命令：

```
cd /opsw/Server/@/server-name/attr
ls -l | grep rw
```

确定某个特性是否可用于筛选器查询中

要确定是否可以将值对象的某个特性用于筛选器查询 （搜索）中，请执行以下步骤：

1 在 API 文档中，转到值对象页面。

2 在对应于该特性的 getter 方法的方法详细信息部分中，查找字符串 “Field can be used in a filter query”。

从 OGSH 会话中，要确定是否可以搜索某个特性，请使用可搜索的特性和有效运算符 （第 42 页）中所述的方法。

3 通过 Pytwist 执行 Python API 访问

Pytwist 概述

Pytwist 是一组 Python 库，允许托管服务器和自定义扩展访问 SA API。（twist 是 Web 服务数据访问引擎的内部名称。）对于托管服务器，您可以设置将通过 Pytwist 调用 SA API 的 Python 脚本，以使最终用户可以将这些脚本作为 DSE 或 ISM 控件调用。自定义扩展是由 HP SA Professional Services 创建的 Python 脚本，可在命令引擎 (Way) 中运行。 Pytwist 允许自定义扩展访问 SA 数据模型中最近添加的内容 （例如文件夹和软件策略），这些内容无法从命令引擎脚本进行访问。

本章的目标读者包括熟悉 SA 数据模型、自定义扩展、代理和 Python 编程语言的开发人员和顾问。

Pytwist 安装

在尝试执行本章中的示例之前，请确保您的环境满足下列各节中详细说明的安装要求。

Pytwist 支持的平台

Pytwist 在托管服务器和核心服务器上受支持。有关这些服务器支持的操作系统的列表，请参见《SA Release Notes》。

Pytwist 依赖于 SA 代理和自定义扩展所使用的 Python 2.4 版。

与 Web 服务和 Java RMI 客户端不同，Pytwist 客户端依赖于内部 SA 库。如果您的客户端程序需要从非托管或核心的服务器访问 SA API，请使用 Web 服务或 Java RMI 客户端而不是 Pytwist。

Pytwist 的访问要求

Pytwist 需要访问正在运行 Web 服务数据访问引擎的核心服务器的端口 1032。默认情况下，引擎在端口 1032 上侦听。

在托管服务器上安装 Pytwist

在 SA 安装或升级过程中， Pytwist 库被放置在具有命令引擎组件的核心服务器上。因此，无需安装 Pytwist 即可在自定义扩展中使用它。

但是，代理安装中不包括 Pytwist。您需要通过修正包含 Pytwist ZIP 文件的策略，在托管服务器上安装 Pytwist。在 SA 客户端中， Pytwist ZIP 文件位于以下文件夹中：

```
/Opsware/Tools/Python Opsware API Access
```

此文件夹还包括预建的软件策略，其中包含适用于每种平台的 Pytwist ZIP 文件。例如，名为 Windows Python SA API Access 的策略包含适用于 Windows XP、2000、2003 等的 ZIP 文件。在修正该策略时，只会安装与平台版本匹配的 ZIP 文件。例如，如果您修正了 Windows 2003 服务器上的策略，则只会安装用于 Windows 2003 的 ZIP 文件。

要在托管服务器上安装 Pytwist，请执行以下步骤：

1 在 SA 客户端中的 “设备” 下，找到托管服务器。

2 在 “内容” 窗格中打开托管服务器。

3 在 “托管服务器” 窗口中，从 “操作” 菜单选择 “安装软件”。

4 在 “安装软件” 窗口中，选择软件策略，例如 Windows Python SA API Access。

5 单击 “安装”。

6 执行 “修正” 向导中的步骤，一直到系统显示 “摘要检查” 窗口。

7 单击 “启动作业”。

Pytwist 示例

本节中的 Python 代码示例显示了如何从托管服务器获取信息、创建文件夹以及修正软件策略。每个 Pytwist 示例将执行以下操作：

1 导入包。

在导入 SA API 命名空间的对象（如 `Filter`）时，路径中将包含 Java 包名称，且该名称前将附加 `pytwist`。下面是 `get_server_info.py` 的 `import` 语句示例：

```
import sys
from pytwist import *
from pytwist.com.opsware.search import Filter
```

2 创建 `TwistServer` 对象：

```
ts = twistserver.TwistServer()
```

有关方法参数的信息，请参见 TwistServer 方法语法 （第 57 页）。

3 获取对服务的引用。

服务的 Python 包名称与 Java 包名称相同，但是没有前导 `opsware.com`。例如， Java `com.opsware.server.ServerService` 包映射到 Pytwist `server.ServerService`：

```
serverservice = ts.server.ServerService
```

4 调用服务的 SA API 方法：

```
filter = Filter()
. . .
servers = serverservice.findServerRefs(filter)
. . .
for server in servers:
  vo = serverservice.getServerVO(server)
. . .
```

get_server_info.py

该脚本将搜索主机名包含命令行参数的所有托管服务器。搜索方法 findServerRefs 将返回一组对服务器永久对象的引用。对于每一个引用， getServerVO 方法将返回值对象 (VO)，它是用于保存服务器特性的数据表示形式。下面是 get_server_info.py 脚本的代码：

```
#!/opt/opsware/bin/python
# get_server_info.py

# Search for servers by partial hostname.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *
from pytwist.com.opsware.search import Filter

# Check for the command-line argument.
if len(sys.argv) < 2:
  print 'You must specify part of the hostname as the search target.'
  print "Example:" + sys.argv[0] + "   " + "opsware.com"
  sys.exit(2)

# Construct a search filter.
filter = Filter()
filter.expression = 'device_hostname *=* "%s"' % (sys.argv[1])

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to ServerService.
serverservice = ts.server.ServerService

# Perform the search, returning a tuple of references.
servers = serverservice.findServerRefs(filter)

if len(servers) < 1:
  print "No matching servers found"
  sys.exit(3)

# For each server found, get the server's value object (VO)
# and print some of the VO's attributes.
for server in servers:
   vo = serverservice.getServerVO(server)
   print "Name:" + vo.name
   print "   Management IP:" + vo.managementIP
```

```
   print "   OS Version: " + vo.osVersion
```

create_folder.py

该脚本将通过调用 createPath 方法创建名为 /TestA/TestB 的文件夹。请注意，createPath 的 path 参数不包含斜杠。 path 中的每个字符串元素表示文件夹中的一个层级。然后，该脚本将检索并打印根文件夹的所有下级文件夹的名称。 create_folder.py 脚本的列表如下所示：

```
#!/opt/opsware/bin/python
# create_folder.py

# Create a folder in SA.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get a reference to FolderService.
folderservice = ts.folder.FolderService

# Get a reference to the root folder.
rootfolder = folderservice.getRoot()
# Construct the path of the new folder.
path = 'TestA', 'TestB'

# Create the folder /TestA/TestB relative to the root.
folderservice.createPath(rootfolder, path)

# Get the child folders of the root folder.
rootchildren = folderservice.getChildren(rootfolder,
'com.opsware.folder.FolderRef')

# Print the names of the child folders.
for child in rootchildren:
   vo = folderservice.getFolderVO(child)
   print vo.name
```

remediate_policy.py

该脚本将创建一个软件策略，并将其附加到服务器，然后修正该策略。脚本中的服务器名称是硬编码的：平台、服务器和父文件夹。您也可以在命令行中指定策略名称。如果要多次运行脚本，则该操作很方便。软件策略的平台必须与策略中所含的包的操作系统匹配。因此，如果更改硬编码的平台名称，则还需更改 unitfilter.expression 中的名称。

下面的列表中存在多个错误的换行符。如果要复制此代码，请首先修复这些错误的换行符，然后再运行代码。以“NOTE”开头的注释行指出了错误的换行符。

```
#!/opt/opsware/bin/python
# remediate_policy.py

# Create, attach, and remediate a software policy.

import sys
sys.path.append("/opt/opsware/pylibs")
from pytwist import *
from pytwist.com.opsware.search import Filter
from pytwist.com.opsware.swmgmt import SoftwarePolicyVO

# Initialize the names used by this script.
foldername = 'TestB'
platformname = 'Windows 2003'
servername = 'd04.example.com'
# If a command-line argument is specified,
# use it as the policy name
if len(sys.argv) == 2:
  policyname = sys.argv[1]
else:
  policyname = 'TestPolicyA'

# Create a TwistServer object.
ts = twistserver.TwistServer()

# Get the references to the services used by this script.
folderservice = ts.folder.FolderService
swpolicyservice = ts.swmgmt.SoftwarePolicyService
serverservice = ts.server.ServerService
unitservice = ts.pkg.UnitService
platformservice = ts.device.PlatformService

# Search for the folder that will contain the policy.
folderfilter = Filter()
folderfilter.expression = 'FolderVO.name = ' + foldername
folderrefs = folderservice.findFolderRefs(folderfilter)

if len(folderrefs) == 1:
  parent = folderrefs[0]
elif len(folderrefs) < 1:
  print "No matching folders found."
  sys.exit(2)
else:
  print "Non-unique folder name:" + foldername
  sys.exit(3)

# Search for the reference to the platform "Windows Server 2003."
platformfilter = Filter()
platformfilter.objectType = 'platform'
doublequote = '\"'
# Because the platform name contains spaces,
```

```
# it's enclosed in double quotes
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
platformfilter.expression = 'platform_name = ' + doublequote + platformname +
doublequote
platformrefs = platformservice.findPlatformRefs(platformfilter)

if len(platformrefs) == 0:
  print "No matching platforms found."
  sys.exit(4)

# Search for the references to some software packages.
unitfilter = Filter()
unitfilter.objectType = 'software_unit'
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
unitfilter.expression = '((UnitVO.unitType = "MSI") & ( UnitVO.name contains
"ismtool" ) & ( software_platform_name = "Windows 2003" ))'
unitrefs = unitservice.findUnitRefs(unitfilter)

# Create a value object for the new software policy.
vo = SoftwarePolicyVO()
vo.name = policyname
vo.folder = parent
vo.platforms = platformrefs
vo.softwarePolicyItems = unitrefs

# Create the software policy.
swpolicyvo = swpolicyservice.create(vo)

# Search by hostname for the reference to a managed server.
serverfilter = Filter()
serverfilter.objectType = 'server'
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
serverfilter.expression = 'ServerVO.hostname = ' + servername
serverrefs = serverservice.findServerRefs(serverfilter)

if len(serverrefs) == 0:
  print "No matching servers found."
  sys.exit(5)

# Create an array that has a reference to the
# newly created policy.
swpolicyrefs = [1]
swpolicyrefs[0] = swpolicyvo.ref

# Attach the software policy to the server.
swpolicyservice.attachToPolicies(swpolicyrefs, serverrefs)

# Remediate the policy and the server.
# NOTE: The following code line has a bad line break.
# The assignment statement should be on a single line.
jobref = swpolicyservice.startRemediateNow(swpolicyrefs, serverrefs)
print 'The remediation job ID is %d' % jobref.id
```

Pytwist 详细信息

本节描述了特定于 Pytwist 的行为和语法。

身份验证模式

Pytwist 客户端的身份验证模式很重要，因为它会影响客户端可以访问的 SA 功能及资源。Pytwist 客户端可在以下模式中运行：

- **已验证**：客户端已在 `TwistServer` 对象上调用 `authenticate(username, password)` 方法。在调用 `authenticate` 方法后，客户端被授予 `username` 参数所指定的 SA 用户权限，这与登录到 SA 客户端的最终用户非常相似。
- **未验证**：客户端未调用 `TwistServer.authenticate` 方法。在托管服务器上，会将客户端验证为类似代理证书控制设备的身份。在自定义扩展中使用时，未验证的 Pytwist 客户端需要访问命令引擎证书。有关自定义扩展和证书的详细信息，请联系技术支持代表。

TwistServer 方法语法

`TwistServer` 方法用于配置从客户端到 Web 服务数据访问引擎的连接。（有关示例调用，请参见 Pytwist 示例 （第 52 页）。） `TwistServer` 的所有参数都是可选的。表 5 设置参数的默认值。

表 5　TwistServer 方法的参数

参数	描述	默认值
`host`	要连接到的主机名。	twist
`port`	要连接到的端口号。	1032
`secure`	是否对连接使用 https 协议。允许的值：1 (true) 或 0 (false)。	1
`ctx`	连接的 SSL 上下文。	无。（另请参见身份验证模式（第 57 页）。）

当创建 `TwistServer` 对象时，客户端不会建立与服务器的连接。因此，如果发生连接问题，直到客户端调用 `authenticate` 或 SA API 方法时才会遇到此问题。

错误处理

如果 `TwistServer.authenticate` 方法或 SA API 方法遇到问题，会引发 Python 异常。可以在 `except` 子句中捕获这些异常，如下例所示：

```
# Create the TwistServerobject.
ts = twistserver.TwistServer('localhost')
```

```
# Authenticate by passing an SA user name and password.
try:
  ts.authenticate('jdoe', 'secretpass')
except:
  print "Authentication failed."
  sys.exit(2)
```

将 Java 包名称和数据类型映射到 Pytwist

Pytwist 接口适用于 Python，但 SA API 是使用 Java 语言编写的。由于这两种编程语言之间的差异， Pytwist 客户端必须采用本节所述的映射规则。

在 SA API 文档中，Java 包名称以 `com.opsware` 开头。当在 Pytwist 中指定包名称时，请在开头处插入 `pytwist`，例如：

```
from pytwist.com.opsware.compliance.sco import *
```

SA API 文档指定方法参数并返回 Java 数据类型的值。表 6 显示对于 Pytwist 中的 API 方法调用如何从 Java 数据类型映射到 Python。

表 6　从 Java 数据类型映射到 Python

SA API 中的 Java 数据类型	pytwist 中的 Python 数据类型
`Boolean`	整数 1 表示 true，整数 0 表示 false。
`Object[]` （对象数组）	作为 API 方法调用的输入参数，对象数组可以是 Python 元组或数组。作为 API 方法调用的输出，对象数组将以 Python 元组形式返回。
Map	字典
List	数组
`Date`	`long` 数据类型，表示自新纪元（1970 年 1 月 1 日午夜）开始所经过的毫秒数。

4 创建自动化平台扩展 (APX)

本章描述了如何创建和管理自动化平台扩展 (APX)，通常简称为*扩展*。APX 提供了一个框架，允许任何熟悉基于脚本的编程工具（如 Shell 脚本、Python、Perl 和 PHP）的用户扩展 SA 功能以及创建与 SA 紧密集成的应用程序。 SA 提供两种类型的 APX：

- **程序 APX**（又称为**脚本 APX**）在全局文件系统 (OGFS) 中运行，并且可以使用所有 OGFS 功能。您可以使用典型的编程方法来利用 SA API 以及访问核心的托管服务器，以实现新的自定义功能。例如，可以编写一个 APX，用于从托管服务器收集 BIOS 信息并使用 shell 命令填充自定义字段。请参见程序 APX （第 61 页）。
- **Web APX** 可用于创建基于 Web 的应用程序，在其中使用 GET 或 POST URL 调用 Apache 2.x 进程或 CGI/PHP 脚本。 Web APX 可包含静态 Web 资源 （如图像），并且可以使用 CGI 或 PHP 进行动态内容生成。请参见Web APX （第 61 页）。

APX 允许您访问托管环境的数据，以及与 Web 应用程序、脚本、程序和其他应用程序共享并处理该数据。下面是 APX 的一些优势：

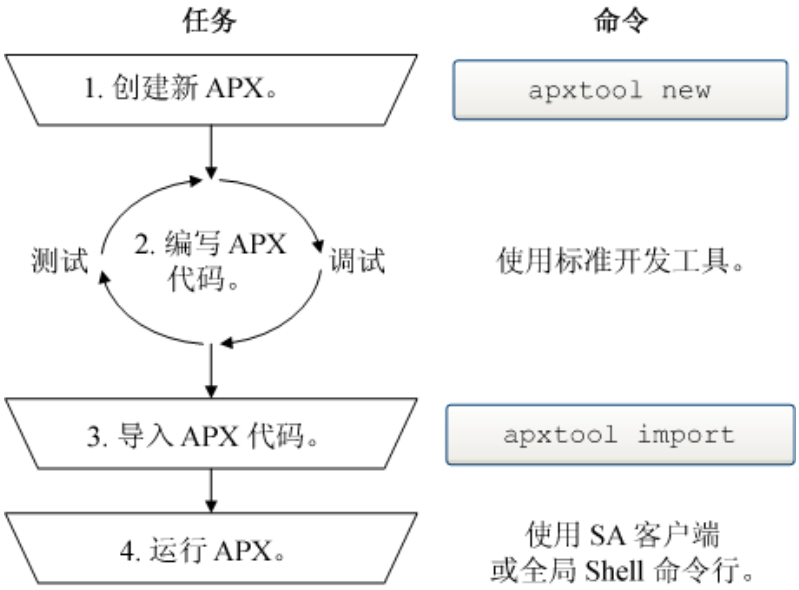
- 列在 SA 库中，可从 SA 客户端使用。
- 通过版本控制进行唯一标识和管理。
- 安全，因为它们可充分利用 SA 的安全模型。当需要时， APX 可以在 APX 会话中安全地将用户权限临时升级到正常默认值之上。
- 可在 SA 核心内部和 SA 核心之间进行扩展。
- 可以计划将它们自动推送到服务器。
- 可审核。
- 可通过 SA 平台升级一直保持有效。在升级后，无需重新编写 APX。

有关使用 APX 扩展的信息，请参见《SA 用户指南：Server Automation》中的“运行 SA 扩展”。由于您也可以从 SA 全局 Shell 中运行 APX 扩展，所以还可参见《SA 用户指南：Server Automation》中的 “SA 全局 Shell”。

创建 APX

下图显示了用于创建 APX 及要使用的对应命令的基本步骤。有关如何创建 Web APX 的教程，请参见教程：创建 Web 应用程序 APX （第 81 页）。有关如何创建程序 APX 的教程，请参见教程：创建程序 APX （第 87 页）。

图 3　创建 APX



1. 要创建新 APX，请使用 `apxtool new` 命令。此命令将创建一组模板文件。可以编辑这些文件来创建自己的 APX。

   可以选择使用 `apxtool new` 注册新 APX。注册 APX 时将在 SA 中保留 APX 的名称。如果不在此步骤中注册 APX，可以在下面的步骤 3 中使用 `apxtool import` 命令进行注册。

   请参见apxtool 命令 （第 68 页）。

2. 在创建 APX 模板文件后开发 APX 代码，方法是修改 `apxtool new` 命令所创建的模板文件并根据需要添加自己的文件。可以对 APX 代码进行测试，确保它能够正常运行。

3. 测试 APX 代码之后，必须使用 `apxtool import` 命令将其导入 SA。

4. 从 SA 客户端或全局 Shell 命令行中运行 APX。

   — 从 SA 客户端：选择 “库” ➤ “按类型” 选项卡 ➤ “扩展” ➤ “程序”。选择 APX。选择 “操作” ➤ “运行” 菜单。

   — 从全局 Shell 命令行：选择 “工具” ➤ “全局 Shell” 菜单，从 SA 客户端打开全局 Shell。通过输入以下命令来运行 APX
   `/opsw/apx/bin/<APX name>`。

   — 有关详细信息，请参见 《SA 用户指南：Server Automation》中的 “运行 SA 扩展” ，以及 《SA 用户指南：Server Automation》中的 “SA 全局 Sheell”。

要创建用于在 VMware ESXi 服务器上运行的 APX 扩展，该 APX 扩展必须使用其 Web 服务接口与 ESXi 服务器进行远程通信。有关 VMware ESXi 服务器的详细信息，请参见《SA 用户指南: Server Automation》中的 “虚拟服务器管理”。

程序 APX

程序 APX 也称为脚本 APX，类似于 shell 命令，可作为 OGFS 服务器脚本执行。可以从 OGFS 命令行调用它们，并使用 STDIN 或命令行参数向其传递输入参数。它们的输出将传递到 STDOUT 和 STDERR 中。

程序 APX 在全局 Shell (OGSH) 会话内执行，并且可以访问对调用该 APX 的用户可用的所有 OGSH 功能。这些功能包括 `rosh`、CLI、OGFS 等等。您可以使用任何基于脚本的工具（如 shell 脚本、 Python 和 Perl 等）编写程序 APX。

可以从 OGSH 命令提示符处调用程序 APX。通常情况下，将以同步方式执行程序 APX，这意味着直到程序 APX 返回时， shell 提示符才会返回。无论是在 twister 还是 OGFS 中，都不能将 APX 计划为重复作业。

程序 APX 位于 OGFS 目录 `/opsw/apx/bin` 中。

在交互式 OGSH 会话中，用户只能在 `/opsw/apx/bin` 中查看其有权执行的程序 APX。如果用户尝试调用其无权执行的程序 APX，将导致从 shell 返回 “文件未找到”错误。

程序 APX 也可以由其他 Web APX 或程序 APX 调用。例如， Web APX 中的 CGI 程序或 PHP 脚本可以调用程序 APX。

Web APX

Web APX 使用 CGI 程序或 PHP 脚本执行。这些 CGI 程序和 PHP 脚本在特定于用户的 OGSH 会话中执行。它们可以访问各种 SA 设施，例如 `rosh`、SA API、CLI 或可在 OGSH 会话中执行的任何命令。 Web APX 由已启用 PHP 模块的内置 Apache Web 服务器提供服务。

可以通过两种方式访问 Web APX: 使用诸如 Internet Explorer 或 Firefox 的独立 Web 浏览器，或从 SA 客户端。不支持 Microsoft ActiveX。

首次从独立 Web 浏览器调用 Web APX 时将触发一个登录对话框，其中要求对 SA 用户凭据执行验证。从 SA 客户端调用 Web APX 时不需要额外登录。Web APX 可以用于构建自定义客户应用程序的用户界面。

要在已启用 “增强的安全配置”的 Windows Server 2003、 2008 和 2012 上使用 Microsoft Internet Explorer 6 和 7 启动 APX，必须首先将 SA Web 客户端 URL 添加到 Internet Explorer 的可信站点列表中。

APX 用户角色

APX 用户有三种常规角色，如表 7 中所示：

表 7　APX 用户角色

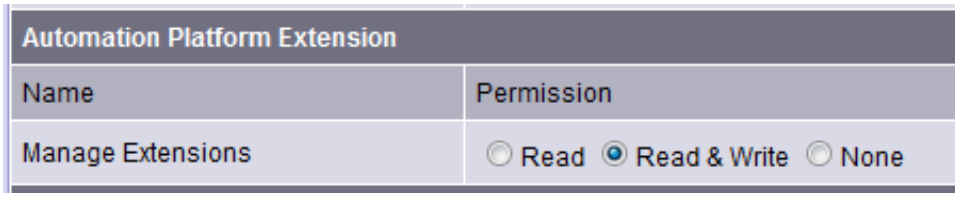
用户角色	描述
最终用户	运行 APX。该类用户通常无权修改 APX 或查看其内容。
APX 开发人员	创建和发布 APX。该类用户可以导入和导出 APX，并且可以修改 APX 内容。
APX 管理员	确定允许运行的 APX 用户。这些用户将通过管理文件夹权限来分配可执行权限以运行 APX。APX 管理员可能无权修改 APX 本身，但有权查看 APX 内容以确定要允许执行的 APX。

APX 权限

APX 需要您拥有 SA 客户端功能权限**管理扩展**。可以为用户组分配以下权限之一：

- 管理扩展：读取
- 管理扩展：读取和写入
- 管理扩展：无

图 4　APX 功能权限



这些功能权限只适用于 APX 开发人员和管理员，而不适用于只需要运行 APX 的用户。

- **Read** 权限将授予显示 APX 源内容或导出 （下载） APX 源存档的能力。
- **Read & Write** 权限除了授予读取访问权限之外，还将授予修改 APX 内容的能力。
- **None** 权限将拒绝对 APX 源的所有访问。

除了 SA 客户端功能权限**管理扩展**之外，必须使用文件夹权限 （列出、读取、写入、执行）确定用户有权访问的 APX。

表 8　APX 权限

权限	描述
列出	列出系统的 APX 的权限。
读取	查看 APX 内容的权限。
写入	修改 APX 内容及导入和导出 APX 的权限。
执行	运行 APX 和查看 APX 属性的权限。

表 9 显示了如何根据 “管理扩展”功能权限和文件夹权限的组合来确定权限的列表。

表 9　APX 权限列表

	管理扩展权限：		
文件夹权限：	**读取**	**读取和写入**	**无**
列出	列出 APX	列出 APX	列出 APX
读取	导出 APX	导出 APX	列出 APX
写入	导出 APX	导入和导出 APX	列出 APX
执行	运行 APX	运行 APX	运行 APX

与其他 SA 功能类似，您可以向用户授予访问 APX 的权限，以及指定用户可向其应用 APX 的托管服务器和 / 或策略。

如果用户尝试访问其无权执行的 Web APX， Web 浏览器会收到返回代码 `HTTP 403 Forbidden`。

有关 SA 权限的详细信息，请参见 《SA 管理指南》。

权限升级

在执行 APX 时，用户仅有权访问已在 SA 中授权的资源和操作。然而在某些情况下，在执行 APX 时，有必要临时向用户授予*升级的权限*，即超出 SA 权限的权限。您可以在运行 APX 的同时，明确地向用户临时授予高于其默认 SA 权限的特定权限。权限升级对于运行 APX 的用户而言是透明的。

例如，您可能希望某个用户能够在托管服务器上运行 BIOS 信息收集应用程序，但该用户没有执行该操作的授权。您可以为没有 BIOS 信息收集应用程序运行权限的用户编写一个 APX，来向该用户临时授予所需的权限。在 APX 结束其运行后，该用户的权限将恢复为默认值。

权限升级可在 `apx.perm` 文件中指定。有关详细信息，请参见 APX 权限升级配置文件 - apx.perm（第 78 页）。

APX 结构

APX 具有以下特性：

- APX 类型：程序 APX （也称为脚本 APX）或 Web APX。
- APX 的唯一名称：这是 APX 必须唯一具有的完整名称。例如， com.hp.sa.RestartMyApp。
- APX 显示名称：该名称通常比 APX 的唯一名称短。例如， RestartMyApp。
- APX 版本：可以通过设置版本字符串来维护多个 APX 版本，也可以让 SA 自动管理版本。 APX 版本可以是一个简单数字，如版本 1、 2、 3 等，也可以是任何字母数字字符串。

  有关详细信息，请参见将 APX 导入 SA - apxtool import （第 72 页）和设置 APX 的当前版本 - apxtool setCurrent （第 75 页）。

文件结构

对于 SA，APX 就是一组符合 APX 类型（程序 APX 或 Web APX）协定的文件和目录，允许 APX 运行时任务可以正确地执行它。例如， Web APX 可能需要 `index.html` 文件或 `index.php` 文件。程序 APX 可能需要使用与该 APX 具有相同名称的 shell 命令。

有关 APX 中的文件的详细信息，请参见 APX 文件 （第 77 页）。

OGFS 集成

在 SA 文件系统中，APX 基础结构依赖于 OGFS 管理用户会话及显示 APX 的各个部分。下列各节描述了 APX 是如何集成到 OGFS 及其各应用程序中的。

APX 可执行文件目录

程序 APX 将在全局 Shell (OGSH) 中以可执行程序形式进行处理。这些 APX 在 OGSH 中作为可执行的命令显示。这使得 shell 用户能够像运行 shell 命令一样调用 APX。

APX 可执行文件目录具有以下格式：

```
/opsw/apx/bin/{apx_name}
```

其中 `apx_name` 是 APX 的名称。在 `/opsw/apx/bin/{apx_name}` 中运行 `apx_name` 将调用 `apx_name` 的当前版本。

APX 运行时目录

APX 运行时目录由 APX 运行时用来支持 APX 执行。APX 运行时目录必须具有 APX 源的访问权限。此外，拥有 APX 开发人员权限和读取权限的用户也可以访问 APX。对于全局 Shell 中的非 APX 开发人员， APX 运行时目录将不可用。

APX 运行时目录将引用当前 APX 版本的源。它具有以下格式：

/opsw/apx/runtime/{apx_type}/{apx_name}

其中 apx_type 可以是 script 或 web。

APX 接口 - 定义 APX 扩展的类别

APX 接口允许您创建命名的 APX 类别，以及查找给定类别的所有 APX。接口是类别的名称。例如，您可以创建一类 APX，全部接受特定的输入参数集，并生成特定类型的输出数据。也可以创建一类 APX，全部执行特定的操作集。

此外，还可以创建 APX 或外部应用程序，用于获取所需类别的所有 APX 的名称并执行这些 APX。或者， APX 或应用程序可以仅显示所需类别的 APX 的列表，而由用户选择要执行的 APX。

APX 接口是一个名称，用于定义 APX 调用者与 APX 之间的非正式协定。

- **定义接口名称**的 APX 将创建一类具有该名称的 APX。
- **实现接口**的 APX 会将其自身声明为属于该类别的 APX。

示例接口

SA 提供了一个名为 RightClickToRun 的接口。该接口定义了一类 APX，这类 APX 接受一个或多个设备作为输入参数，并针对这些设备运行。此外， SA 客户端将在 “操作” ➤ “运行扩展” 菜单中显示所有实现此接口的 APX，以允许用户选择一个或多个设备，并对选定设备运行这些 APX。有关此接口的详细信息，请参见 RightClickToRun 接口 （第 67 页）。

定义接口

APX 接口定义了一类 APX 的名称。所有实现该接口的 APX 将属于该类别，并且必须符合该接口的约定。要创建新类别，您可以让 APX “定义” 接口。

要让 APX 定义接口，请执行以下步骤。

1. 使用 apxtool new 命令创建 APX。有关此命令的详细信息，请参见创建新 APX - apxtool new （第 69 页）。
2. 找到新 APX 的文件，并在文本编辑器中打开名为 interfaces 的文件。该接口文件位于 APX 目录的 APX-INF 目录中。
3. 在 interfaces 文件的末尾添加对应于以下项的三行：
   - 文件中的接口部分的名称。这是接口的唯一名称。
   - 接口的显示名称。
   - 接口描述。

   例如，下面显示了名为 “com.hp.sa.MyNewInterface” 的接口的接口部分名称、显示名称和描述：

```
[com.hp.sa.MyNewInterface]
name=MyNewInterface
description=”This is a simple interface for testing purposes.”
```

4 保存更改并关闭文件。

5 使用 apxtool import 命令将修改后的 APX 导入 SA。有关此命令的详细信息，请参见将 APX 导入 SA - apxtool import （第 72 页）。

要升级现有 APX 以定义接口，必须创建 interfaces 文件，并如上所述添加接口。

实现接口

APX 接口指定了一类须符合该接口约定的 APX。要将 APX 指定为属于某个类别，可以让 APX“实现”接口。要让 APX 实现接口，请执行以下步骤。

1 使用 apxtool new 命令创建 APX。有关此命令的详细信息，请参见创建新 APX - apxtool new （第 69 页）。

2 找到新 APX 的文件，并在文本编辑器中打开名为 apx.cfg 的文件。

3 在 apx.cfg 文件中找到关于“实现”的部分。该部分简要介绍了如何指定 APX 所实现的接口。

4 在 apx.cfg 文件中找到以下行：

```
[Implementing]
interfaces=
```

5 修改 interfaces= 行，并在该行结尾处添加接口名称。例如，如果 APX 实现了名为“com.hp.sa.MyNewInterface”的接口，则 apx.cfg 文件应包含以下行：

```
[Implementing]
interfaces=com.hp.sa.MyNewInterface
```

要实现多个接口，请将它们添加到接口行，各接口之间用冒号分隔，如下所示：

```
[Implementing]
interfaces=com.hp.sa.MyNewInterface:com.hp.sa.AnotherInterface
```

6 保存更改并关闭 apx.cfg 文件。

7 使用 apxtool import 命令将修改后的 APX 导入 SA。有关此命令的详细信息，请参见将 APX 导入 SA - apxtool import （第 72 页）。

必须设置 APX 的当前版本，才能在通过 SA 客户端或 apxtool query 命令查看 APX 时看到已实现的接口。有关详细信息，请参见设置 APX 的当前版本 - apxtool setCurrent （第 75 页）。

要升级现有 APX 以使用接口，必须按如上所述将接口添加到现有 apx.cfg 文件中。

RightClickToRun 接口

SA 提供了一个可在您的 APX 中使用的接口，名为 `com.hp.client.server.RightClickToRun`。该接口仅适用于程序 APX，不适用于 Web APX。当希望 APX 执行以下所有操作时，请使用该接口：

- 接受一个或多个设备作为 APX 的输入参数。实现此接口的 APX 必须接受“`-d device id`”作为输入参数。
- 出现在 “操作”➤“运行扩展”➤“选择扩展 ...” 窗口中。
- 出现在 SA 客户端的 “操作”➤“运行扩展”菜单中。当已使用 “操作”➤“运行扩展”➤“选择扩展 ...” 菜单运行一次 APX 之后， APX 将出现在该菜单中。

要从 “操作”➤“运行扩展”菜单中执行 APX，用户必须具有执行该 APX 的权限。用户无权执行的任何 APX 都不会出现在该菜单项中。有关权限的信息，请参见 《SA 管理指南》。

RightClickToRun 接口允许用户在 SA 客户端中选择一个或多个设备，并针对这些设备运行 APX。

当选择 “操作”➤“运行扩展”菜单项时， SA 客户端显示实现 com.hp.client.server.RightClickToRun 接口的所有 APX 程序。当选择 APX 时，将针对所有选定服务器运行。将为每个选定的服务器调用一次 APX。

有关如何让 APX 实现此接口的说明，请参见实现接口 （第 66 页）。有关如何使用实现此接口的 APX 的详细信息，请参见 《SA 用户指南：Server Automation》中的 “运行 SA 扩展”。

使用接口 API

您可以使用 SA API 将自己的应用程序与 SA 和 APX 集成。您的应用程序可以使用 SA API 中 com.opsware.apx 包内的 APXInterfaceService 接口，确定将实现该特定接口的所有 APX。有关使用 SA API 的信息，请参见第 21 页上的第 1 章 API 文档和 Twister。

apxtool 命令

可以在 OGFS 会话中使用 apxtool 命令创建和管理 APX。apxtool 命令可在全局 Shell 中使用，位于 /opsw/bin/apxtool 目录中。

有关如何使用 apxtool 创建 Web APX 的教程，请参见教程：创建 Web 应用程序 APX （第 81 页）。

apxtool 的语法

可在 OGFS 命令行中按照如下所示调用该 APX 工具：

```
apxtool [-h | --help] {function} arguments
```

要获取该 APX 工具支持的命令和参数的完整列表，请从 OGSH 命令行中运行不带参数的 apxtool。

APX 工具支持以下主要函数：

表 10　APX 工具函数

函数	用法
new	在 OGFS 中创建新的 APX 源目录和一组新模板文件。可以选择在 SA 中注册 APX。通过注册，会分配一个 APX ID，并使您的 APX 名称对使用 SA 的其他人（需具有相应权限）可用。请参见创建新 APX - apxtool new （第 69 页）。
import	将 APX 文件导入 SA 库，并创建 APX 的新版本。可以选择在 SA 中注册 APX。通过注册，会分配一个 APX ID，并使您的 APX 名称对使用 SA 的其他人（需具有相应权限）可用。请参见将 APX 导入 SA - apxtool import （第 72 页）。
setcurrent	设置 SA 库中 APX 的当前版本。SA 中可存在多个版本的 APX，但只能执行当前版本。请参见设置 APX 的当前版本 - apxtool setCurrent （第 75 页）。
query	显示关于 APX 的信息。请参见查询 APX 信息 - apxtool query （第 73 页）。
export	将所有 APX 文件从 SA 库复制到一组单独的文件。
delete	从 SA 库中删除 APX。

使用短命令选项和长命令选项

apxtool 命令的大多数选项接受短格式或长格式。

- 短格式包括一个连字符和一个字符，例如 “-t” 和 “-v”。
- 长格式包括两个连字符后跟一个单词，例如 “--type” 和 “--view”。

有些选项需要后跟一个参数。例如，“-t webapp” 和 “-t details”。可以按照四种格式之一指定参数，这些格式是等效的。为了进行说明，下列这些命令等效并会生成相同的结果：

```
apxtool query -t webapp
```

```
apxtool query -twebapp
apxtool query -tw
apxtool query --type webapp
apxtool query --type=webapp
```

有些选项只需要键入足以标识选项参数的最少字符数。例如，在查询函数中， `--view` 选项需要“`list`”、“`details`”、“`versions`”参数。下列命令将生成相同的结果：

```
apxtool query --view=details
apxtool query --view=d
apxtool query -vdetails
apxtool query -vd
```

创建新 APX - apxtool new

可以使用该 APX 工具创建新 APX，并选择在 SA 中注册该 APX 的名称。此命令将创建一组可供您修改的 APX 模板文件。有关组成 APX 的文件的信息，请参见 APX 文件 （第 77 页）。

用法

```
apxtool new [options] {src_dir}
```

其中 `src_dir` 参数用于指定要在其中创建新 APX 的模板文件的目录。如果省略该参数，会将模板文件放置到当前目录中。

表 11 列出了可用于创建新 APX 的选项：

表 11　apxtool new 的选项

选项	用法
`-h, --help`	显示此帮助消息并退出。
`-t <type>` `--type=<type>`	（**必需**）APX 类型。有效值包括：`script` 或 `webapp`。例如， **-ts** 表示脚本 APX， **-tw** 表示 Web APX。（脚本 APX 也称为程序 APX）。
`-u <unique name>` `--uniquename=<unique name>`	（**必需**） APX 的唯一名称。唯一名称是符合文件系统格式的点分隔名称。必须至少包含一个点。有效字符包括：`[a-zA-Z0-9_.]`。 **例如：** `com.hp.sa.security.scan_ports`

表 11　apxtool new 的选项 （续）

选项	用法
-n <name> --name=<name>	（**可选**）APX 在文件夹中的显示名称。如果未指定名称，但是指定了唯一名称，则会将 APX 唯一名称的结尾部分用作显示名称。请注意，该名称必须在指定的文件夹中唯一。 例如，如果唯一名称为 com.hp.sa.MyWebExt，则默认显示名称为 MyWebExt。
-d <description> --description=<description>	（**必需**）对 APX 的简短描述。如果描述是带有扩展名 .txt 文件名，则会假定该文件是一个文本文件，其内容将用作 APX 描述。
-r --register	（**可选**）向系统中注册 APX 的名称。如果指定此选项，则还必须指定 -f 或 --folder。 如果没有在 apxtool new 中指定 **-r** 和 **-f**，则必须在 apxtool import 中使用 **-f**。
-f <path> --folder=<path>	（**可选**）将在其中注册 APX 的 SA 文件夹路径。它可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。只有在使用了 -r 或 --register 时，才需要使用此选项。 如果没有在 apxtool new 中指定 **-r** 和 **-f**，则必须在 apxtool import 中使用 **-f**。
-Q, --quiet	（**可选**）禁止显示输出消息。
-F, --force	（**可选**）禁止显示确认提示。

删除 APX - apxtool delete

可以使用该 APX 工具从 SA 库中删除现有 APX。

用法

```
apxtool delete [options]
```

表 12 列出了可用于删除 APX 的选项：

表 12　apxtool delete 的选项

选项	用法
-h --help	显示此帮助消息并退出。
-t <type> --type=<type>	（**必需**）APX 类型。有效值包括：script 或 webapp。例如， -ts 表示脚本。
--id=<APX id>	（**可选**）所需 APX 的对象标识符。
-u <unique_name> --uniquename=<unique_name>	（**可选**） APX 的唯一名称。唯一名称是符合文件系统格式的点分隔名称。必须至少包含一个点。有效字符包括：[a-zA-Z0-9_.]。 **例如：** com.hp.sa.security.scan_ports
-n <name>, --name=<name>	（**可选**） APX 在文件夹中的显示名称。
-f <path>, --folder=<path>	（**可选**） SA 文件夹路径。 path 可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。
-Q, --quiet	（**可选**）禁止显示输出消息。
-F, --force	（**可选**）禁止显示确认提示。

从 SA 导出 APX - apxtool export

可以使用该 APX 工具导出 APX。导出操作会下载 APX 源存档文件的特定版本，并将这些文件放到一个目录或 .zip 存档文件中。

用法

```
apxtool export [options] {target_dir}
```

其中参数 target_dir 是要将 APX 源存档文件复制到的目录，或要将 APX 源存档内容扩展到的目录，具体取决于是否指定了 --archive 选项。如果省略该参数，将使用当前目录。

表 13 列出了可用于导出 APX 的选项。

表 13　apxtool export 的选项

选项	用法
`-h, --help`	显示此帮助消息并退出。
`-t <type>, --type=<type>`	（**必需**）APX 类型。有效值包括：`script` 或 `webapp`。例如， `-ts` 表示脚本。
`--id=<APX id>`	（**可选**）所需 APX 的对象标识符。
`-u <unique_name>, --uniquename=<unique_name>`	（**可选**） APX 的唯一名称。唯一名称是符合文件系统格式的点分隔名称。必须至少包含一个点。有效字符包括：`[a-zA-Z0-9_.]`。 **例如：** `com.hp.sa.security.scan_ports`
`-n <name>, --name=<name>`	（**可选**） APX 在文件夹中的显示名称。
`-f <path>, --folder=<path>`	（**可选**） SA 文件夹路径。 `path` 可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。
`-v v<ersion_string>, --version=<version_string>`	（**可选**）此选项指定要下载的 APX 版本。如果省略该参数，将下载当前版本。
`-a, --archive`	如果已指定该选项，会将 APX 原始源存档中的 APX 源以 ZIP 或 JAR 文件形式导出。
`-Q, --quiet`	（**可选**）禁止显示输出消息。
`-F, --force`	（**可选**）禁止显示确认提示。

将 APX 导入 SA - apxtool import

可以使用该 APX 工具导入 APX。导入操作将发布 APX 的新版本，并且可以选择将该版本设置为当前版本。如果 APX 尚未注册，则该命令还会注册 APX。

只能运行 APX 的当前版本。如果不设置当前版本， APX 将不可运行。可以使用 `apxtool import` 或 `apxtool setcurrent` 设置当前版本。请参见设置 APX 的当前版本 - apxtool setCurrent（第 75 页）。

用法

```
apxtool import [options] {apx_src}
```

其中 apx_src 可以是扩展名为 .zip 或 .jar 的存档 APX 源文件，也可以是要发布的 APX 文件所在目录的名称。apx_src 可以是相对或绝对路径。如果省略该参数，将使用当前目录。所指定的目录或存档文件必须包含目录 APX-INF。

表 14 列出了可在导入 APX 时使用的选项：

表 14　apxtool import 的选项

选项	用法
-h, --help	显示此帮助消息并退出。
-c, --setcurrent	如果已指定该选项，会将新发布的版本设置为 APX 的当前版本。
--version=<version_string>	该 APX 的新版本。如果已经在 apx.cfg 中指定了 version_string，则不能使用此选项。如果未指定版本，系统会自动分配一个版本。
-f <path>, --folder=<path>	（**可选**）SA 文件夹路径。path 可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。 如果没有在 apxtool new 中指定 **-r** 和 **-f**，则必须在 apxtool import 中使用 **-r**。
-Q, --quiet	（**可选**）禁止显示输出消息。
-F, --force	（**可选**）禁止显示确认提示。

查询 APX 信息 - apxtool query

可以使用该 APX 工具获取和查看 APX 信息。可以指定其他选项来限制所得到的 APX。多个相同选项将构成逻辑 OR 表达式。如果没有找到匹配的结果，该命令将返回退出代码 100。

用法

```
apxtool query [options]
```

表 15 列出了可在查询 APX 信息时使用的选项：

表 15　apxtool query 的选项

选项	用法
-h, --help	显示此帮助消息并退出。
-v <view>, --view=<view>	（**可选**）选择查询结果的预定义视图之一。选项包括 list（默认值）、 details 和 versions。 -v list 是 APX 基本信息的单行显示，以表格形式呈现。 -v details 是 APX 信息的多行显示。 -v versions 将列出所有 APX 版本。您仅需为视图类型指定足够的字符；例如，-vd 与 -v details 是相同的。如果已选择 versions 布局，则查询必须得到单个 APX 对象。
-t <type>, --type=<type>	（**可选**）指定要显示的 APX 类型。有效值包括：script、 webapp 或 interface。默认情况下将显示所有类型。 -t script 将显示所有脚本 APX。 -t webapp 将显示所有 Web APX。 -t interface 将显示已定义一个或多个接口的所有 APX。 例如， apxtool query -ts 将显示所有脚本 APX。
--id=<APX id>	（**可选**）所需 APX 的对象标识符。
-u <unique_name> --uniquename=<unique_name>	（**可选**） APX 的唯一名称。唯一名称是符合文件系统格式的点分隔名称。必须至少包含一个点。有效字符包括：[a-zA-Z0-9_.]。 **例如：** com.hp.sa.security.scan_ports
-n <name>, --name=<name>	（**可选**） APX 在文件夹中的显示名称。
-f <path>, --folder=<path>	（**可选**） SA 文件夹路径。 path 可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。
--current	（**可选**）如果已指定该选项，则仅查询已设置当前版本的 APX 对象。

表 15　apxtool query 的选项（续）

选项	用法
`--format=<format_string>`	（**可选**）此高级选项允许您指定 APX 列表的自定义显示格式。 `format_string` 是包含嵌入式标记名称的字符串，在显示时这些嵌入式标记名称将替换为相应的值。标记名称的格式必须是 `%(tag_name)`。 可使用格式字符串“`__show_tags__`”显示所有支持的标记名称的列表。
`--csv`	（**可选**）以逗号分隔值格式显示输出。如果已指定 `--format` 选项，则将忽略该选项。
`-Q, --quiet`	（**可选**）禁止显示无关的输出消息。

设置 APX 的当前版本 - apxtool setCurrent

可以使用该 APX 工具将一个 APX 版本设置为当前版本。

只能运行 APX 的当前版本。如果不设置当前版本，APX 将不可运行。可以使用 `apxtool import` 或 `apxtool setcurrent` 设置当前版本。请参见将 APX 导入 SA - apxtool import（第 72 页）。

用法

```
apxtool setcurrent [options] {version_str}
```

其中 `version_str` 参数为必需，用于唯一标识 APX 的现有版本。

表 16 列出了可在设置 APX 版本时使用的选项：

表 16　apxtool setcurrent 的选项

选项	用法
`-h, --help`	显示此帮助消息并退出。
`-t <type>, --type=<type>`	（**必需**）APX 类型。有效值包括：`script`、`webapp`。例如，`-ts` 表示脚本。
`--id=<APX id>`	（**可选**）所需 APX 的对象标识符。

表 16　apxtool setcurrent 的选项（续）

选项	用法
-u <unique_name>, --uniquename=<unique_name>	（**可选**） APX 唯一名称。唯一名称是符合文件系统格式的点分隔名称，必须至少包含一个点。有效字符包括 [a-zA-Z0-9_.]。 **例如：** com.hp.sa.security.scan_ports
-n <name>, --name=<name>	（**可选**） APX 在文件夹中的显示名称。
-f <path>, --folder=<path>	（**可选**） SA 文件夹路径。 path 可以是完整路径、部分路径、绝对路径或相对路径，只要它可以唯一标识特定文件夹即可。
-Q, --quiet	（**可选**）禁止显示输出消息。
-F, --force	（**可选**）禁止显示确认提示。

错误处理

APX 工具命令符合标准 POSIX 约定。在成功时返回 0，在发生其他错误时返回非零值。该 APX 工具会将正常输出发送到 STDOUT，将错误和警告发送到 STDERR。当发生错误时，该 APX 工具通常会向 STDERR 返回一条描述性消息。

表 17 中显示了典型的错误情况分类：

表 17　APX 工具错误情况

返回代码	描述
0	成功
1	语法或用法错误
2	权限相关错误
3	用户取消操作
4	运行时错误

可能还存在其他一些未记录的退出代码。唯一可确保的是，如果退出代码为 0，则说明命令已成功完成其操作。

APX 文件

本节描述了在运行 apxtool new 命令时创建的模板文件。下表概述了这些文件。后面的各节详细描述了其中一些文件。

表 18　APX 文件

文件名	描述
apx.cfg	APX 配置文件，其中包含用于完整描述 APX 的元数据。请参见 APX 配置文件 - apx.cfg （第 77 页）。
apx.perm	APX 权限文件，用于指定权限升级规则。请参见 APX 权限升级配置文件 - apx.perm （第 78 页）。
description.txt	对 APX 的文本描述。在 apxtool new -d 选项中指定。请参见创建新 APX - apxtool new （第 69 页）。
interfaces	APX 接口定义文件。指定 APX 定义或实现的接口。请参见 APX 接口 - 定义 APX 扩展的类别 （第 65 页）。
usage.txt	对 APX 用法的文本描述。
run.sh	仅适用于程序 APX。该文件包含 APX 的可执行代码。该文件包含程序 APX 的功能。有关示例，请参见教程：创建程序 APX （第 87 页）。
index.php	仅适用于 Web APX。该文件包含 Web APX 的 PHP 源代码。该文件包含 Web APX 的功能。有关示例，请参见教程：创建 Web 应用程序 APX （第 81 页）。

APX 配置文件 - apx.cfg

无论类型如何，所有 APX 都必须具有配置文件 apx.cfg。apxtool new 命令将创建该文件的模板，供您修改。该文件包含用于完整描述 APX 的元数据。apx.cfg 使用“key=value”格式定义 APX 的属性。将使用行继续符 “\” 连接多行。

表 19 APX 配置文件特性描述了所有 APX 的通用特性。特定于 APX 类型的特性将在对应 APX 类型的功能规范中描述。请注意，某些特性可从 apx.cfg 配置文件中提取，并在 SA 中进行管理。对于可修改的特性 （例如描述），后续更新 apx.cfg 文件时将相应地更新 SA 管理的数据。

要查看 apx.cfg 文件的示例，请运行 apxtool new 命令，并打开该命令创建的文件。

表 19　APX 配置文件特性

特性	是否可修改?	描述
type	否	APX 的类型。必须是 webapp 或 script。（脚本 APX 也称为程序 APX）。 APX 类型一旦创建，即不能更改。
name	是	这是 APX 显示名称，可包含多字节字符。可以随时更改该名称。该名称将列在 SA 客户端 APX 文件夹中。
unique_name	否	APX 的唯一名称。该名称将用作显示在 OGFS 中的 APX 文件名。该名称与类型一起构成唯一标识 APX 的键。该名称一旦创建，即不能更改。由于该名称将在文件系统中使用，所以它必须符合文件系统命名规范。通常情况下，该名称应该是 ASCII。
version	是	表示 APX 当前版本的版本字符串。如果值以字符串“auto:”开头，则 SA 将使用针对每个新版本递增的整数自动管理版本。
description	是	对 APX 所执行操作的文本描述。也可以使用 description.txt 文件来代替该特性。
usage	是	对 APX 用法的文本描述。也可以使用 usage.txt 文件来代替该特性。
interfaces	是	APX 实现的一个或多个接口。使用冒号 (:) 字符分隔多个接口。
command	是	在调用时将由 APX 运行的可执行文件。

APX 权限升级配置文件 - apx.perm

可使用 apx.perm 文件指定权限升级规则。如果该文件不存在，或者不包含升级权限，将使用用户的默认权限运行 APX。

当使用 APX 工具的 **New** 命令创建新 APX 时，它会生成特定默认文件，包括默认的 apx.perm 文件，该文件默认情况下未定义升级权限。该默认文件包含一些注释掉的示例，APX 开发人员可以将这些示例用作模板。

可使用如下所述的三种方法指定升级。

- 无升级 （第 79 页）。
- 所有权限 （第 79 页）。
- 执行升级 （第 79 页）。

无升级

未指定升级特性。APX 运行时将使用当前用户权限执行 APX。如果 APX 调用了用户无权执行的特权操作， APX 将终止执行并显示错误。

所有权限

这是一个特殊权限，用于向用户临时授予所有操作权限。该权限仅用于开发或演示用途。对于需快速证明概念或执行演示的情况，这是非常有用的工具，因为用户无需关注细粒度的权限调整。而对于生产环境，这并不是一个好的选择，因为其安全性不高。

要授予所有权限，请通过使用通配符匹配所有功能的宏来编辑 `apx.perm` 文件。例如：

```
use_feature(name="*")
```

执行升级

在 `apx.perm` 文件中指定一组预定义的常用操作。执行 APX 时，APX 运行时临时将这些权限授予 APX。 SA 具有功能和资源权限的全面列表。为了简化用于升级相关功能的任务，用户可以使用通配符来匹配相关功能组。例如：

```
@use_feature(name="Application.*")
```

显示 APX 的进度

可以在程序 APX 中使用 `apxprogress` 命令提供 APX 进度信息。对于运行很长时间的程序 APX，如果您需要向用户提供该 APX 的进度状态，则这很有用。

可以使用 Web APX 作为程序 APX 的前端，并显示 Web APX 中的进度。

apxprogress 命令

可使用 `apxprogress` 命令定义程序 APX 执行过程中的步骤数，以及记录每个步骤的完成时间。这使得用户能够了解 APX 的进展情况。

apxprogress 的语法

```
apxprogress {option}...
```

表 20　apxprogress 命令的选项

选项	描述
-i <*total number of steps*>	指定 APX 运行的步骤总数。可以在 APX 开始时使用该选项指定 APX 将运行的步骤总数。 可以在 APX 中多次使用该选项以增加步骤数。每次使用之后，都会按指定的值递增步骤总数。
-c <*current step*>	指定当前步骤编号。在 APX 代码中的每个步骤完成后，可使用该选项调用 apxprogress。
-m <*message*>	指定用于描述 APX 状态的文本消息。
-a <*data*>	指定 APX 可提供的关于其自身的其他信息。
-d	表示调试模式。显示命令向 stdout 输出的内容，以供调试。
-h	显示关于 apxprogress 命令的帮助信息。

使用 apxprogress 的 Shell 脚本示例

以下 shell 脚本是使用 apxprogress 命令的程序 APX 的一部分。该 APX 总共定义了 100 个步骤，并将声明其当前进度 100 次。每次声明进度时，它还会提供一条包括步骤编号的消息。

```
#!/bin/sh
#############################################################################
# A simple shell script for a program APX that displays progress
# about itself.
# Author:<name>
#############################################################################
echo "This is a simple APX that uses apxprogress."

totalsteps=100
apxprogress -i $totalsteps -c 1

for i in `seq $totalsteps`; do
        apxprogress -c $i -m "APX is running, working on step $i"  -d
        sleep 10
done
```

查看 APX 进度

可以使用 SA API 方法 `JobService.getProgress()` 来访问已调用 `apxprogress` 命令、正在运行的 APX 的进度信息。有关该方法的示例，请参见 7. 在 Twister 界面中查看 APX 进度 （第 92 页）中的教程：创建程序 APX （第 87 页）。

教程：创建 Web 应用程序 APX

本教程演示了如何创建、发布和运行名为 `mywebapp` 的简单 Web 应用程序 APX。

运行本教程中所创建的 APX 默认版本将显示 PHP 命令 `phpinfo` 的输出。本教程后面将告诉您如何修改 PHP 代码，以便显示托管服务器的列表。因为本教程提供了源代码，所以无需预先掌握 PHP 知识。

按照顺序完成以下任务。

1. 设置权限并创建教程文件夹 （第 82 页）
2. 创建新 Web 应用程序 （第 82 页）
3. 向 SA 导入新 Web 应用程序 （第 84 页）
4. 运行新 Web 应用程序 （第 84 页）
5. 修改 Web 应用程序 （第 85 页）
6. 运行修改后的 Web 应用程序 （第 86 页）

教程先决条件

要完成本教程，您必须具有以下能力和环境：

- 可作为 `admin` 或其他**超级管理员**组成员登录 SA。以 `admin` 身份登录将允许您设置权限。
- 可作为属于**高级用户**组的用户登录 SA。

  高级用户有权创建和运行 Web 应用程序。在本教程所显示的示例命令中，该用户的名称是 `jdoe`。
- 了解如何在 SA Web 客户端中设置客户端功能权限。

  有关权限的详细信息，请参见 《SA 管理指南》的 “用户和组设置”一章。
- 了解如何在 SA 客户端中创建文件夹。

  有关文件夹的详细信息，请参见 《SA 用户指南：Server Automation》。
- 了解如何打开全局 Shell 会话。

  有关说明，请参见 《SA 用户指南：Server Automation》的 “全局 Shell”一章。
- 了解 `ls` 和 `cd` 等基本 UNIX 命令。
- 具有开发在 HTTP 服务器上运行的 Web 应用程序的经验。

1. 设置权限并创建教程文件夹

1 以 admin 身份登录 SA Web 客户端，并验证**高级用户**组是否具有以下权限：

– 管理扩展：读取和写入

可以在 SA Web 客户端上的 “客户端功能” 选项卡中找到该权限。

2 以**高级用户**组成员身份登录 SA 客户端，并在 SA 库中创建以下文件夹：

/Dev/MyApp

在本教程的后面，您将向 MyApp 文件夹上载 Web 应用程序。在非教程环境中，此文件夹的名称可以是任意名称。可以创建或选择任何其他文件夹来包含 Web 应用程序。

3 退出 SA 客户端。

4 以 admin 身份登录 SA 客户端，并打开 MyApp 文件夹的 “文件夹属性”。

5 在 “文件夹属性” 的 “权限” 选项卡中，确保**高级**用户组具有下列权限：

— 列出文件夹内容

— 在文件夹中读取对象

— 在文件夹中写入对象

— 在文件夹中执行对象

6 退出 SA 客户端。

2. 创建新 Web 应用程序

1 以属于**高级用户**组的 SA 用户身份打开全局 Shell 会话。

2 在核心的 OGFS 主目录中，创建一个名为 mywebapp 的目录，然后更改该目录：

```
$ mkdir mywebapp
$ cd mywebapp
```

Web 应用程序文件将存储在 mywebapp 目录中。

3 使用 apxtool new 命令创建 Web 应用程序的目录结构和默认文件，如下所示。

```
$ pwd
/home/jdoe/mywebapp
$ ls
$
$ apxtool new -tw  -d "This is my first app." \
-u com.hp.sa.jdoe.mywebapp
Create source directory /home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp? Y/N y
Info: Successfully created APX 'mywebapp' source directory: /home/jdoe/
mywebapp.
```

-tw 选项表示 APX 类型是 Web 应用程序， -d 指定描述信息， -u 指定应用程序的唯一名称。

有关 apxtool new 命令选项的详细信息，请参见联机帮助：$ apxtool new -h

4 将目录更改为 apxtool new 命令所创建的新目录，并列出目录中的文件。

```
$ pwd
/home/jdoe/mywebapp
$ cd com.hp.sa.jdoe.mywebapp
$ ls
APX-INF  cgi-bin  css  images  index.php
$ ls -R
.:
APX-INF  cgi-bin  css  images  index.php

./APX-INF:
apx.cfg  apx.perm  description.txt  interfaces  usage.txt

./cgi-bin:

./css:
hp_sa.css

./images:
```

5 显示默认 index.php 文件的内容：

```
$ cat index.php
<?php

// Show information about PHP
phpinfo();

?>
```

与其他 Web 应用程序一样，您可以用 index.html 文件替换 index.php 文件。不过该本教程中使用的是 index.php 文件，您将在下一节中对其进行修改。

6 检查 APX-INF 目录中的一些文件。有关详细信息，请参见 APX 文件 （第 77 页）。

APX-INF 目录包含特定于 APX Web 应用程序的信息。如以下 cat 命令所示， description.txt 文件中保存了使用 apxtool new 的 -d 选项指定的文本。

```
$ ls APX-INF/
description.txt  apx.cfg  apx.perm  usage.txt
$ cat APX-INF/description.txt
This is my first app $
```

以下 grep 命令显示了 APX 配置文件 apx.cfg 中的一些属性。type 和 uniquename 的值来自于 apxtool new 命令的 -t 和 -u 选项。有关 APX 配置文件的详细信息，请参见 APX 配置文件 - apx.cfg （第 77 页）。

```
$ grep "=" APX-INF/apx.cfg
type=webapp
name=mywebapp
unique_name=com.hp.sa.jdoe.mywebapp
```

3. 向 SA 导入新 Web 应用程序

导入 Web 应用程序时将执行以下操作：

- 在 SA 中的 HTTP 服务器上安装 Web 应用程序。
- 将 Web 应用程序复制到在 SA 库和全局 Shell 中显示的文件夹。
- 为 Web 应用程序分配版本号。

输入 apxtool import 命令并使用 y 来响应提示消息，如下所示。-f 选项指定 SA 库中将用于存储 Web 应用程序的文件夹。 -c 选项设置 Web 应用程序的当前版本。

```
$ pwd
/home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory:/home/jdoe/mywebapp/
com.hp.sa.jdoe.mywebapp?Y/N y
APX with unique name 'com.hp.sa.jdoe.mywebapp' does not exist.
Register it into the system?Y/N y
Info:Successfully registered APX 'mywebapp' (310001) in folder '/Dev/
MyApp'.
Info:Successfully published a new version '1' for APX 'mywebapp'.
Info:Successfully set APX 'mywebapp'(310001) current version as '1'.
```

4. 运行新 Web 应用程序

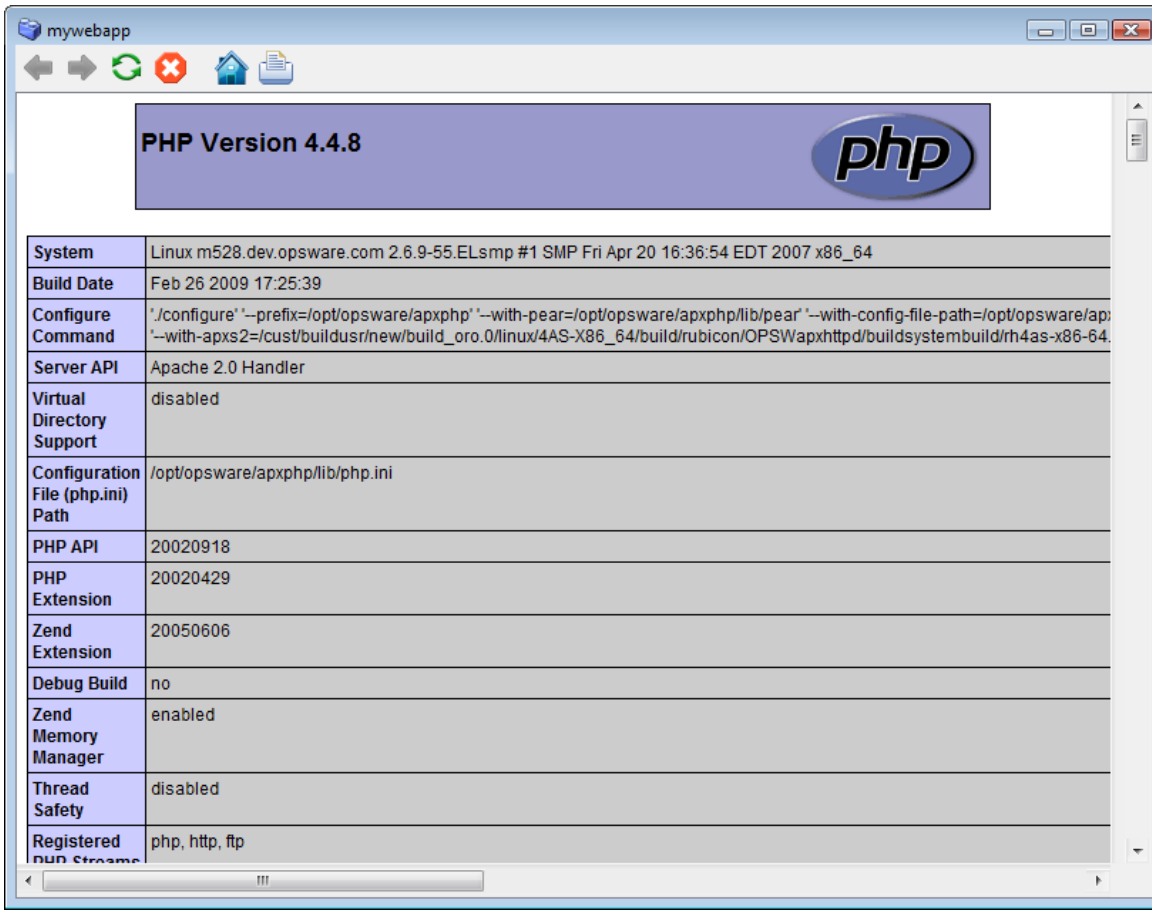
现在，您已发布了 Web 应用程序，可以像最终用户那样在 SA 客户端中运行该 Web 应用程序。

1. 以属于**高级用户**组的用户身份登录 SA 客户端。
2. 选择 “库”选项卡，然后选择 “按类型”选项卡。
3. 导航到 “扩展”➤“Web”节点，您将在此处看到 mywebapp 扩展。

   如果未看到 mywebapp，请确保您拥有 1. 设置权限并创建教程文件夹 （第 82 页）中所述的必需权限。
4. 要运行 Web 应用程序，请选择 mywebapp，并选择 “操作”➤“运行”菜单。

   此时将显示图 5 中所示的窗口。Web 应用程序将显示由 index.php 文件的 phpinfo 语句所生成的信息。

图 5　Web 应用程序版本 1



5. 修改 Web 应用程序

运行默认 `index.php` 文件是用来检查开发环境的一个好方法，但是它无法充分利用 SA 功能。在本节中，将修改 `index.php` 文件，使其列出 SA 管理的服务器的名称。

1　在全局 Shell 会话中，找到 Web 应用程序的 `index.php` 文件。

```
$ cd /home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp
$ ls
APX-INF  cgi-bin  css  images  index.php
```

2　在文本编辑器（如 `vi`）中打开 `index.php` 文件。

3　使用以下几行替换 `index.php` 的内容：

```
<html>
<head>
<title>Servers</title>
</head>
<body>
<p>List of servers:</p>
```

```
<?php
passthru("ls /opsw/Server/@");
?>

</body>
</html>
```

上面的 passthru 语句将运行 ls 命令，并将 stdout（无 reinflate）传回网页。 ls 命令将列出托管服务器在 OGFS 中显示的名称。

4 保存 index.php 文件，并退出文本编辑器。

5 发布修改后的 Web 应用程序。

以下 apxtool import 命令将当前版本设置为 2。 -F 选项抑制确认提示。

```
$ apxtool import -f "/home/jdoe/mywebapp/com.hp.sa.jdoe.mywebapp" \
-c --version=2 -F
Info:Successfully published a new version '2' for APX 'mywebapp'
Info:Successfully set APX 'mywebapp'(310001) current version as '2'.
```

6. 运行修改后的 Web 应用程序

1 在 SA 客户端中，使用 “视图” ➤ “刷新” 菜单刷新 Web 扩展视图，此时该视图应包含 mywebapp 的版本 2。

2 选择 mywebapp，然后选择 “操作” ➤ “运行” 菜单。输出内容将类似于图 5，不同之处是它将显示 PHP passthru 语句和 OGSH ls 语句的输出，其中列出了所有托管服务器。请注意，passthru 语句将删除用于分隔 ls 命令所返回的服务器名称的换行符。

教程：创建程序 APX

本教程演示了如何创建、发布和运行名为 `myshellapp` 并运行简单 shell 脚本的简单程序 APX。本教程后面将告诉您如何修改 shell 脚本来调用 `apxprogress` 命令以及提供进度信息。因为本教程提供了源代码，所以无需预先掌握 shell 编程知识。

按照顺序完成以下任务。

教程先决条件

要完成本教程，您必须具有以下能力和环境：

- 可作为 `admin` 或其他**超级管理员**组成员登录 SA。以 `admin` 身份登录将允许您设置权限。
- 可作为属于**高级用户**组的用户登录 SA。

  高级用户有权创建和运行 Web 应用程序。在本教程所显示的示例命令中，该用户的名称是 `jdoe`。
- 了解如何在 SA Web 客户端中设置客户端功能权限。

  有关权限的详细信息，请参见 《SA 管理指南》的 “用户和组设置”一章。
- 了解如何在 SA 客户端中创建文件夹。

  有关文件夹的详细信息，请参见 《SA 用户指南：Server Automation》。
- 了解如何打开全局 Shell (OGSH) 会话和使用全局 Shell。

  有关说明，请参见 《SA 用户指南：Server Automation》的 “全局 Shell”一章。
- 了解 `ls` 和 `cd` 等基本 UNIX 命令。

1. 设置权限并创建教程文件夹

1 以 `admin` 身份登录 SA Web 客户端，并验证**高级用户**组是否具有以下权限：

  – 管理扩展：读取和写入

  可以在 SA Web 客户端上的 “客户端功能”选项卡中找到该权限。

2 以**高级用户**组成员身份登录 SA 客户端，并在 SA 库中创建以下文件夹：

  `/Dev/MyApp`

在本教程的后面，您将向 MyApp 文件夹上载程序 APX。在非教程环境中，此文件夹的名称可以是任意名称。可以创建或选择任何其他文件夹来包含 APX。

3 退出 SA 客户端。

4 以 admin 身份登录 SA 客户端，并打开 MyApp 文件夹的 “文件夹属性”。

5 在 “文件夹属性” 的 “权限” 选项卡中，确保**高级**用户组具有下列权限：

— 列出文件夹内容

— 在文件夹中读取对象

— 在文件夹中写入对象

— 在文件夹中执行对象

6 退出 SA 客户端。

2. 创建新程序 APX

1 以属于**高级用户**组的 SA 用户身份打开全局 Shell 会话。

2 在核心的 OGFS 主目录中，创建一个名为 myshellapp 的目录，然后更改该目录：

```
$ mkdir myshellapp
$ cd myshellapp
```

程序 APX 文件将存储在 myshellapp 目录中。

3 使用 apxtool new 命令创建程序 APX 的目录结构和默认文件，如下所示。

```
$ pwd
/home/jdoe/myshellapp
$ ls
$
$ apxtool new -ts -d "This is my first program APX."\
-u com.hp.sa.jdoe.myshellapp

Create source directory under '/home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp' for APX 'myshellapp'?Y/N y
Info:Successfully created source directory '/home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp for APX 'myshellapp'.
```

-ts 选项表示 APX 类型是程序 APX（也称为脚本 APX），-d 指定描述信息，-u 指定应用程序的唯一名称。

有关 apxtool new 命令选项的详细信息，请参见联机帮助：

```
$ apxtool new -h
```

---

4 列出由 apxtool new 命令创建的文件：

```
$ pwd
/home/jdoe/mywebapp
```

```
$ ls
com.hp.sa.jdoe.myshellapp
$ cd com.hp.sa.jdoe.myshellapp
$ pwd
/home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$ ls -R
.:
APX-INF  run.sh

./APX-INF:
apx.cfg  apx.perm  description.txt  interfaces  usage.txt
```

5 显示默认 run.sh 文件的内容：

```
$ cat run.sh
#!/bin/sh

###############################################################################
# APX myshellapp
#
# Created by: jdoe
#
###############################################################################
echo "This is APX myshellapp"
```

6 检查 APX-INF 目录中的一些文件。有关这些文件的详细信息，请参见 APX 文件 （第 77 页）。

APX-INF 目录包含特定于 APX 的信息。如以下 cat 命令所示，description.txt 文件中保存了使用 apxtool new 的 -d 选项指定的文本。

```
$ ls APX-INF/
apx.cfg  apx.perm  description.txt  interfaces  usage.txt
$ cat APX-INF/description.txt
This is my first program APX.$
```

以下 grep 命令显示了 APX 配置文件 apx.cfg 中的一些属性。type 和 uniquename 的值来自于 apxtool new 命令的 -t 和 -u 选项。有关 APX 配置文件的详细信息，请参见 APX 配置文件 - apx.cfg （第 77 页）。

```
$ grep "=" APX-INF/apx.cfg
type=script
name=myshellapp
unique_name=com.hp.sa.jdoe.myshellapp
command=run.sh
```

3. 向 SA 导入新 APX

导入 APX 时将执行以下操作：

— 将 APX 复制到在 SA 库中显示的文件夹。

— 为 APX 分配版本号。

输入 `apxtool import` 命令并使用 `y` 来响应提示消息，如下所示。`-f` 选项指定 SA 库中将用于存储 Web 应用程序的文件夹。 `-c` 选项设置 Web 应用程序的当前版本。

```
$ pwd
/home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$
$ apxtool import -f "/Dev/MyApp" -c
APX source is not specified.
Do you want to publish current directory: /home/jdoe/myshellapp/
com.hp.sa.jdoe.myshellapp? Y/N y
APX with unique name 'com.hp.sa.jdoe.myshellapp' does not exist.
Register it into the system? Y/N y
Info: Successfully registered APX 'myshellapp' (20001).
Info: Successfully published a new version '1' for APX 'myshellapp'
Info: Successfully set APX 'myshellapp'(20001) current version as '1'.
```

现在，您已发布了 APX，可以像 SA 用户那样在 SA 客户端中运行它。

4. 运行新 APX

现在，您已发布了 APX，可以在 SA 客户端中运行它。

1 以属于**高级用户**组的用户身份登录 SA 客户端。

2 在导航窗格中，选择 “库” 选项卡，然后选择 “按类型” 选项卡。

3 打开 “扩展” 节点，并选择 “程序” 节点。此时将显示 SA 库中的所有程序 APX。您将在此处看到您的 APX。如果未看到 `myshellapp`，请确保您拥有 1. 设置权限并创建教程文件夹（第 87 页）中所述的必需权限。

4 选择 APX。

5 选择 “操作” ➤ “运行” 菜单项。此时将显示 “运行程序扩展” 向导。

6 选择 “下一步” 按钮。

7 选择 “启动作业” 按钮。

8 当您的 APX 完成运行后，选择状态指示器以显示详细信息。

9 选择 “关闭” 按钮。

5. 修改 APX

在本节中，您将修改 `run.sh` 文件，并添加对 `apxprogress` 命令的调用以提供进度信息。

1 在全局 Shell 会话中，找到 APX 的 `run.sh` 文件。

```
$ cd /home/jdoe/myshellapp/com.hp.sa.jdoe.myshellapp
$ ls
APX-INF run.sh
```

2 在文本编辑器（如 vi）中打开 run.sh 文件。

3 使用以下几行替换 run.sh 的内容：

```
echo "This is a simple APX that uses apxprogress."

totalsteps=100
apxprogress -i $totalsteps -c 1

for i in `seq $totalsteps`; do
   apxprogress -c $i -m "myshellapx is running, working on step $i" #-d
   sleep 10
done
```

这些 apxprogress 命令指定该 APX 包含 100 个步骤，并且调用 apxprogress 100 次（对每个步骤调用一次，两次调用之间等待 10 秒）。有关详细信息，请参见显示 APX 的进度（第 79 页）。

如果要进行调试，可以将 “#-d” 更改为 “-d”，并手动运行 Shell 脚本以显示 apxprogress 命令在 stdout 上的输出。

4 保存 run.sh 文件，并退出文本编辑器。

5 发布修改后的 APX。

以下 apxtool import 命令将加载 APX 的新版本，并将当前版本设置为 2。-F 选项抑制确认提示。

```
$ apxtool import -f "/home/jdoe/myshellapp" \
-c --version=2 -F
Info: Successfully published a new version '2' for APX 'myshellapp'
Info: Successfully set APX 'myshellapp'(20001) current version as '2'.
```

6. 运行修改后的 APX

现在，您已经修改并重新发布了 APX，可像以前一样从 SA 客户端中运行它。

1 在 SA 客户端中，使用 “视图” ➤ “刷新” 菜单刷新程序扩展视图，此时该视图应显示 myshellapp 的版本 2。

2 选择 APX。

3 选择 “操作” ➤ “运行” 菜单项。此时将显示 “运行程序扩展” 向导。

4 选择 “下一步” 按钮。

5 选择 “启动作业” 按钮。

7. 在 Twister 界面中查看 APX 进度

`apxprogress` 命令将报告正在运行的 APX 的进度。您可以通过调用 API 方法 `JobService.getProgress()` 来获取此进度信息。本节说明如何从 Twister 界面中运行该方法。有关 SA API 的 Twister 界面的详细信息，请参见 API 文档和 Twister （第 21 页）。

1 在 SA 客户端中，选择 “作业和会话” 选项卡。

2 在作业列表中找到您的 APX。

3 记录 APX 作业的作业 ID 号。将在后面的步骤中使用此 ID 号。

4 通过在 Web 浏览器中输入以下 URL 来运行 SA Twist 界面：

`https://<core_host>:1032`

其中 `core_host` 是 SA 核心服务器的 IP 地址或主机名。此时将在 Web 浏览器中显示 SA API 的 Twist 界面。

5 选择 “Twister” 链接。此时将显示 SA API 的 Twister 界面，您可以在其中获取关于 API 接口、包和方法的完整信息，并可以运行方法。

6 找到并选择 `JobService` 接口，该接口位于 `com.opsware.job` 包中。

7 向下滚动并找到 `getProgress()` 方法。

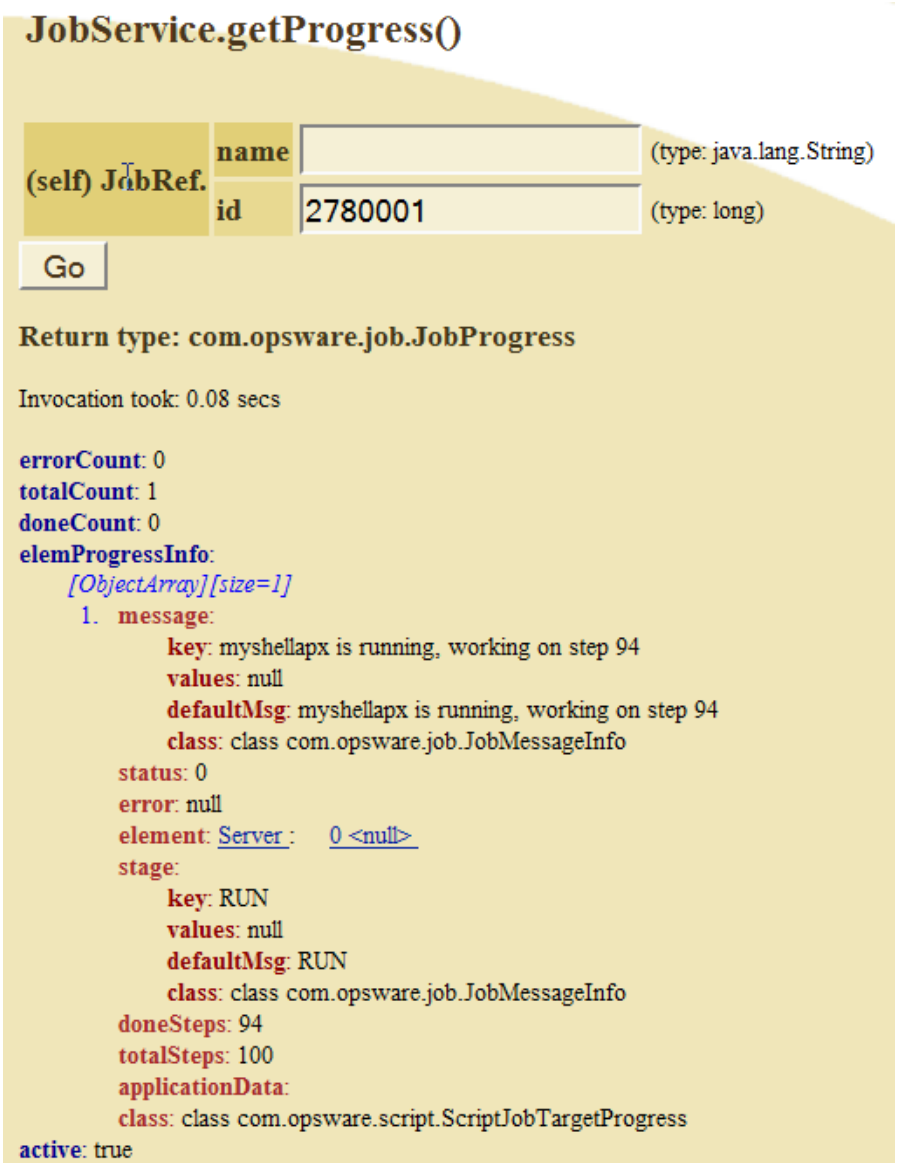
8 选择位于 `getProgress()` 方法正上方的 “尝试” 按钮。

9 输入 SA 凭据。

10 选择 “登录” 按钮。

11 在 “ID” 字段中，输入正在运行的 APX 的作业编号 （已从上面的步骤 3 中获取）。

12 选择 “Go” 按钮。此时将调用 `getProgress()` 方法，并显示来自 `apxprogress` 命令的 APX 当前进度信息，如下所示。请注意，在此截图中，步骤总数为 100，已完成的步骤数为 94。有关 `getProgress()` 方法的输出内容的详细信息，请参见 Javadoc 文档。可通过在 Twister Web 浏览器的导航窗格中选择 `getProgress()` 方法来访问该文档。



5 代理工具

代理工具简介

代理工具是一组经过专门设计的 shell 脚本、批处理文件和 Python 脚本，用于检索和修改托管服务器的信息。这些信息将从 SA 数据库中进行检索和修改。

通过使用脚本，您可以检索和修改诸如自定义字段、客户分配、自定义特性等数据。在具备这种能力之后，您可以自动完成许多在过去必须逐个服务器执行的程序。

此外，您可以将脚本检索的信息包含到自己设计的自定义脚本中。由于客户分配和自定义特性等信息对于不同的托管服务器有所不同，因此能够在自定义脚本中*实时*检索和使用这些信息非常重要。

例如：

- 您可能具有一个用于处理特定应用程序的后安装配置的脚本，该脚本必须能够发现注册服务器的设施的名称。代理工具将提供一个脚本来获取设施名称，并将其插入到后安装脚本，而无需人工干预。
- 在安装监控代理时，后安装脚本必须修改配置文件，以包括相应特定设施中的监控服务器的 IP 地址。代理工具将提供一个脚本，通过读取核心上的自定义特性来发现监控服务器的 IP 地址，以便将其插入配置文件。
- 可以编写 DSE 用于从很多服务器检索 EEPROM 版本，并将信息存储为自定义特性或自定义字段。

代理工具脚本的一些其他用途包括：

- 在软件安装过程中收集 SA 核心的信息，以在配置中使用。
- 执行 DSE、全局 Shell 脚本或软件安装时，在 SA 数据库中存储来自托管服务器的元数据。
- 检索托管服务器的自定义特性信息。

安装要求

代理工具套件要求满足下列条件：

操作系统支持

代理工具支持 SA 托管服务器所支持的操作系统。有关支持的操作系统的列表，请参见《SA Standard/Advanced Installation Guide》。

安全、访问控制和身份验证

在 Unix/Linux 系统上，代理工具必须作为*根用户*运行；在 Windows 系统上，必须作为*管理员*运行。代理工具使用服务器代理的证书连接到 Web 服务数据访问引擎 (twist)（这是 pyTwist 的默认行为），其被授予的权限与 Web 服务数据访问引擎向代理授予的权限相同。这通常适用于代理工具所在服务器上的读取 / 写入权限，因此，无需执行用户身份验证。

`set_customer` 脚本是一个例外。您必须具有客户读取访问权限，才能将服务器与该客户关联。代理证书没有其他客户的读取访问权限，因此用户必须在运行此脚本时进行身份验证。

其他要求

- pyTwist 的访问权限
- SA API 的访问权限
- 已安装 Python 2.4（服务器代理附带）

安装

在正常的 HP SA 安装程序核心安装过程中，会将代理工具安装在核心。但是，还必须在托管服务器上安装代理工具，使其在这些服务器上可用。本节描述了该过程。

代理工具将作为一组可执行脚本安装在托管服务器上。根据操作系统，这些脚本将是由 shell 和批处理脚本调用的 shell 或批处理脚本和 Python 脚本。可以从托管服务器运行这些脚本，以在 SA 核心中检索和修改信息。这些脚本可以手动运行，也可以从包安装脚本、DSE、全局 Shell 脚本等进行调用。

代理工具包含在 Python SA API Access (pyTwist) 软件策略中。该策略位于以下目录中：

```
/Opsware/Tools/Python Opsware API Access
```

手动安装代理工具

要在托管服务器上安装代理工具，请执行以下操作：

1. 启动 SA 客户端。
2. 转到 “托管服务器”列表，选择要安装代理工具的托管服务器。
3. 单击右键并选择 “安装软件”。
4. 选择 “Python Opsware API Access”软件策略。
5. “软件策略”安装向导将引导您完成其余过程。

在安装代理时安装代理工具

也可以指定 Python SA API Access 软件策略 ID，并指定在代理安装过程中对其进行修正。有关代理安装的信息，请参见 《SA 管理指南》。

升级代理工具

由于代理工具以软件策略形式（作为 pyTwist 软件策略的组成部分）提供，因此您可以在升级核心后通过执行修正操作来升级到更新版本的代理工具。

在升级 SA 核心时，Python SA API Access 软件策略也将更新。将删除任何旧版本的代理工具，并向策略中附加新版本。在升级 SA 核心之后 （将在升级核心过程中自动升级代理工具），可以在托管服务器上执行以下任务来升级代理务工具：

1. 选择已安装代理工具的托管服务器。可以通过打开 Python SA API Access 软件策略来查看附加到该策略的服务器和组的列表。
2. 右键单击所选服务器并选择 “修正”。
3. 选择 “Python Opsware API Access”软件策略。
4. 将删除旧版本的 pyTwist 和代理工具包，并安装新版本。

数据迁移

由于代理工具不会在托管服务器上保留永久数据，因此在数据迁移或保存方面没有要求。

代理工具脚本

用法

```
<scriptname>.py|bat|sh --arguments
```

表 21　代理工具脚本

脚本	功能
get_all_cust_attr	检索服务器记录的所有自定义特性。 用法：`get_all_cust_attr.py [--localonly] [--mode=python|shell|pretty]` 模式确定了输出格式（如 Python 字典、 shell 语句等）。 `Pretty` 是默认模式。 **注意**：当存在多行自定义特性时， Shell 模式无法工作。
get_cust_attr	检索单个自定义特性的值。 **用法**： `get_cust_attr.py [--localonly] <custom attribute name>`
set_cust_attr	设置服务器上单个自定义特性的值。 **用法**：`set_cust_attr.py <custom attribute name> <custom attribute value>|--valuefile <path to file with value in it>`
del_cust_attr	从数据库中的服务器记录中删除自定义特性。 **用法**：`del_cust_attr.py <custom attribute name>`
get_cust_field	检索单个自定义字段的值。 **用法**：`get_cust_field.py <custom field name>`
set_cust_field	设置服务器上单个自定义字段的值。 **用法**：`set_cust_field.py <custom field name> <custom field value>|--valuefile <path to file with value in it>`
get_customer	检索与服务器关联的客户名称。 **用法**：`./get_customer.py`
set_customer	设置与服务器关联的客户名称。 **用法**：`set_customer.py <customer name>`
get_facility	检索与服务器关联的设施的名称。 **用法**：`./get_facility.py`

表 21　代理工具脚本 （续）

脚本	功能
get_info	打印服务器的所有字段 （格式类似于 OGSH 中的服务器信息文件）。 **用法：** `get_info.py`

get_history	打印出服务器特定的事件。
	用法：
	`get_history.py --startdate <start date in seconds since epoch>`
	`[--enddate <end date in seconds since epoch>]`
	`[--username <SAS user name>] [--password <SAS password>]`

sub_text_file	在文本文件中读取数据、在文件中查找标记 / 参数、用自定义特性值替换它们，以及将修改的文件打印到 `stdout`。有关预期的文件格式的详细信息，请参见下文。 **用法：** `sub_text_file.py [--localonly] <path to file with tokens in it>`

sub_text_file 脚本的格式

传递到 `sub_text_file` 脚本的文本文件可以包含任何内容，但是该脚本将查找任何含有两个 @ 字符的行，并将这两个 @ 字符之间的字符串 （包括 @ 字符对）视为标记。您可以在一行中放置一个 @ 字符，此时该字符会被忽略。但是，同一行中的第二个 @ 字符会导致脚本将两个 @ 字符之间的任何文本视为标记。

标记将被 @ 符号之间指定的自定义特性的值替换。例如，字符串 `@dns_server@` 被自定义特性 `dns_server` 的值替换。如果该自定义特性不存在，或者它的值为空，则标记将被替换为一个空字符串。

例如，包含以下条目的文本文件：

```
IP:@monitoring_server_ip@
```

该脚本将输出类似如下所示的内容：

```
IP:82.159.202.117
```

其中 `IP` 是 `monitoring_server_ip` 检索到的值。

输出

`sub_text_file` 脚本会输出到 `stdout`。如果需要，可以将输出重定向到文件中。此外，还可以使用存储在 zip 文件中的 `.template` 文件来设置输出格式。例如：

```
$AGENTTOOLSPATH/sub_text_file.sh petstore_config.template >
petstore_config.cfg
```

示例代理工具脚本

下面是一些使用代理工具脚本的简单示例。

Unix/Linux

下面的示例将输出一条消息，其中包含用户在登录 Unix 服务器时在每日消息 (MOTD) 中看到的设施名称。

```
. /etc/opt/opsware/pytwist/pytwist.conf
facility_name=`$AGENTTOOLSPATH/get_facility.sh`
echo "You have connected to a server in the $facility_name facility.  For
hardware information on this server as stored in Opsware, run $AGENTTOOLSPATH/
get_info.sh." > /etc/motd
```

Windows

下面的 Windows 示例将在所有用户的桌面上输出一个包含服务器信息的文本文件。

```
call "C:\Program Files\Common Files\Opsware\etc\pytwist\
pytwist_conf.bat"

call"%AGENTTOOLSPATH%\get_info.bat" > "%SYSTEMDRIVE%\Documents and
Settings\All Users\Desktop\server_info_from_Opsware.txt"
```

不要对代理工具的路径执行硬编码，而必须执行以下操作：

1. 获取 PyTwist 配置文件：
Unix：
`./etc/opt/opsware/pytwist/pytwist.conf`
Windows：
call
`C:\Program Files\Common Files\Opsware\etc\pytwist\pytwist_conf.bat`

2. 使用环境变量：
Unix：
`$AGENTTOOLSPATH`
Windows：
`%AGENTTOOLSPATH%`

使用此方法可防止将来更改代理工具路径时脚本出现错误。

6 Microsoft Windows PowerShell/SA 集成

Microsoft Windows PowerShell 简介

Windows PowerShell 是适用于系统管理员和程序员的可扩展命令 shell，已与 Microsoft .Net 2.0 Framework 类库集成。它使用 .NET 公共语言运行时和 .NET Framework，并接受和返回 .NET 对象。因此，增强了可用于管理和配置 Windows 的工具和方法。

Windows PowerShell 提供了大量 *cmdlet*，这些 cmdlet 内置到 shell 中，提供了丰富的功能。Cmdlet 可以单独或组合使用，以执行更复杂的任务。

Windows PowerShell 不仅支持访问计算机的文件系统， PowerShell *提供程序* 还允许您访问数据存储，如注册表和数字签名证书存储。*提供程序* 是一个软件模块，可在服务和数据源之间提供统一的接口。

在 SA 中尝试使用 Windows PowerShell 时，将假定您熟悉 Microsoft Windows PowerShell 并能轻松使用。如果需要关于使用 PowerShell 的背景或说明信息，请访问 *http://www.microsoft.com* 。

因为所包含的 cmdlet 可以修改托管服务器上的数据，所以您必须深入了解 Windows PowerShell 及其用法。

Windows PowerShell 与 SA 集成

在运行 Windows 的托管服务器上， SA 提供了与 Microsoft Windows PowerShell 的初始集成。PowerShell 可在 SA 用户界面中使用，而 SA 数据可从标准 PowerShell 环境或任何 PowerShell 运行空间中获取。 *PowerShell 运行空间* 是 PowerShell 运行时系统的托管环境。

SA 中提供了下列 PowerShell cmdlet：

- Get-SASServer
- Set-SASServer
- Get-SASServer

SA 还包括一个 PowerShell *SA 提供程序*（在 PowerShell 环境中提供对 SA 核心内各对象的访问权限的组件）。

集成的 PowerShell/SA Cmdlet

表 22 列出并描述了 SA 中的集成 PowerShell/SA cmdlet。

表 22　PowerShell Cmdlet

Cmdlet	描述	参数
Get-SASServer	从指定的服务器检索服务器数据	-Credential <PSCredential> -Core <Hostname|IPAddress> -Name < ListOfHostnameFragments> | -Id <ListOfServerIDs>
Get-SASServer	检索指定作业的数据	-Credential <PSCredential> -Core <Hostname|IPAddress> -JobFilter <ListOfJobIDs>
Set-SASServer	检索托管服务器的列表	-Credential <PSCredential> -Core <Hostname|IPAddress> -Server <ServerVO>

安装要求

包含 cmdlet 和 PowerShell SA 提供程序配置集、配置和安装文件的 MSI 安装程序包，用于在系统管理员的 Windows 桌面上进行安装。

操作系统支持

- Windows Server 2003
- Windows Server 2008
- Windows Server 2008 R2 x64
- Windows Server 2012

安装

要实现 Microsoft Windows PowerShell/SA 集成，必须执行以下任务：

1. 在 OCC“库” ➤ “软件策略”中找到 Microsoft Windows PowerShell/SA Connector MSI 包。
2. 运行此 MSI 安装可定义特定于 SA 的 cmdlet 和 SA 提供程序的配置集。文件 `readme.rtf` 提供了最新信息。此外，还将安装 Microsoft Windows PowerShell 初始化脚本 `profile.ps1`（类似于 `.bashrc`）和一组示例 PowerShell 脚本（用于演示如何在 SA 环境中使用 PowerShell）。

默认情况下，此 MSI 会将连接器安装到 `C:\Program Files\Opsware\PsSas` 中。

文件 SAS-WSAPI.ps1 描述了如何在不使用 cmdlet 的情况下从 PowerShell 直接访问 WS-API。

Microsoft PowerShell 与 SA 功能集成

可在以下几种情况中使用 Microsoft PowerShell：

- 远程访问托管服务器
- 审核和快照规则
- DSE 脚本集成

远程访问托管服务器

在 SA 客户端中，可以为任何托管服务器 （对于服务器组不可用）打开远程 PowerShell 会话，就像打开远程终端一样。

1 启动 SA 客户端。

2 在导航窗格中，选择 “设备” ➤ “所有托管服务器”。

3 选择一个托管服务器并将其打开。

在 “设备资源管理器” 窗口中，从 “操作” 菜单选择 “启动远程 PowerShell”。

登录到远程 PowerShell 会话之后，将无法运行包含 *WMI 调用* 的脚本。如果尝试运行包含 WMI 调用的脚本，系统将显示 “访问被拒绝” 错误，即使您属于有权运行该脚本的组也是如此。

审核和快照规则

Microsoft PowerShell 已集成 SA 审核功能。在配置自定义脚本规则时，Microsoft PowerShell 脚本与批处理、Python 2 和 Visual Basic 脚本一起作为选项提供。有关审核的详细信息，请参见《SA 用户指南：审核与符合性》。

DSE 脚本集成

对于托管服务器，您可以设置将通过 Pytwist 调用 SA API 的 PowerShell 脚本，以使最终用户可以将这些脚本作为 DSE 或 ISM 控件调用。有关编写用于调用 Pytwist API 的脚本的详细信息，请参见通过 Pytwist 执行 Python API 访问 （第 51 页）。

示例会话

本节提供了四个演示如何使用 Windows PowerShell/SA 集成的场景。

- 场景 1 演示了如何从 SA 核心中提取托管服务器数据，以及如何修改这些数据并将其重新写入核心。
- 场景 2 演示了如何使用 Windows PowerShell/SA 集成将 SA 托管服务器数据导出到 Excel 电子表格中。
- 场景 3 演示了如何将 SA 核心安装为 Windows PowerShell PSDrive，以及如何在虚拟文件系统中导航。
- 场景 4 演示了如何列出 Windows PowerShell 环境中可用的所有 SA 对象类型。

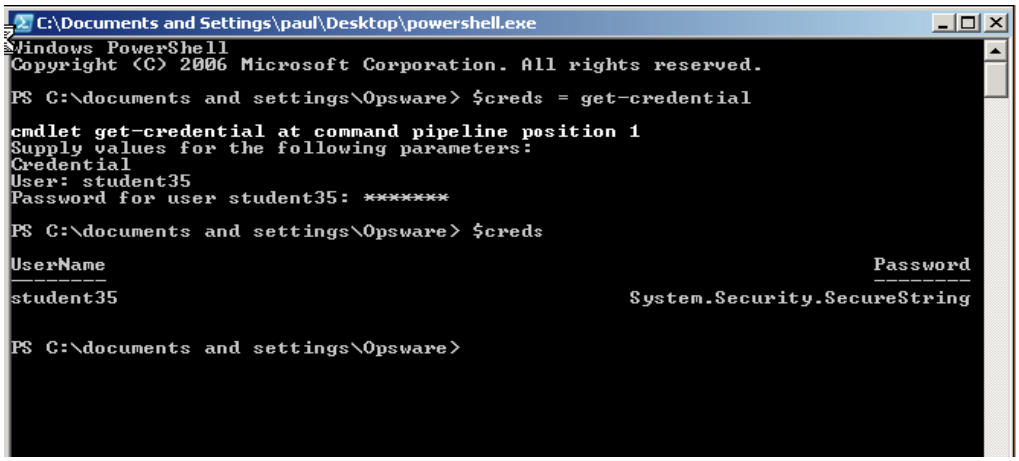
场景 1

向 SA 核心进行身份验证、获取关于托管服务器的数据、修改这些数据并将其重新写入 SA 核心。

1 通过桌面图标打开 PowerShell 提示符。

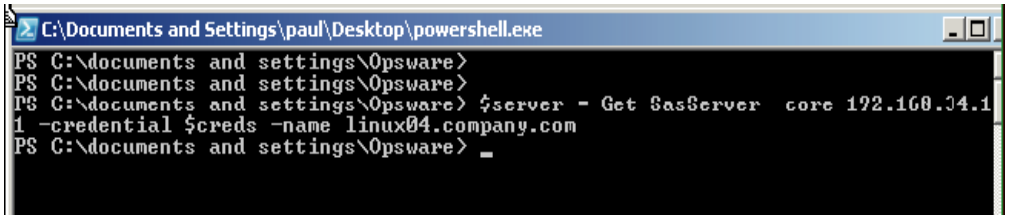
2 在 PowerShell shell 变量中安全地存储 SA 核心凭据。请参见图 6。

图 6 在 PowerShell 变量中存储 SA 凭据



3 使用 `Get-SasServer` cmdlet，您可以检索代表服务器的 SA 记录，如图 7 中所示。

图 7 使用 Get-SasServer cmdlet



返回的对象将存储在 shell 变量中。

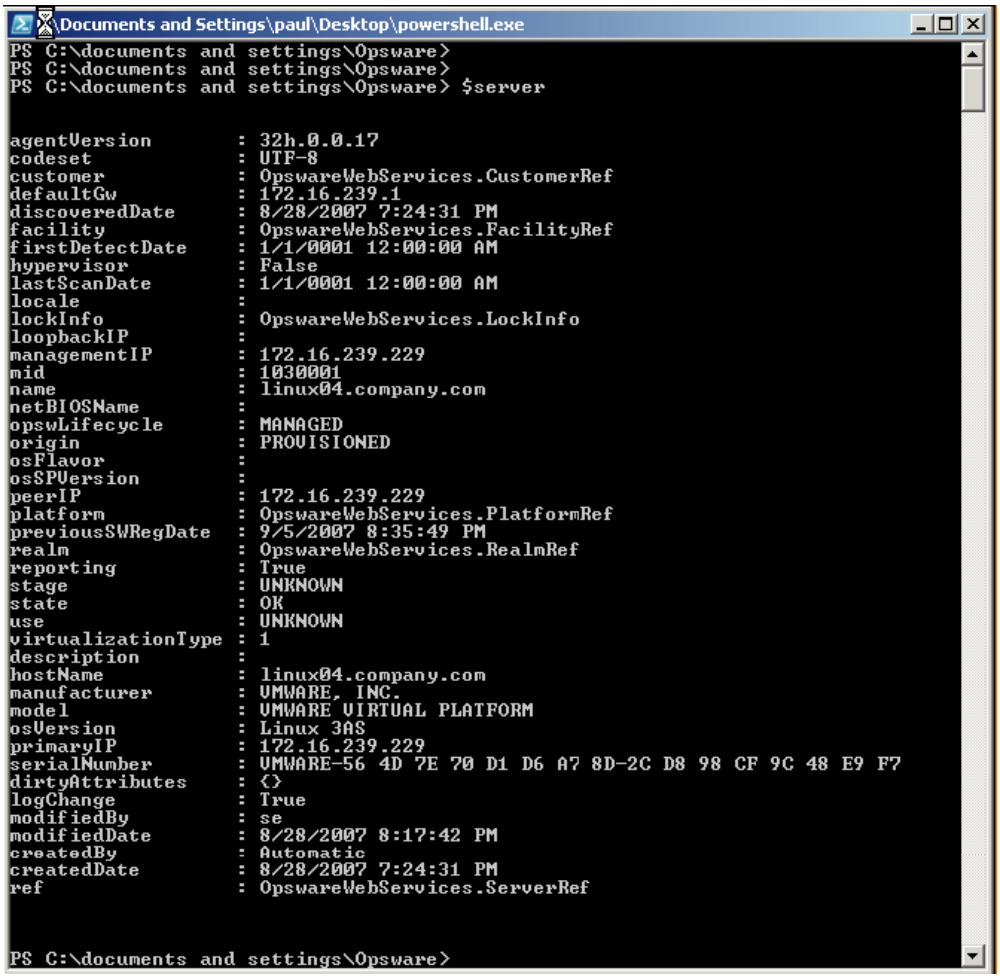
`Get-SasServer` cmdlet 将使用一个参数确定要从中检索服务器数据的 SA 核心，使用另一个参数向 SA 核心提供操作凭据（用于确定和验证要使用其身份尝试操作的 SA 用户帐户），还将使用一个参数来确定所请求的服务器。

可使用 PowerShell `Get-Help` 基础 cmdlet 获取 `Get-SasServer` cmdlet 参数或其他任何 cmdlet 的参数的详细信息，例如：

```
Get-Help Get-SasServer -detailed
```

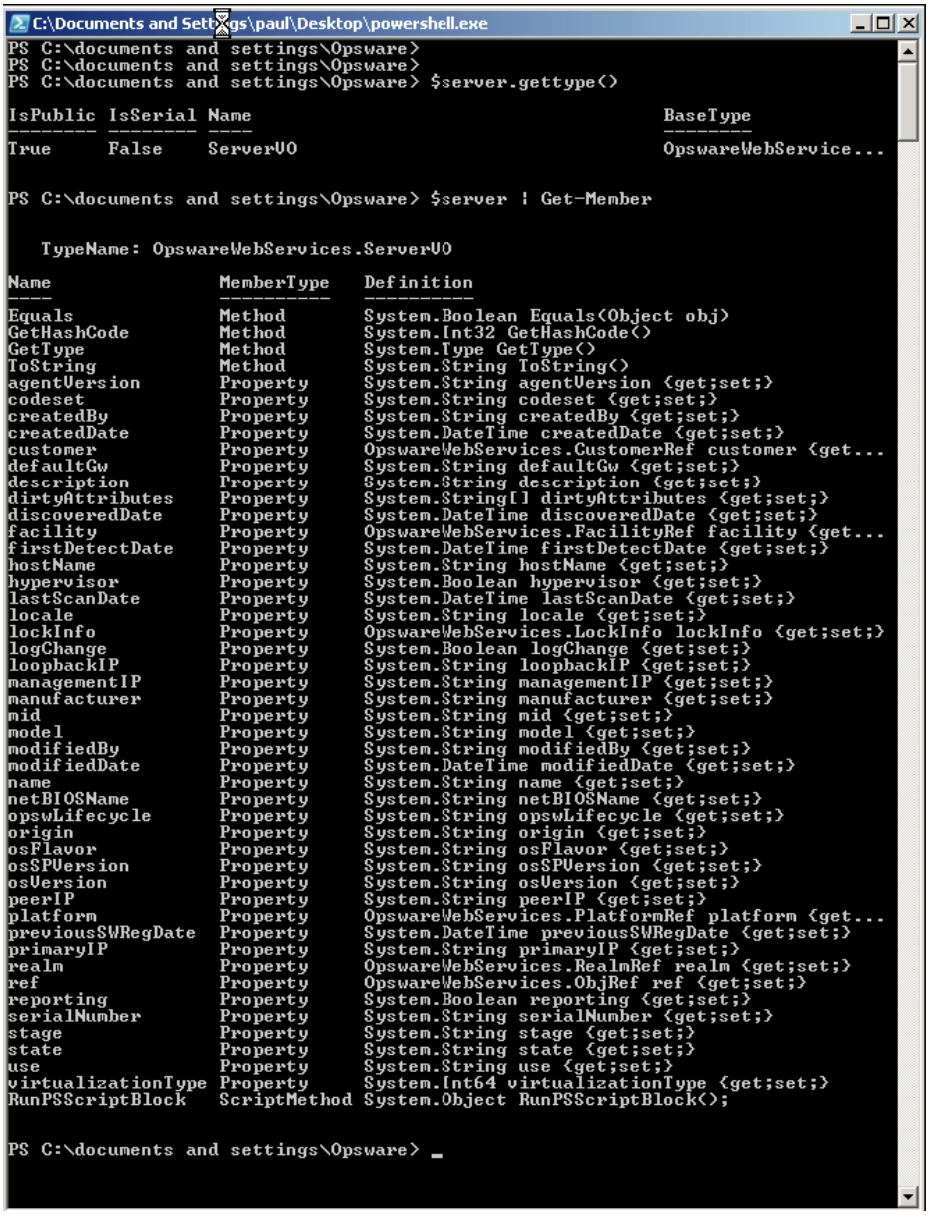
4 现在，可以通过输入 shell 变量名称来检查所返回对象的属性。请参见图 8。

图 8 检查 SA 服务器属性



5　列出对象属性、属性类型，以及可以通过 PowerShell 脚本在对象上调用的方法，如图 9 中所示。

图 9　列出对象的属性



6 可以在 Windows PowerShell 中修改对象的**描述**特性，然后调用 `Set-SasServer` cmdlet，并将修改后的 `ServerVO` 对象传递给 cmdlet。此 cmdlet 将接受 `ServerVO` 对象并更新 SA 核心中的托管服务器记录。 `Set-SasServer` cmdlet 将获取用于标识要写入更新数据的 SA 核心的参数，以及用于标识操作执行身份的 SA 用户帐户的凭据。

更新操作结束时，会将更新后的 `ServerVO` 返回到 Windows PowerShell，并在提示符处显示属性，如图 10 中所示。

图 10 修改对象的描述

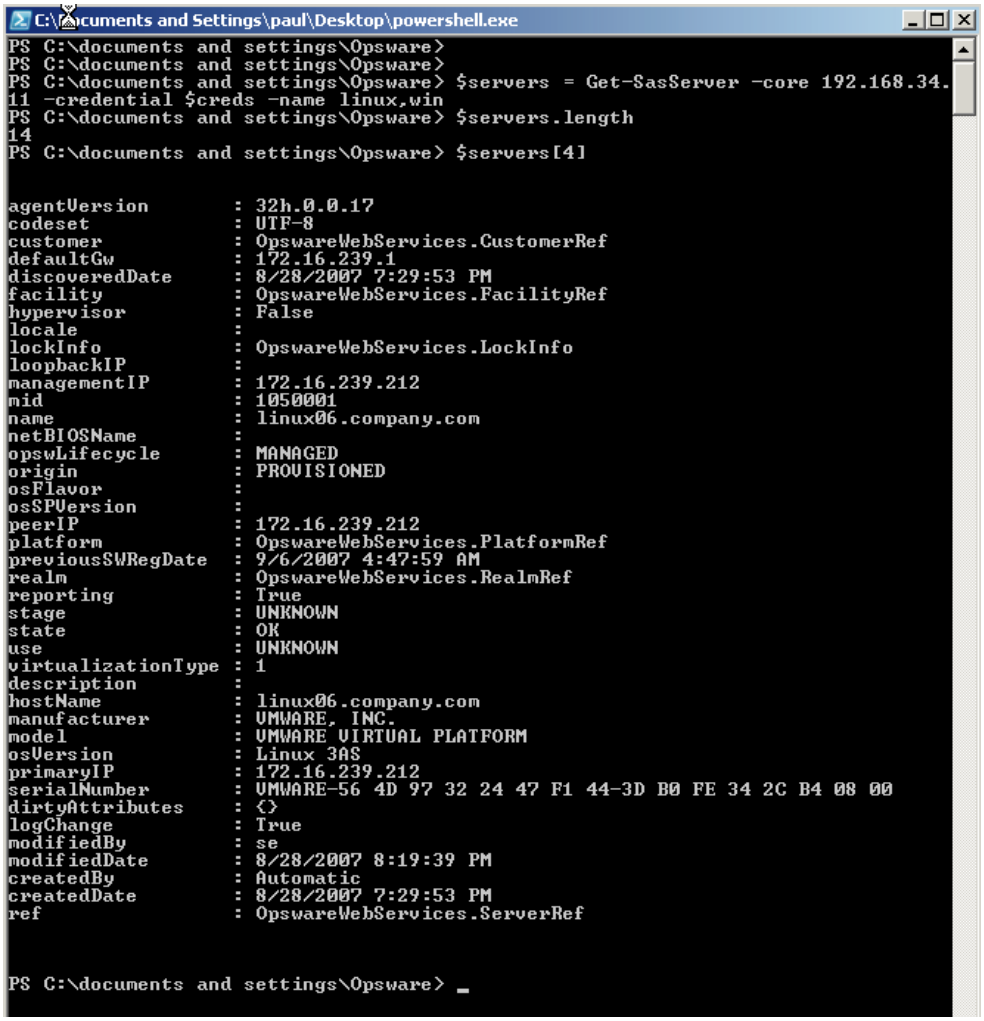
场景 2

此场景演示如何从 SA 核心检索所有托管服务器数据，并将其显示在 Microsoft Excel 中。

1 可使用 `Get-SasServer` cmdlet 从 SA 核心检索每个 Linux 和 Windows 托管服务器的 `ServerVO`。在下面的会话中，`-name` 参数用于向 SA 核心提供名称匹配筛选器列表，例如 `-name linux,win`。

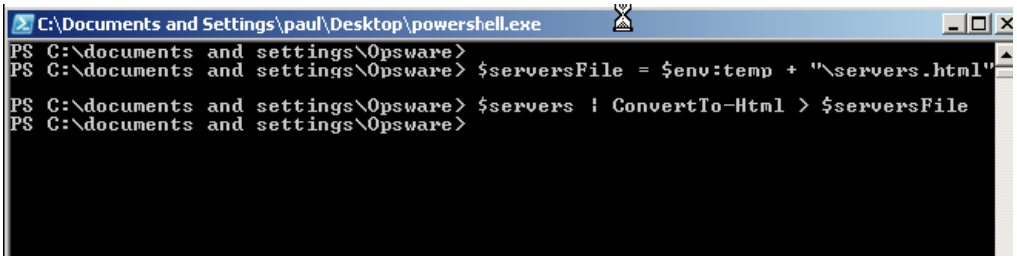
Get-SasServer cmdlet 将返回 ServerVO 数组，在该示例中是包含 14 项的数组。您可以对该数组进行索引，检查其中的任何一个 ServerVO 对象。请参见图 11。

图 11 将 Get-SasServer cmdlet 与名称筛选器一起使用



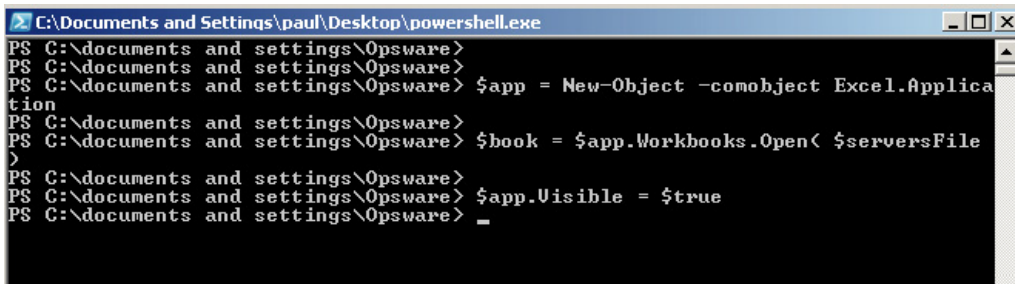
2 现在，可以将 ServerVO 数据的格式设置为 HTML，并将其保存到临时文件中。将在 TEMP 目录中创建临时文件。在 PowerShell 会话中，要获取 %TEMP% 环境变量的值，请输入 $env:temp。请参见图 12。

图 12 将 ServerVO 数据转换为 HTML 并保存到临时文件



3 使用 New-Object 基础 Windows PowerShell cmdlet，可以启动 Microsoft Excel，然后在该 Excel 实例中创建新工作簿，并使用临时文件的内容填充该工作簿。最后，将正在运行的 Excel 实例设置为可见。这样会将 Excel 置于前台。现在，可以按日期、列值等对数据进行排序，以确定诸如每个服务器接受 SA 核心管理的日期等信息。请参见图 13。

图 13 使用 New-Object cmdlet 启动 Microsoft Excel

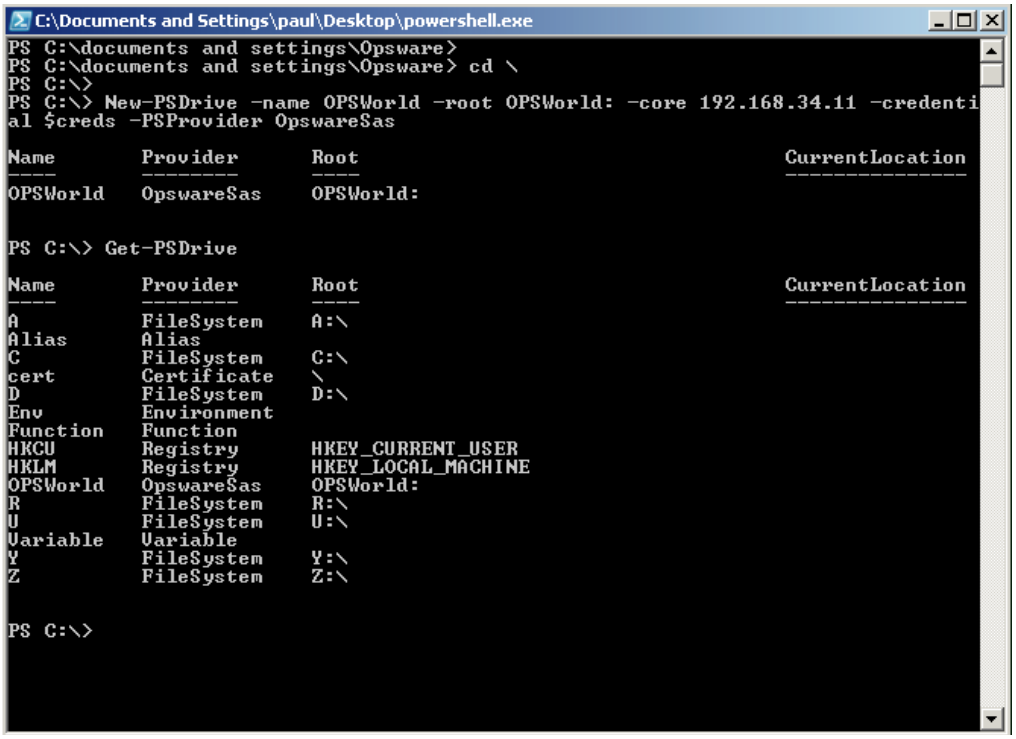


场景 3

此场景演示如何将 SA 核心安装为 Windows PowerShell PSDrive、如何导航到 SA **Jobs** 文件夹，以及如何检索其内容。

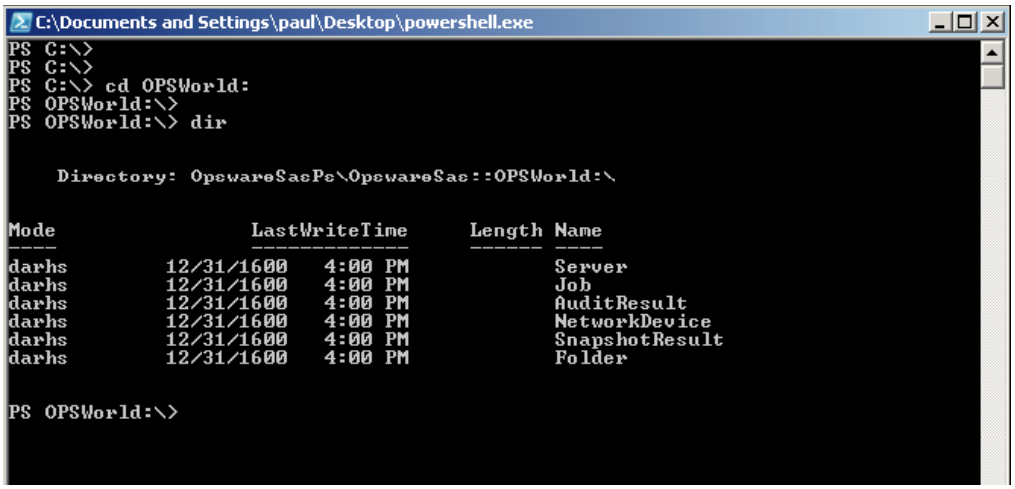
1 将 SA 核心安装为 Windows PowerShell PSDrive。PowerShell 支持将不同的数据存储或库当做文件系统一样导航。在此场景中，应*安装* SA 核心，特别是托管环境数据存储，在此假定它是名为 `OPSWorld` 的驱动器。然后，每当对此虚拟文件系统进行数据读取或写入或者客户端对此文件系统导航时， Windows PowerShell 基础系统就会调用 PowerShell SA 提供程序 – `PSProvider OpswareSas`。请参见图 14。

图 14 将 SA 核心安装为 Windows PowerShell PSDrive



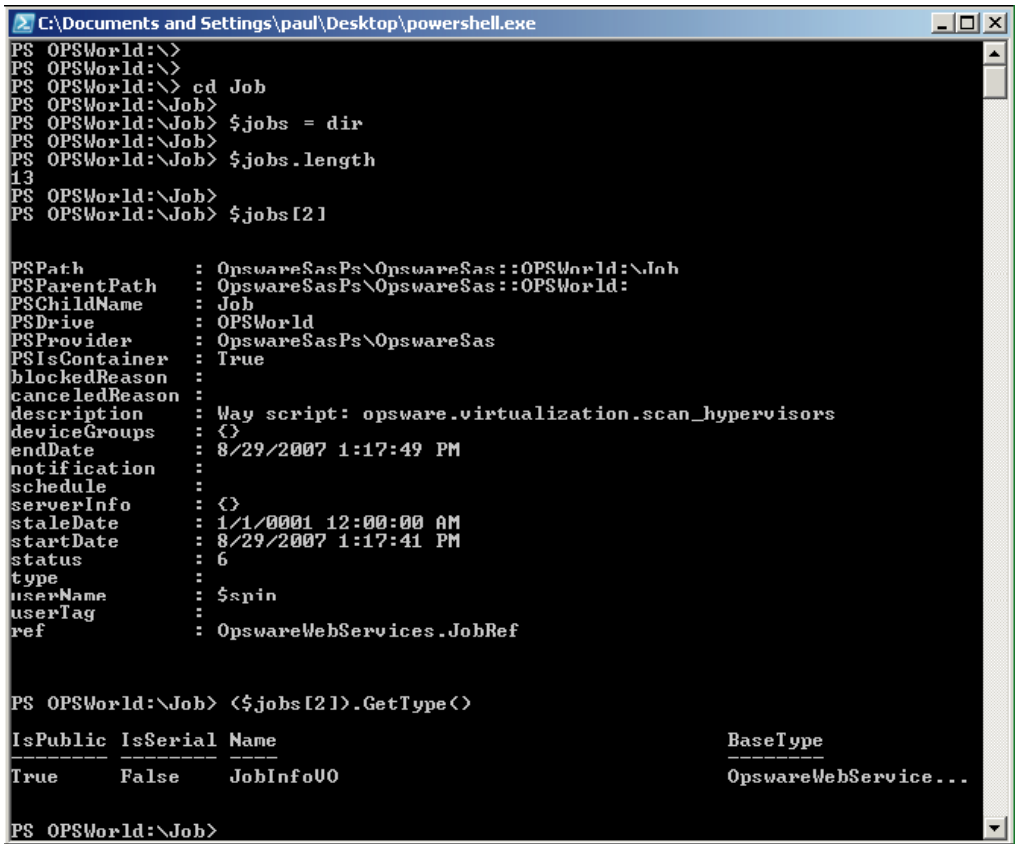
2 将目录更改为新安装的驱动器并获取目录列表。 `dir` 是 `Get-ChildItem` cmdlet 的 PowerShell 别名。请参见图 15。

图 15 DIR 充当 Get-Child cmdlet 的别名



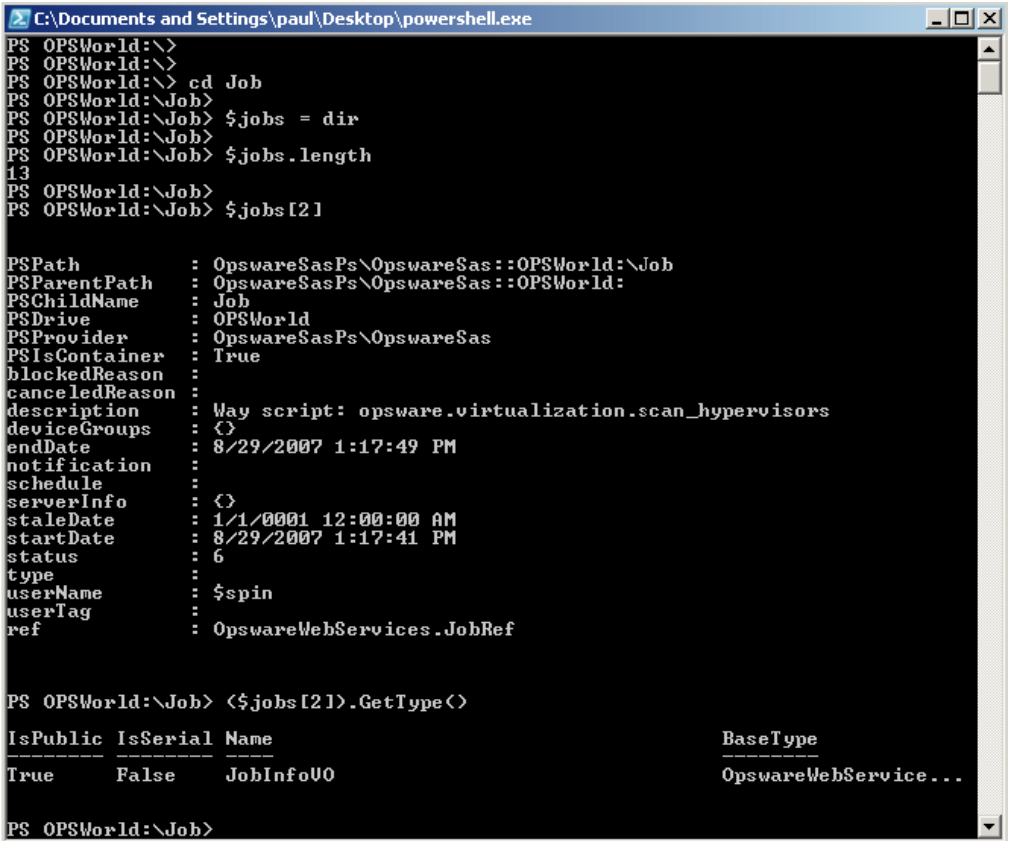
3 将目录更改为 **Jobs** 文件夹，获取目录列表并将其保存为 shell 变量。有关审核的详细信息，请参见该 shell 变量将包含 `JobInfoVO` 对象（来自 SA 核心）的数组，可以对该数组进行索引。请参见图 16。

图 16 将目录列表保存为 PowerShell 变量



4 将目录更改为 C: 驱动器并删除 OPSWorld PSDrive。请参见图 17。

图 17 删除 OPSWorld PSDrive

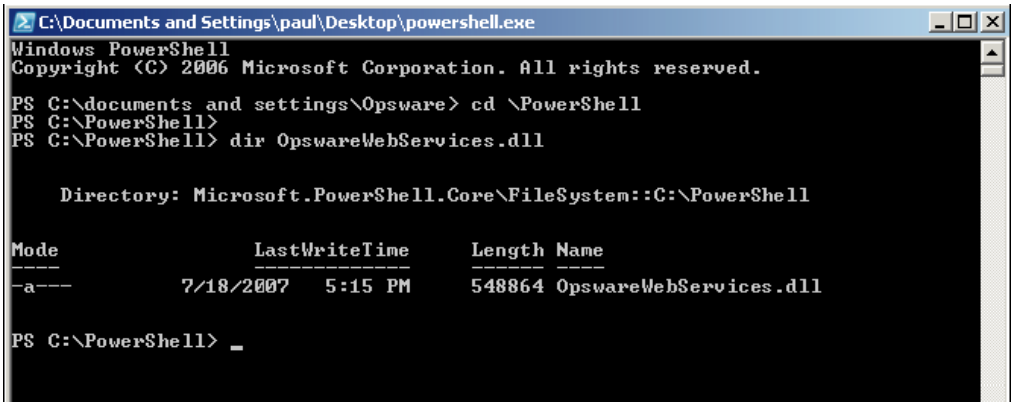


场景 4

此场景描述了如何检查 Windows PowerShell 环境中可用的所有 SA 对象类型。

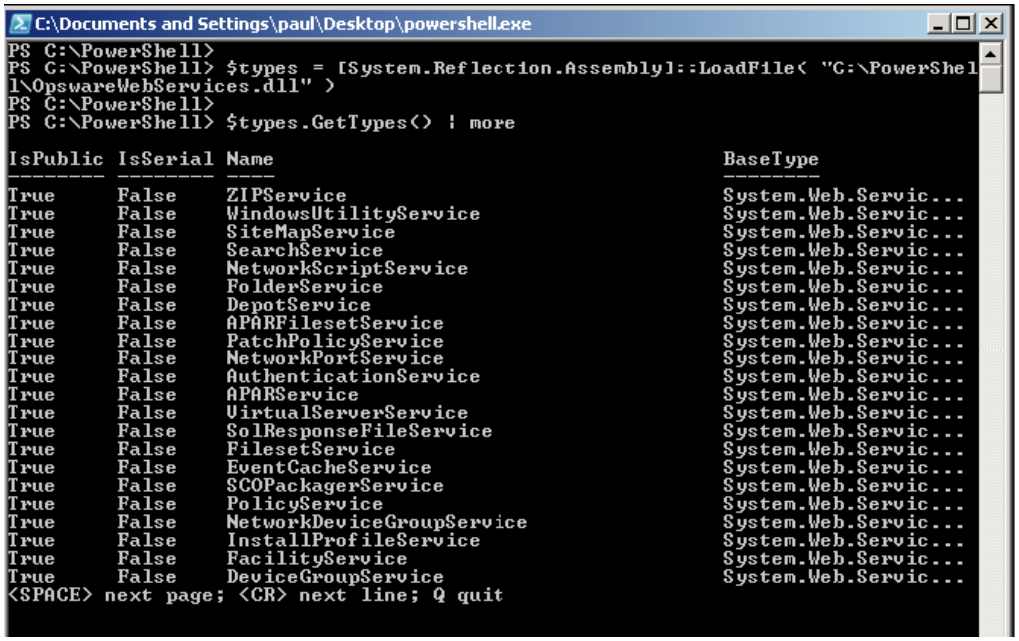
1 找到包含 PowerShell SA 提供程序和 cmdlet 的 .NET 配置集。请参见图 18。

图 18 找到包含 PowerShell SA 提供程序和 cmdlet 的 .NET 配置集



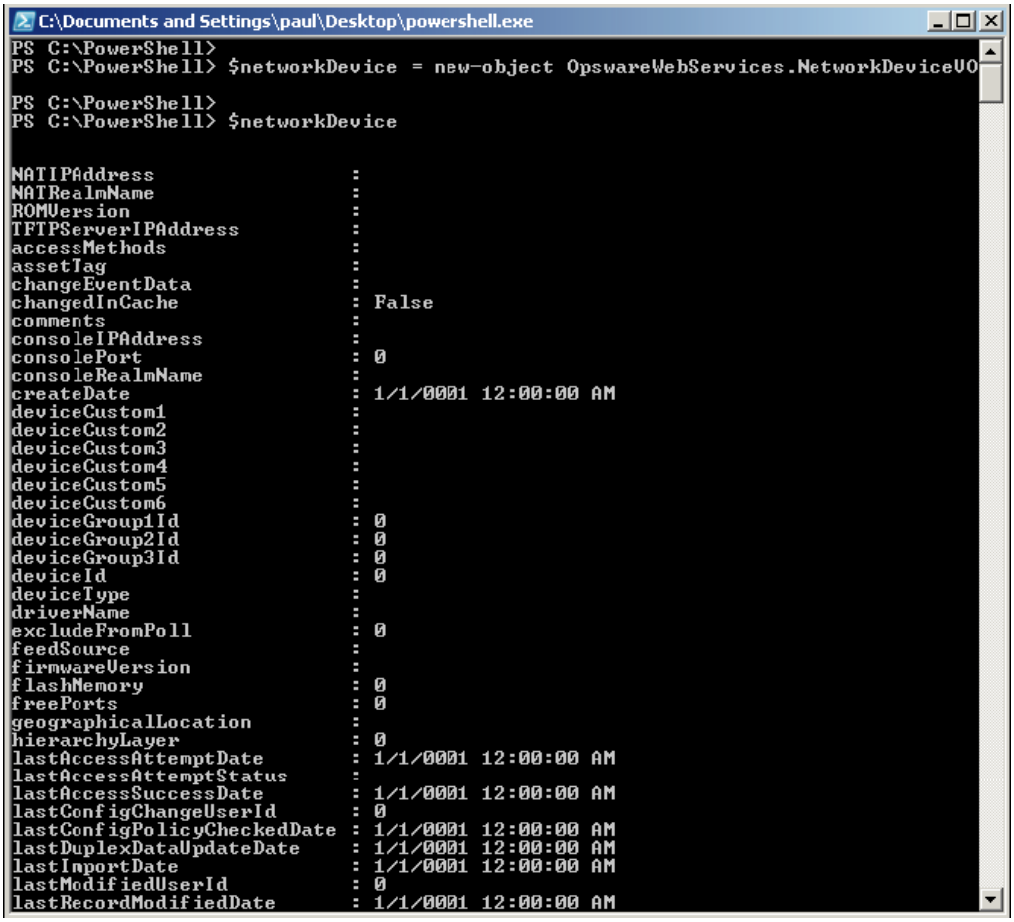
2 使用 .NET Reflection 加载 .NET 配置集，并检查所加载的类型。这将显示所有可用于 Windows PowerShell 环境的 SA 类型。请参见图 19。

图 19 加载 .NET 配置集并检查类型



3 创建 NetworkDeviceVO 实例。这是一个初期 NetworkDeviceVO，其中显示了 PowerShell 环境中可用于脚本、报告等功能的所有网络设备特性。请参见图 20。

图 20 创建 NetworkDeviceVO 实例



7 Java RMI 客户端

Java RMI 客户端概述

Java 远程调用 (RMI) 客户端可以从具有 SA 核心的网络访问权的服务器调用 SA API 方法。运行客户端的服务器不必是 SA 核心或托管服务器。当客户端连接到核心时，它会指定 SA 用户名和密码，这与登录到 SA 客户端的最终用户非常相似。用户所属的组确定了客户端可用的 SA 资源和任务。

本章的目标读者包括熟悉 SA 基础知识和 Java 编程语言的软件开发人员。

Java RMI 客户端安装

在开发 SA API 的 Java RMI 客户端之前，请执行以下步骤：

1. 在开发环境中安装 SA 核心。请不要使用生产核心。
2. 获取一台要在其中构建和运行 Java RMI 客户端的开发服务器。
3. 在开发服务器上，安装 Java SE 6 SDK。
4. 验证开发服务器是否已通过网络连接到运行 OCC 组件的 SA 核心服务器。
5. 将 `opswclient.jar` 文件从 SA 核心服务器下载到开发服务器中。`opswclient.jar` 文件包含 SA API 的 Java RMI 存根。在编译和运行 Java RMI 客户端时，应在 `classpath` 选项中包含 `opswclient.jar`。

   要下载 `opswclient.jar`，请指定以下 URL（其中 *occ_host* 是运行 OCC 组件的核心服务器）：

   `https://`*occ_host*`:/twister/opswclient.jar`

也需要 `spinclient-latest.jar` 和 `opsware_common-latest.jar` 文件。这些文件可从 /opt/opsware/twist/lib/ 中运行的 SA 核心获得。

当编译和运行这些示例时，必须将 `.jar` 文件添加到 classpath 参数中。

Java RMI 示例

本节描述了一个名为 GetServerInfo 的简单 Java RMI 客户端。

GetServerInfo 客户端通过完整或部分主机名搜索托管服务器，主机名可作为命令行参数指定。对于找到的每个托管服务器，客户端将打印服务器的名称、管理 IP 地址和操作系统版本。

GetServerInfo 客户端执行以下步骤：

1 连接到 SA：

```
OpswareClient.connect("https", host, (short)port,
userPasswd[0], userPasswd[1], true);
```

2 获取对 ServerService 接口的引用：

```
serverSvc = (ServerService)OpswareClient.getService
(ServerService.class);
```

3 在 ServerService 上调用方法：

```
ServerRef[] serverRefs = serverSvc.findServerRefs(filter);
. . .
ServerVO[] serverVOs = serverSvc.getServerVOs(serverRefs);
. . .
System.out.println(serverVOs[i].getName());
```

编译和运行 GetServerInfo 示例

在编译和运行该示例之前，请执行以下任务：

1 获取 opsware_common-latest.jar、spinclient-latest.jar 和 opswclient.jar 文件，如 Java RMI 客户端安装 （第 115 页）中所示。

2 从 HP Self Solve 下载 SA_Platform_Developer_Guide_examples.zip 文件 *(http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html)* 并提取文件 GetServerInfo.java。

3 要编译此客户端，请为 classpath 参数指定 opsware_common-latest.jar、 spinclient-latest.jar 和 opswclient.jar 文件：

```
javac -classpath :path/opswclient.jar:path/opsware_common-latest.jar:path/
spinclient-latest.jar GetServerInfo.java
```

4 要运行客户端，请输入以下命令，其中 *target* 是 SA 所管理的服务器的完整或部分名称（注意：Windows 的 Java classpath 分隔符是 “;”）：

```
java -classpath  .:path/opswclient.jar:path/opsware_common-
latest.jar:path/spinclient-latest.jar \
GetServerInfo [options]    target
```

在以下示例中， GetServerInfo 连接到主机 c44 （运行 OCC 核心组件的主机）和端口 443 上的 SA。程序显示主机名包含字符串 opsw 的托管服务器的信息。

```
java -classpath .:/home/jdoe/opswclient.jar:/home/jdoe/opsware_common-
latest.jar:/home/jdoe/spinclient-latest.jar \
GetServerInfo --host c44.dev.example.com --port 443 opsw
```

5 对 SA 用户名和密码提示消息做出响应。SA 用户必须有权读取与命令行上指定的 *target* 匹配的服务器。

8 Web 服务客户端

Web 服务客户端概述

SA API 支持 Web 服务，这是基于 SOAP（简单对象访问协议）和 WSDL（Web 服务定义语言）等开放行业标准构建的编程环境。您可以使用 Perl 和 C# 等各种编程语言（将在本章后面介绍），或通过支持 Web 服务的开发环境（例如 Microsoft Visual Studio .NET 和 BEA WebLogic Workshop）来创建 Web 服务客户端。

本章的目标读者包括熟悉 SA 基础知识和 Web 服务开发的软件开发人员。

本发布版中提供的编程语言绑定

本 SA 发布版包括用于 C# 的 Web 服务客户端存根。以 Perl 语言编写的 Web 服务客户端不需要客户端存根。

本发布版不包括用于 Java 或 Python 的 Web 服务客户端存根。但是，Java 客户端可以通过 RMI 访问 SA API， Python 客户端可以通过 Pytwist 访问 SA API，如前面章节中所述。

服务位置和 WSDL 的 URL

客户端可以从具有以下语法的 URL 访问 Web 服务，其中 *host* 是运行 OCC 核心组件的服务器，*port* 是 HTTPS 代理的端口。（默认代理端口为 443。）*packageName* 对应于服务所属的 Java 库。

```
https://host:port/osapi/packageName/WebServiceName
```

可从具有以下语法的 URL 访问 WSDL 文件：

```
https://host:port/osapi/packageName/WebServiceName?WSDL
```

例如，以下 URL 指向 FolderService 的位置和 WSDL：

```
https://occ.c38.example.com:443/osapi/com/opsware/folder/FolderService

https://occ.c39.example.com:443/osapi/com/opsware/folder/
FolderService?wsdl
```

SOAP 绑定样式为 RPC （远程过程调用），传输协议为 HTTPS。

Web 服务客户端安全性

与 SA API 的其他客户端一样，Web 服务客户端必须经过身份验证和授权才能在 SA 中执行操作。SA 核心中的客户端与 Web 服务组件之间的通信将加密。访问操作限制为通过 OCC 核心组件的 HTTPS 代理端口的 HTTPS 客户端。（默认端口为 443。）

重载操作

SA API 具有重载操作，但是 WSDL 2.0 规范不支持重载。在 SA API 中，重载操作以单个操作的形式由 Web 服务提供。

Java 接口支持

SA API 使用 Java 接口，但是 Web 服务不支持接口。解决该问题的方法是，使用 WSDL 文件将接口映射到 `xsd:anyType`。对于以面向对象的编程语言（如 C#）编码的客户端，如果 API 方法返回一个接口，则必须将返回类型强制转换为具体类。接口数组将转换为 `Object[]`。特定类型的数组成员将通过序列化 / 反序列化保留。有关 C# 代码的示例，请参见处理接口返回类型（第 129 页）。

不支持的数据类型

SA API 使用以下数据类型，但是 SOAP 不支持这些数据类型：

```
java.util.Properties
com.opsware.common.ModifiableMap
com.opsware.acm.ValueSet
com.opsware.swmgmt.PolicyOverrideFilter
```

Web 服务中省略的方法

以下 SA API 方法使用不受支持的数据类型作为参数或返回类型。因此，Web 服务中未以操作形式提供这些方法。

```
com.opsware.custattr.CustomAttribute.getCustAttrs
com.opsware.custattr.CustomAttribute.setCustAttrs
com.opsware.custattr.CustomField.getCustomFields
com.opsware.custattr.CustomField.setCustomFields
com.opsware.pkg.Patch.getPolicyOverrideRefs
```

对 java.util.Map 的部分支持

Axis 会将 `java.util.Map` 转换为 `apachesoap:Map`，即一组“键 - 值”对。对于 .NET，无法执行该转换。例如，C# 客户端将接收到空的“键 - 值”对数组。但是，该转换可在 Perl 语言编写的 Soap::Lite 中正常工作。因此，使用 `java.util.Map` 的 SA API 方法将在 Web 服务中以操作形式提供。

以下方法使用 `java.util.Map` 作为参数或返回类型：

```
com.opsware.acm.GroupConfigurable.getApplicationInstances
com.opsware.acm.ServerConfigurable.getCustAttrsWithRC
```

```
com.opsware.compliance.sco.CMLSnapshot.getValueSet
com.opsware.compliance.sco.CMLSnapshot.setValueSet
com.opsware.compliance.sco.SnapshotResultService.remediateCMLSnapshot
com.opsware.custattr.VirtualColumnVO.getConfigInfo
com.opsware.custattr.VirtualColumnVO.setConfigInfo
```

使用不受支持的数据类型的 VO 方法

以下 VO 方法使用不受支持的数据类型作为参数或返回类型：

```
com.opsware.acm.ApplicationInstanceVO.getValueset
com.opsware.acm.ApplicationInstanceVO.setValueset
com.opsware.acm.ConfigurableVO.getValueset
com.opsware.acm.ConfigurableVO.setValueset
com.opsware.virtualization.VirtualConfigNode.getProperties
com.opsware.virtualization.VirtualConfigNode.setProperties
com.opsware.virtualization.VirtualServerConfig.getProperties
com.opsware.virtualization.VirtualServerConfig.setProperties
```

创建或更新 VO 时调用 setDirtyAtrributes

Web 服务客户端必须首先调用 setDirtyAttributes，然后才能对服务调用 create 或 update 方法。 setDirtyAttributes 方法将显式标记需由 create 或 update 调用设置的 VO 的特性 （字段）。由 setDirtyAttributes 指定的特性名称区分大小写。

例如，为修改 FolderVO 对象的 description 特性，下列代码将首先调用 setDirtyAttributes，然后调用 update：

```
// fs is FolderService
FolderVO folderVO = fs.getFolderVO(folderRef);
folderVO.setDescription("credit card processing");
folderVO.setDirtyAttributes(new String[]{"description"});
fs.update(folderRef, folderVO, true, true);
```

Axis 从 XML 反序列化 XML 对象的方式要求必须对 Web 服务客户端调用 setDirtyAttributes。如果不调用 setDirtyAttributes， Axis 将对所有 VO 特性 （包括只读特性）调用 setter，从而导致 ReadOnlyException。

与 SA Web 服务 API 2.2 的兼容性

SA Web 服务 API 2.2 与本指南中描述的 SA API 不兼容。方法签名、服务、WSDL 和端口绑定不相同。如果要创建新的 Web 服务客户端，请务必使用 SA API 而不是 SA Web 服务 API 2.2。

Perl Web 服务客户端

本节包含用于创建将访问 SA API 的 Perl Web 服务客户端的分步式说明和示例代码。

Perl 客户端所需的软件

开发环境中必须具备以下 Perl 模块：

- Crypt-SSLeay-0.51
- IO-Socket-SSL-0.95
- Net_SSLeay.pm-1.25
- HTML-Parser-3.35
- MIME-Base64-3.01
- URI-1.30
- libwww-perl-5.76
- SOAP-Lite-0.65_6

基于您的 Perl 版本，可能需要这些模块的新版本。

运行 Perl 演示程序

要运行演示程序，请执行以下步骤：

1. 从 HP Self Solve 下载 SA_Platform_Developer_Guide_examples.zip 文件 *(http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html)* 并提取文件 `uapisample.pl`。
2. 编辑 `uapisample.pl` 文件，更改 `host`、`username`、`password` 以及诸如 `serverID` 等对象 ID 的硬编码值。
3. 运行 `uapisample.pl`。

如果收到 “`Certificate Verify Failed`” 错误，您必须从样本文件中取消备注下列行，并将有效路径提供给证书文件：

```
#$ENV{HTTPS_CA_FILE} = "path_to/opsware-ca.crt";
```

您可以从 SA 核心的以下路径查找该证书文件：

```
/var/opt/opsware/crypto/twist/opsware-ca.crt
```

Perl 示例代码

下面的代码片段摘自 uapisample.pl（包含在之前下载的 ZIP 文件中的一个 Perl 程序）。

设置服务 URI

```
# Construct the URI for the service.
#
my $username = "integration";
my $password = "integration";
my $protocol = "https";
my $host = "occ.c38.dev.example.com";
my $port = "443";
my $contextUri = "osapi/com/opsware/";
my $folderServiceName = "folder/FolderService";
my $folderUri   = "http://www.example.com/" . $contextUri .
$folderServiceName;

# Create a proxy to the FolderService.
#
my $folderProxy = $protocol . "://" . $username . ":" . $password . "@" .
$host . ":" . $port . "/" . $contextUri . $folderServiceName;
```

启动新服务

```
my $folderPort = SOAP::Lite
    -> uri($folderUri)
    -> proxy($folderProxy);
```

调用服务方法

```
my $root = $folderPort->getRoot()->result();
print 'Got root folder:' .$root->{'name'} . "\n";

# Alternative:
my $root = $folderPort->SOAP::getRoot();
print 'Got root folder:' .$root->{'name'} . "\n";
```

获取 VO

```
$rootVO = $folderPort->getFolderVO(SOAP::Data->name('self')
->value(\SOAP::Data->name('id')->type('long')->value(0)))
->result();

# The preceding call to getFolderVO does not pass a FolderRef
# parameter.If a method such as FolderService.remove accepts a
# FolderRef parameter, use the following code:
#
```

```
my $folderToBeRemoved = SOAP::Data->name('self')
->attr({ 'xmlns:ns_fs' => 'http://folder.example.com/FolderService'}) -
>type('ns_fs:FolderRef')->value(\SOAP::Data->name('id')->type('long') -
>value(123456));
$folderPort->remove($folderToBeRemoved);

# To see the Perl representation of the returned VO, you can use
# the Dumper method. This will help you understand how to
# construct the dirty attributes of a VO for a create or update
# method.
#
use Data::Dumper;
print Dumper($folderVO);
```

获取数组

```
# Construct $folder, the FolderRef before getting the array.
#
my $folder = SOAP::Data->name('self') ->attr({ 'xmlns:ns_fs' => 'http://
folder.example.com'}) ->type('ns_fs:FolderRef')->value(\SOAP::Data-
>name('id')->type('long') ->value($root->{'id'}));

# The getChildren method returns an array of FNodeReference
# objects.
#
my $children = $folderPort->getChildren($folder, SOAP::Data->name('type')-
>type('string')->value(''))->result();

foreach $child (@{$children}){
   print 'Get child: ' . $child->{'name'} . "\n";
}
```

构造对象数组

```
# For a function that takes an object array as a parameter,
# such the getVOs method, take the following approach:
# First, construct the Array object elements individually
# and put them in an array.
#
my @refs = [];
foreach my $ref (@{$myRefs}){
     # Assume myRefs was returned from a previous
     # Web Services call.
     my $object = SOAP::Data->name('FacilityRef')
                      ->value(\SOAP::Data->name('id')
                              ->type('long')
                              ->value($ref->{'id'}
                       )
                )->attr({ 'xmlns:facility' => 'http://locality.example.com'})
                   ->type('facility:FacilityRef');
     push @refs, $object;
}
```

```
# Second, construct an Array Object and put the array in it.
#
my $selves = SOAP::Data->name("selves" =>
                \SOAP::Data->name("element" => @refs)-
>type("facility:FacilityRef"))
             ->attr({ 'xmlns:facility' => 'http://locality.example.com'})
             ->type("facility:ArrayOfFacilityRef");
```

更新或创建 VO

```
# This example updates the description attribute of a ServerVO.
#
my $serverID = 40038;
my $server = SOAP::Data->name('self')->value(\SOAP::Data->name('id')-
>type('long')->value($serverID));

# Don't forget to set dirtyAttributes for the attributes
# you want to update. You also need dirtyAttributes for
# create methods that pass a VO.
#
my @dirtyAttrs = ('description');
my $serverVO = SOAP::Data->name('vo') ->attr({ 'xmlns:ns_ss' => 'http://
server.example.com'}) ->value(\SOAP::Data->value( SOAP::Data-
>name('description')->value('PERL_UPDATE_DESC')->type('string'), SOAP::Data-
>name('logChange')->value('false')->type('boolean'), SOAP::Data-
>name('dirtyAttributes' => \SOAP::Data->name("element" => @dirtyAttrs)-
>type("string")) ->type("ns_ss:ArrayOf_soapenc_string"), ));

my $force = SOAP::Data->name('force')->value('true')->type('boolean');
my $refetch = SOAP::Data->name('refetch')->value('true')->type('boolean');

# Call the update method.
#
print 'Invoking method serverWSPort.update...', "\n";
my $updatedServerVO = $serverWSPort->update(
                          $server,
                          $serverVO,
                          $force,
                          $refetch)->result();
print "New description: ", $updatedServerVO->{'description'}, "\n";
```

处理 SOAP 错误

```
# Make sure that you turn off on_fault subroutine in the
# "use SOAP::Lite ..." statement.
#
# The fault member of a SOAP return will be set if the Web
# Service call throws an exception.
# The following code tries to get a folder that does not exist:
#
my $testVO = $folderPort->getFolderVO(SOAP::Data->name('self') -
>value(\SOAP::Data->name('id')->type('long')->value(123456)));
```

```
if($testVO->fault){
    print $testVO->faultstring . "\n";
    # This will print the error msg.
    print "ExceptionName: " . getExceptionName($testVO) . "\n";     # A
NotFoundException should be displayed here
    # The code that deals with the error goes here....
}
. . .
# The following subroutine extracts the exception name from the
# returned faultdetail.
#
sub getExceptionName {
    my $fault = shift; #get the fault object
    if($fault->faultdetail->{'fault'}){
        return ref($fault->faultdetail->{'fault'});
    }
}
. . .
# As shown in the preceding code, it's easier to handle SOAP
# faults if you execute functions like this:
#
#     my $data = $port->function(...);
# Not like this:
#     $port->SOAP::function(...);
#     $port->function(...)->result;
```

构造 Web 服务的 Perl 对象

在调用 Web 服务操作之前，Perl 客户端必须设置输入参数所需的数据结构。API 文档 (javadoc) 以及服务的 WSDL 文件中提供了设置数据结构所需的信息。本节中的 Perl 代码示例显示了如何构造 getServerVO 操作的输入参数。代码后的分步式说明显示了从 API 文档和 WSDL 文件中获取输入参数信息的位置。

用于调用 getServerVO 的源代码

以下 Perl 代码将设置输入参数 self，然后调用 getServerVO 操作。该调用将检索 ID 为 12345 的托管服务器的 VO （值对象）。

```
# Create a top-level SOAP::Data object that represents the
# with the name self.
#
$self = SOAP::Data->name('self')

# The namespace corresponds to the schema of the data type
# of the SOAP:Data object. The name chosen (ns_ss) is
# arbitrary.
#
$self->attr({'xmlns:ns_ss =>
'http://server.example.com/ServerService'});

# Specify the type (ServerRef) for the parameter self, using the
# name of the namespace from the preceding statement.
```

```
#
$self->type('ns_ss:ServerRef');

# Create the value for the parameter.  The value is a pointer
# to a SOAP::Data object. The number 12345 is the SA ID of # a managed server.
#
my $id = SOAP::Data->name('id')->type('long')->value(12345);

# From the self object, point to the value.
#
$self->value(\$id);

# Finally, call getServerVO:
#
my $data = $serverPort->getServerVO($self);
if($data->fault){
   # Handle exceptions here ...
}
else{
   my $serverVO = $data->result;
}
. . .
```

getServerVO 设置信息的位置

要获取编写 getServerVO 调用的代码所需的信息，请执行以下步骤：

1. 在浏览器中，访问位于以下 URL 的 API 文档 (javadoc)：

   https://*occ_host*:1032/twister/docs/index.html

   其中 *occ_host* 是运行命令中心组件的核心服务器的 IP 地址或主机名。（有关在 Twister 中调用方法的说明，请参见 API 文档和 Twister （第 21 页）。）

2. 检查 API 文档，确定方法的输入参数和返回值。

   getServerVO 方法在接口 com.opsware.server.ServerService 中定义。在以下方法签名中，注意 getServerVO 将接受 ServerRef 作为参数，并返回 ServerVO：

   ```
   public ServerVO getServerVO(ServerRef self)
                  throws java.rmi.RemoteException,
                         NotFoundException,
                         AuthorizationException
   ```

3. 在浏览器中，指定以下 URL 打开 ServerService 的 WSDL 文件：

   https://occ_host/osapi/com/opsware/server/ServerService?wsdl

4. 在 WSDL 文件中，找到 ServerService 的命名空间：

   ```
   <schema targetNamespace="http://server.example.com" xmlns="http://
   www.w3.org/2001/XM
   LSchema">
   ```

以下 Perl 语句（来自前面列出的代码）指定了命名空间：

```
$self->attr({'xmlns:ns_ss =>
'http://server.example.com/ServerService'});
```

5 在 WSDL 文件中，找到 getServerVO 操作，并注意输入消息名称 getServerVORequest。

```
<wsdl:operation name="getServerVO" parameterOrder="self">
  <wsdl:input message="impl:getServerVORequest" name="getServerVORequest"/>
  <wsdl:output message="impl:getServerVOResponse" name="getServerVOResponse"/
>
  <wsdl:fault message="impl:NotFoundException" name="NotFoundException"/>
  <wsdl:fault message="impl:AuthorizationException"
name="AuthorizationException"/>
</wsdl:operation>
```

6 在 WSDL 文件中，找到 getServerVORequest 消息：

```
<wsdl:message name="getServerVORequest">
  <wsdl:part name="self" type="impl:ServerRef"/>
</wsdl:message>
```

getServerVORequest 消息元素定义了 getServerVO 的输入参数的名称 (self) 和类型 (ServerRef)。以下 Perl 语句指定了 ServerRef：

```
$self->type('ns_ss:ServerRef');
```

7 在 WSDL 文件中，找到 ServerRef 的 complexType：

```
<complexType name="ServerRef">
 <complexContent>
  <extension base="tns1:ObjRef">
   <sequence>
    <element name="secureResourceTypeName" nillable="true"
type="soapenc:string"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

请注意 ServerRef 扩展了 ObjRef。

8 在 WSDL 文件中，找到 ObjRef 的 complexType：

```
<complexType abstract="true" name="ObjRef">
 <sequence>
  <element name="id" type="xsd:long"/>
  <element name="idAsLong" nillable="true" type="soapenc:long"/>
  <element name="name" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
```

在 ObjRef 中，注意名称 (id) 和类型 (long)。这些数据类型在以下 Perl 语句中指定：

```
my $id = SOAP::Data->name('id')->type('long')->value(12345);
```

C# Web 服务客户端

本节包含用于创建将访问 SA API 的 C# Web 服务客户端的分步式说明和示例代码。

C# 客户端所需的软件

要开发 C# Web 服务客户端，您的开发环境必须具备以下软件：

- Microsoft .NET Framework SDK 1.1 版
- SA API 的 C# 客户端存根

获取 C# 客户端存根

SA 为每个服务提供一个存根文件，例如 `FolderService.cs`。所有存根都具有相同命名空间：`OpswareWebServices`。除存根之外， SA 还提供了 `shared.cs` 文件，该文件包含 `ServerRef` 等共享类。

要获取包含 C# 存根的 ZIP 文件，请指定以下 URL，其中 occ_host 是运行 OCC 组件的核心服务器：

```
https://occ_host:1032/twister/opswcsharpclient.zip
```

在服务和对象中定义的常量未在 C# 存根中进行定义。要获取有关常量的信息，请使用 API 文档 (javadoc)，如常量字段值 （第 22 页）中所述。

访问 C# 存根文档

Ndoc 生成的引用文档作为已编译的 Windows 帮助文件提供，与 C# 存根包含在同一个 ZIP 文件中。（NDoc 将从 .NET 配置集和 C# 编译器输出的 XML 文档文件生成代码文档。）该引用文档包含关于类层次结构和成员方法签名的的语法信息（但不包含描述性信息）。有关说明，请参见API 文档和 Twister （第 21 页）中介绍的相应 javadoc。

构建 C# 演示程序

要构建演示程序，请执行以下步骤：

1. 从 HP Self Solve 下载 SA_Platform_Developer_Guide_examples.zip 文件 *(http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html)* 并提取文件：
   - `App.config` - 应用程序设置
   - `WebServicesDemo.cs` - 用于调用服务方法的客户端代码
   - `MyCertificateValidation.cs` - 证书验证类
2. 创建以下目录：

   ```
   C:\wsapi
   ```

3 从 Visual Studio 2008 起始页中，选择 “New Project”，并创建一个具有以下值的项目：

- 项目类型：Visual C# 项目
- 模板：控制台应用程序
- 名称：WSAPIDemo
- 位置：C:\wsapi

此操作将创建一个包含某些文件的新目录 C:\wsapi\WSAPIDemo。

4 在新项目中，从对象列表中删除默认文件 Program 和 AssemblyInfo.cs。

5 将在 step 1 中获取的文件复制到 C:\wsapi\WSAPIDemo 目录。

6 从获取 C# 客户端存根 （第 127 页）中指定的 URL 下载客户端存根。

7 将 C# 客户端存根复制到 C:\wsapi\WSAPIDemo 目录。

8 将在上述两个步骤中复制的文件添加到 WSAPIDemo 项目：

- 在 Visual Studio 中，从 “Project” 菜单中选择 “Add Existing Item”。
- 浏览至目录 C:\wsapi\WSAPIDemo，并选择所有的演示文件 （.cs 和 .config）。

9 向 System.Web.Services.dll 添加引用：

- 在 Visual Studio 中，从 “Project” 菜单中选择 “Add Reference”。
- 在 .NET 标记下，浏览到具有以下名称的组件：System.Web.Services.dll。
- 依次单击 System.Web.Services.dll、“Select” 和 “OK”。

10 如果在创建项目时使用了不同模板，则可能需要将引用添加到 System、 System.XML 和 System.Data。检查项目引用以确定是否需要添加这些引用。

11 在 App.config 文件中，更改 username、 password、 host 以及诸如 serverID 等硬编码对象 ID 的值。

12 在 Visual Studio 中，从 “Build” 菜单中选择 “Build WSAPIDemo”。

运行 C# 演示程序

要运行演示程序，请执行以下步骤：

1 打开 Visual Studio 2008 命令提示符：

开始 ➤ 所有程序 ➤ Microsoft Visual Studio 2008 ➤ Visual Studio Tools ➤ Visual Studio 2008 Command Prompt

2 将目录更改为：

```
C:\wsapi\WSAPIDemo\bin\Debug
```

3 输入以下命令：

```
WSAPIDemo.exe
```

C# 示例代码

下面的代码片段摘自 WebServicesDemo.cs（包含在之前下载的 ZIP 文件中的一个 C# 程序）。

设置证书处理方式

```
# This setup is required just once for the client.
#
ServicePointManager.CertificatePolicy = new MyCertificateValidation();
```

分配 URL 前缀

```
# This is the URL prefix for all services.
#
wsdlUrlPrefix = protocol + "://" + host + ":" + port + "/" + contextUri + "/";
```

启动服务

```
FolderService fs = new FolderService();
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
```

调用服务方法

```
FolderRef root = fs.getRoot();
FolderVO vo = fs.getFolderVO(root);
```

处理接口返回类型

```
# In the API, FolderVO.getMembers returns an array of
# FNodeReference interfaces, but Web Services does not support
# interfaces. In the C# stub, the return type of
# FolderVO.members is Object[]. If a returned Object type will
# be used as a parameter that must be a specific type, then you
# must cast it to that type. For example, the following code
# casts elements of the returned array to FolderRef as
# appropriate.
#
Object[] members = vo.members;
for(int i=0;i<members.Length;i++)
{
Console.WriteLine("Got object: " + members[i].GetType().FullName + " --> " +
((ObjRef)members[i]).name);
   if(members[i] is FolderRef) {
   Console.WriteLine("I am a FolderRef: " +
   ((FolderRef)members[i]).name);
   }
}
```

更新或创建 VO

```
# When updating a VO, the changed attributes must be set in
# dirtyAttributes. (The VO passed to a create method has
# the same requirement.)
#
# Note: If you update a VO that was returned from a service
# method invocation, such as getFolderVO, then you must
# set the logChange attribute of the VO to false:
#    vo.logChange = false;
#
# The following code changes the name of a folder.
#
Console.WriteLine("Changing name from " + vo.name +
" to yo_csharp.");
vo.name = "yo_csharp";
vo.dirtyAttributes = new String[]{"name"};
# Manually set dirty fields being changed.
#
vo = fs.update(folder, vo, true, true);
Console.WriteLine("Folder name changed to: " + vo.name);
```

处理异常

```
# .NET converts Web Services faults into SoapExceptions
# without trying to deserialize them into application
# exceptions first. As a result, your code cannot catch
# application exceptions. As a workaround, the C# stubs
# provided by SA include SOAPExceptionParser,
# a class that enables you to get information from
# SOAPExceptions. The following code shows how to get the
# exception name and error message by calling the getDetail
# method of SOAPExceptionParser.
#
try{
// Try to get a non-existent folder here.
} catch(SoapException e){
    SoapExceptionDetail detail =
    SoapExceptionParser.getDetail(e);
    Console.WriteLine("SoapExceptionDetail.name: " +
    detail.exceptionName);
    Console.WriteLine("SoapExceptionDetail.msg: " +
    detail.message);
...
}
```

使用 C# 时的密码安全性

FolderService 方法将从 App.config 文件读取用户名和密码对。下面显示了该方法的示例。

```
User user = new User();
user.username = "user";
user.password = "password";
FolderService fs = new FolderService();
fs.Url = wsdlUrlPrefix + "com.opsware.folder/FolderService";
fs.user = user;
```

如果您不希望在 App.config 文件中以明文形式存储密码，可以使用 SecureUser 类对密码进行加密。SecureUser 类使用 .NET 2.0 中的 C# SecureString。密码将以加密形式存储在 SecureString 中。此外， getPassword() 仅在内部可见。 SecureUser 是一个静态类，因此您只需一次性设置用户名和密码，或在每次切换用户时进行设置。为了向后兼容，每个服务首先会从 SecureUser 检索用户名和密码，然后从其用户成员变量检索，最后从 App.config 中检索。 SecureUser 将获取密码的 String 或 SecureString。在任一种情况下，客户端都负责清理传递给 SecureUser.setUser() 方法的密码变量。

在某一时刻，密码需在内存中转换为常规 C# 字符串，该字符串将一直存在于内存中，直到下一次执行垃圾回收操作。如果使用 SecureUser，将只能确保内部密码存储是安全的。

以下示例显示了如何安全地设置用户名和密码。

```
SecureString passwd = new SecureString();
passwd.AppendChar('p');
passwd.AppendChar('a');
passwd.AppendChar('s');
passwd.AppendChar('s');
passwd.AppendChar('w');
passwd.AppendChar('d');
SecureUser.setUser("username", passwd); // that's it, no need to set up user
for each service.
passwd.Dispose(); // resets passwd and frees up memory so no copy remains from
caller.
```

9 可插入检查

可插入检查概述

SA 审核和修正功能允许您定义和监控 SA 托管服务器的符合性信息。由于符合性标准在不断发展，因此 SA 允许您创建专门的自定义检查和策略，以及扩展已在 SA 中提供的检查和策略。可插入检查是审核规则，属于一个或多个审核策略。可以使用 SA 客户端在命令行环境中创建可插入检查、上载该检查，然后将它添加到审核策略。

本章的目标读者包括熟悉 XML 以及 SA 的审核和修正功能的软件开发人员。

可插入检查安装

在开发可插入检查之前，请执行以下步骤：

1 在开发环境中安装 SA 核心。请不要使用生产核心。

2 在已安装代理的服务器上，安装 OCLI 1.0。有关 OCLI 1.0 的信息，请参见 《SA 用户指南：Server Automation》。

可插入检查教程

本教程演示了如何创建名为 HelloWorld Check 的可插入检查。该简单检查将验证 `/var/tmp/helloworld` 文件是否存在于 Unix 托管服务器上。如果该文件不存在，则此可插入检查的修正脚本将创建该文件。

要开发 HelloWorld Check，请执行以下步骤：

1 按照可插入检查安装 （第 133 页）中的说明执行操作。在本教程中，安装了 OCLI 1.0 的服务器将充当开发服务器。

2 HelloWorld Check 示例代码包含在含有 API 代码示例的 ZIP 文件中。从 HP Self Solve 下载 SA_Platform_Developer_Guide_examples.zip 文件 (*http://support.openview.hp.com/selfsolve/document/KM00417670/binary/SA_10_Platform_Developer_Guide_examples.html*)。

3 压缩在之前步骤中下载的文件，并验证 `pluggable_checks/helloworld` 目录中是否包含以下文件：

```
config.xml
gethelloworld.py
sethelloworld.py
```

HelloWorld Check 由下列三个文件组成。config.xml 文件是配置文件。gethelloworld.py Python 脚本执行审核。 sethelloworld.py Python 脚本执行修正。在下面的步骤中，会将这些文件打包成一个 ZIP 文件，然后将该 ZIP 文件导入 SA。

4 在开发服务器上，将解压缩后的 helloworld 文件复制到工作目录，例如：

```
cd /home/jdoe/dev
mkdir helloworld
cd helloworld
cp unzip_dest/pluggable_checks/helloworld/* .
```

5 获取全局唯一 ID (GUID)。每个可插入检查都需要一个 GUID。可以使用下列方法之一获取有效的 GUID：

— 登录网站，如下列网站：

http://kruithof.xs4all.nl/uuid/uuidgen

— 从以下网站下载免费的 Windows 工具 guidgen：

http://www.microsoft.com/downloads/details.aspx?FamilyID=94551F58-484F-4A8C-BB39-ADB270833AFC&displaylang=en

如果您以编程方式创建 GUID，则代码应遵循 RFC4122 (http://www.ietf.org/rfc/rfc4122.txt)。

6 使用文本编辑器在 config.xml 文件中插入 GUID，例如：

```
<checkGUID>6c7ed38c-d8d6-11db-8314-0800200c9a66</checkGUID>
```

这是本教程中需要在 config.xml 文件中修改的唯一元素。

7 在文本编辑器中，在 config.xml 中保存对 GUID 的更改。

将文本编辑器保持打开状态。在本教程中，将检查 config.xml 中的各个元素，了解它们是如何映射到 HelloWorld Check 的 Python 脚本和 SA 客户端显示字段的。

8 在 config.xml 文件中，注意下列与 HelloWorld Check 中的审核 (get) 和修正 (set) 脚本相关的元素：

```
<!-- The name of the script that performs the check.-->
<checkGetScriptName>gethelloworld.py</checkGetScriptName>


<!-- The name of the script that remediates the audit.-->
<checkSetScriptName>sethelloworld.py</checkSetScriptName>


<!-- The exit code of the gethelloworld.py script will be checked.-->
<checkReturnType>EXITCODE</checkReturnType>


<!-- A string argument is passed to gethelloworld.py.-->
<checkGetArgumentType>STRING</checkGetArgumentType>


<!-- The default argument for gethelloworld.py is the name of the file the
script is checking for.-->
```

```
<checkGetArgumentDefaultValue>/var/tmp/helloworld
</checkGetArgumentDefaultValue>


<!-- If the helloworld file exists, the exit code of gethelloworld.py is 0.
-->
<checkSuccessExitCodeValue>0</checkSuccessExitCodeValue>


<!-- If the helloworld file does not exist, the exit code of
gethelloworld.py is 1. -->
<checkSuccessExitCodeValue>1</checkSuccessExitCodeValue>
```

9 检查 gethelloworld.py 脚本，该脚本通过检查 /var/tmp/helloworld 是否存在来执行审核。在本教程中，无需编辑该脚本。在本教程的后面 （按照第 139 页下的步骤 29），当您在 SA 客户端中运行审核时，该脚本会在托管服务器上执行。

/var/tmp/helloworld 字符串是脚本的默认参数，由 config.xml 中 <checkGetArgumentDefaultValue> 的值表示。该脚本的退出代码 (result) 对应于为 <checkSuccessExitCodes> 指定的值。

下面是 gethelloworld.py 脚本的源代码：

```
import sys
import os
import string

if __name__ == "__main__":

    if len(sys.argv) != 2:
        sys.stderr.write("No argument found!  Please enter a
            file name!\n")
        sys.exit(220)

    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        result = 0
    else:
        result = 1

    sys.stderr.write("Debugging:  Found result %s\n"
       % result)
    sys.stdout.write("%s\n" % result)

    sys.exit(result)
```

10 然后，检查修正脚本 sethelloworld.py。该脚本将创建 /var/tmp/helloworld 文件。如果确定要在按照第 139 页下的步骤 34 中修正审核，该脚本将在托管服务器上运行。在本教程中，将不更改该脚本。

sethelloworld.py 的源代码如下：

```
import sys
import os
import string
```

```
if __name__ == "__main__":

    if len(sys.argv) != 2:
        sys.stderr.write("No argument found!
           Please enter a file name!\n")
        sys.exit(220)

    filename = sys.argv[1]
    if os.path.isfile(filename) or os.path.isdir(filename):
        # Do nothing because the file already exists.
        pass
    else:
        try:
           fd = open(filename, "w")
           fd.write(" ")
           fd.close()
        except:
            sys.stderr.write("Could not open file %s for
               writing!\n" % filename)
            sys.exit(220)

    # Exit successfully with a 0 exit code.
    sys.stderr.write("Successfully created file\n")
    sys.exit(0)
```

11 将 HelloWorld Check 打包。

要将 HelloWorld 可插入检查打包，请将工作目录内容存档到一个 ZIP 文件中，例如：

```
cd /home/jdoe/dev/helloworld
zip ../helloworld.zip *
```

12 通过输入下列 `unzip` 命令，验证 ZIP 文件是否包含两个 Python 脚本和 `config.xml` 文件：

```
unzip -t ../helloworld.zip
 testing: config.xml      OK
 testing: gethelloworld.py     OK
 testing: sethelloworld.py     OK
No errors detected in compressed data of ../helloworld.zip.
```

13 使用 OCLI 1.0 的 `oupload` 命令将可插入检查导入 SA：

```
oupload -C"Customer Independent" \
-t"Server Configuration Check" \
--forceoverwrite --old -O"SunOS 5.8" ../helloworld.zip
```

注意：对于所有 Unix 和 Linux 检查，平台选项 (`-O`) 为 `SunOS 5.8`。对于 Windows 检查，平台选项为 `Windows 2003`。

如果 `oupload` 运行失败，请确保您已安装正确版本的 OCLI 1.0、正确设置 PATH 环境变量，并已将 `login` 文件包括到环境中。有关这些要求的详细信息，请参见 《SA 用户指南：Server Automation》中的 OCLI 1.0。

14 打开 SA 客户端。

在接下来的几个步骤中，您将创建新审核，将其添加到使用 `ouplo ad` 命令导入的 HelloWorld Check 中。

15 在“工具”菜单中，选择“更新缓存”。

16 从“导航”页面中，选择“库” ➤“按类型” ➤“审核和修复”➤“审核” ➤“Unix”。

17 在“操作”菜单中，选择“新建”。

18 在“审核”窗口“属性”窗格的“名称”字段中，输入“HelloWorld Audit”。

19 在“视图”窗格中，选择“规则”➤“文件系统”。

“内容”窗格将在“可用于审核”下列出 HelloWorld Check，如图 21 所示。

图 21 文件系统规则中的 HelloWorld Check

20 在 `config.xml` 文件中，注意下列与图 21 中所示信息相关的元素：

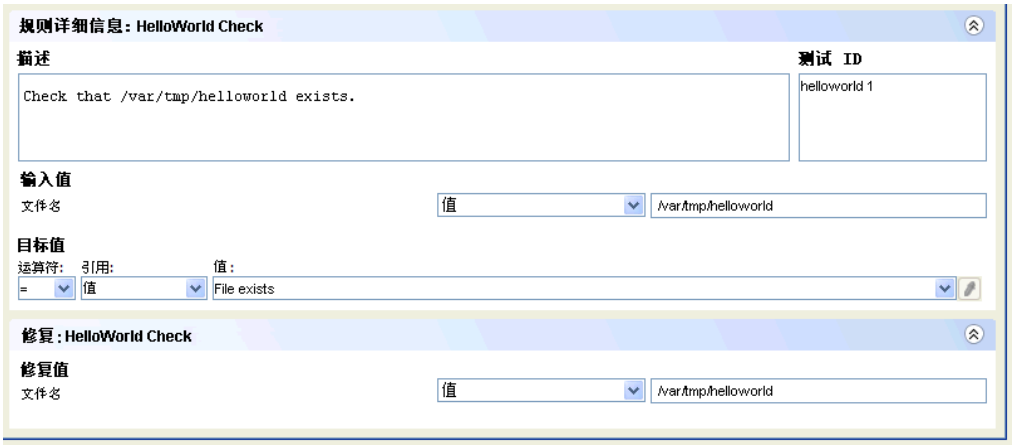
```
<!-- The check name is the rule name shown in the SA 客户端 . -->
<checkName>HelloWorld Check</checkName>


<!-- The category corresponds to the rule hierarchy displayed by the SA 客
户端 . -->
<checkCategory>File System|My Custom Checks</checkCategory>
```

21 在 SA 客户端的“审核”窗口的“可用于审核”下，选择“HelloWorld Check”，然后单击加号。

“内容”窗格将列出 HelloWorld Check 的详细信息，如图 22 所示。

图 22 HelloWorld Check 规则详细信息



22 在 config.xml 文件中，检查下列与图 22 中 “规则详细信息” 下显示的信息相关的元素：

```
<!-- The following value appears under Description in the Rule Details of
the SA 客户端 . -->
<checkDefaultDescription>
Check that /var/tmp/helloworld exists.
</checkDefaultDescription>


<!-- The following element corresponds to the Test ID in the SA 客户端 . -->
<checkTestID>helloworld 1</checkTestID>


<!-- This label is under Input Values in the SA 客户端 . -->
<checkGetArgumentDefaultLabel>File Name
</checkGetArgumentDefaultLabel>


<!-- The default argument to the gethelloworld.py script also appears
under Input Values in the SA 客户端 . -->
<checkGetArgumentDefaultValue>/var/tmp/helloworld
</checkGetArgumentDefaultValue>
```

23 在 SA 客户端的 “视图” 窗格中选择 “目标”。

在以下步骤中，将向 HelloWorld Audit 中添加目标服务器。在其后的步骤中，gethelloworld.py 和 sethelloworld.py 脚本将在目标服务器上运行。

24 在 “内容” 窗格中，单击 “添加”。

25 在 “选择服务器” 窗口中，向下钻取到服务器，然后单击 “确定”。

26 在 “审核” 窗口中，选择 “文件” ➤ “保存”。

此时， HelloWorld Audit 将包含 HelloWorld Check （规则），并与目标服务器关联。

27 在 “审核” 窗口中，从 “操作” 菜单选择 “运行审核”。

28 在“运行审核”任务的各个窗口中完成操作。

29 在“运行审核”窗口中，单击“启动作业”。

此操作将在目标服务器上启动运行 gethelloworld.py 脚本的作业。

30 在该作业完成后，单击“查看结果”。

31 在“审核结果”窗口的“视图”窗格中，选择“策略规则 (1)”。

32 在“审核结果”窗口的“内容”窗格中，打开“HelloWorld Check”。

此时将显示“差异详细信息”窗口，如图 23 所示。

图 23 HelloWorld Check 差异详细信息

33 在 config.xml 文件中，注意下列与图 23 的“差异详细信息”窗口所示信息相关的元素：

```
<!-- The following value appears as the Policy Value in the Difference
Details window.-->
<checkSuccessExitCodeDefaultDisplayName>
File exists</checkSuccessExitCodeDefaultDisplayName>


<!-- The next value appears as the Actual Value in the same window.-->
<checkSuccessExitCodeDefaultDisplayName>
File does not exist</checkSuccessExitCodeDefaultDisplayName>
```

34 如果要在目标服务器上创建 /var/tmp/helloworld，则在“差异”窗口中单击“修复”。

该操作将运行 sethelloworld.py 脚本。有关详细信息，请参见《SA 用户指南：审核与符合性》。

审核和修正概述

Sarbanes-Oxley (SoX)、IT 基础设施库 (ITIL) 和 ISO20000 要求服务器配置必须符合规定。SA 的审核和修正功能提供了一组结构清晰的策略，帮助您解决符合性问题。而通过图形界面，您可以轻松针对指定服务器选择和运行审核，以及查看服务器与各种专业标准的符合情况。

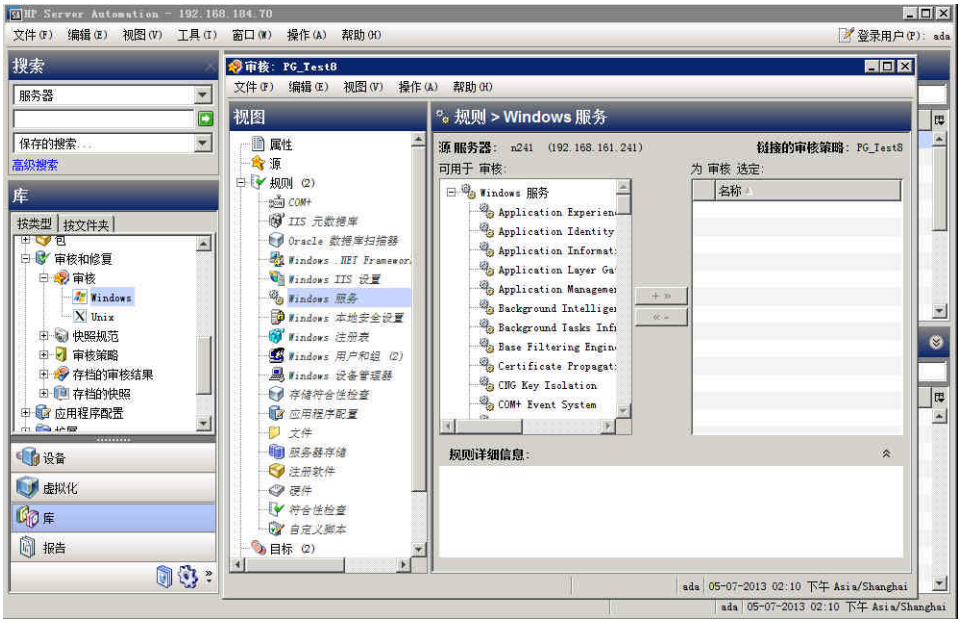
审核和修正功能还简化了系统管理任务。例如，您可以监控一类运行由自己团队所构建应用程序（例如数据库服务器或中间件应用程序）的服务器。当配置和监控运行该类应用程序的服务器时，您需要一个清单来跟踪配置的理想状态。该清单可包括文件、目录和网络共享权限。

您可以创建用来定义这些配置的审核，然后在安装应用程序后审核服务器。审核结果将确认应用程序是否已安装，以及是否已根据条件成功配置。如果配置不符合条件，可以创建临时审核，用于解决问题。如果审核结果表明发生了错误，则可以修正服务器，以实现可满足要求的配置。为了确保所做配置变更能够在生产环境中正常工作，可以将审核设置为按可配置的计划运行，并在完成时发出通知。

图 24 中显示了用于选择审核的窗口，其中包括以下标注：

- **标注 A**：在 “视图” 面板中列出的任何类别可能具有 SA 无法修改的功能，或者可修改的可插入检查。
- **标注 B**：指向用于处理 Windows 服务的 SA 功能。
- **标注 C**：列出用于处理 Windows 服务的可插入检查。

图 24 Windows 服务审核规则



每项检查评估一个规则。可以将多项检查捆绑到一个策略中。

SA 审核和修正功能附带了许多预置检查。通过选择所需检查，可以运行大部分审核。随着开发人员设计、编码、测试以及通过 HP Live Network 向系统添加更多的检查，审核选项将持续增多。这些检查作为完整策略导入。

然而，由于每个业务都具有独特挑战和独有资源，因此您可能需要根据 SA 审核和修正框架中一组尚不存在的审核条件来确定符合性。为此，系统提供了一种方法来创建自定义可插入检查。

审核和修正功能将根据特定规则来评估 SA 所管理的服务器的符合性状态。此功能还可以修正与规则中定义的期望配置状态不一致的服务器。这些规则包括各种服务器参数、注册表值、文件权限、应用程序配置、文件存在性、 COM+ 对象，等等。

在 Windows 环境中， Web 服务器规则也使用应用程序配置指定，基于 Microsoft Internet Information Services (IIS) Web 服务器配置文件 UrlScan.ini。SA 可以比较指定配置文件的部分或全部值，从文件选择所需元素并确保这些值或配置文件条目存在。有关详细信息，请参见 《SA 用户指南：应用程序配置》。

SA 包含许多预先设计的审核规则。每个规则都定义了服务器或服务器组的期望配置状态。一些规则基于值，提供了比较运算符（<, >, ==, !=, contains 等）、一个或一组值以及一个或多个检查，用来拼写出评估审核的项目状态所用的基础代码。比较数据将确定是否符合条件。如果检查支持修正操作，则规则也可能包含修正值。

规则由单个检查组成。通过以可插入检查形式来使用自定义内容对象，您可以创建新功能。为方便起见，还可以将多个相关的可插入检查捆绑到审核策略中。

可插入检查创建

可插入检查是下载到一个或多个托管服务器并由审核和修正框架执行的代码。可以使用检查来扩展本地审核和修正属性，以及提供其他专用功能。每个可插入检查都包括一个自定义的 config.xml 文件以及至少一个脚本，用于对照该 config.xml 文件中指定的值来比较审核功能。可插入检查还可包括一个脚本，用于将审核的服务器中的指定变量设置为 config.xml 文件中指定的值。您可以在 Python、Visual Basic 脚本 (VBS)、 BAT 或 shell 脚本中编写可插入检查。可插入检查将打包为 zip 存档。

大多数 CIS 检查是 CIS 基准的直接转换。有关详细信息，请访问 http://www.cisecurity.org 。

大多数类型的检查可划分为以下类别之一：

- Windows 注册表检查
- Unix 服务检查

- 用户检查，这些检查可使用密码或影射文件信息

可插入检查准则

为了简化服务器维护操作，请遵循以下准则：

- 在创建新的可插入检查时，应特别留意名称。名称应描述检查目的，并用下划线代替空格。例如，Users_Without_Password_Expiration 就一目了然。在服务器已获取数百个或更多检查的情况下，这可帮助您快速找到某个检查。
- 编写通用检查。这样，只需更改几行代码就能轻松创建更多执行类型相同的检查。例如，对于大多数 CIS2k3 Windows 服务检查，只需更改一行代码即可为新服务创建新检查。
- 在命名审核 (get) 和修正 (set) 脚本时，请从目录名称中删除空格或下划线，并相应地加上前缀 get 或 set。例如， getUsersWithoutPasswordExpiration.sh 就是一个很好的审核文件名称。即使认为您的自定义检查不会由其他人使用，也请遵循该准则。
- 注意错误检查。请记住，脚本失败时，意外的返回值可能会将审核报告为不符合条件。请捕获意外的错误或异常，并将相关信息写入到 stdout 或 stderr 以简化故障排除操作。
- 在可能的情况下，将大多数检查转换为简单的二进制 True 或 False 结构。
- 务必始终尝试同时处理特定基准情形及其对应情形。例如，您可以在创建“Enable Service X”可插入检查的同时，轻松地创建 “Disable Service X”可插入检查，并重用大多数代码。如果您决定以后测试相反条件，这将十分有用。
- 尽可能使用框架所定义的标准退出代码。这些退出代码为：

  ```
  EXIT_FAILURE=220
  EXIT_ERR_USAGE=221
  EXIT_ERR_INVALID_OS=222
  ```

- 在布尔类型检查中返回已禁用或已启用时，返回 0 表示已禁用，返回 1 表示已启用。
- 将所有可插入检查打包为 ZIP 存档。单个文件系统目录中包含表 23 所列出的文件。

表 23　可插入检查内容

文件名	描述
config.xml	（必需）用于定义该可插入检查如何执行、返回及最终报告符合或不符合条件的 XML 配置文件。

表 23　可插入检查内容（续）

文件名	描述
get*Name*. {py | sh | BAT | vbs}	（必需）以 Python、VBS、BAT 或 shell 编写的审核脚本，用于评估审核对象，并根据 config.xml 中的定义返回文本和退出代码。
setName. {py | sh | BAT | vbs}	（可选）以 Python、VBS、BAT 或 shell 编写的修正脚本，用于修正审核脚本所检查的条件。
其他代码、脚本或库	（可选）由审核或修正脚本使用的帮助程序和补充脚本。

审核和修正脚本的文件名不必以 get 和 set 开头，但是该命名约定可简化文件维护工作。

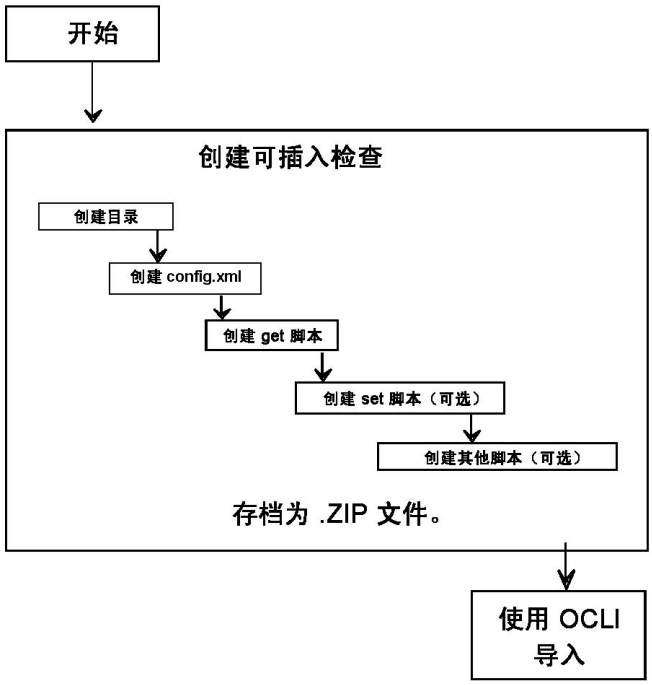
以下示例显示了可插入检查的目录结构：

```
./check_name/
./check_name/config.xml
./check_name/getcheckname.py
./check_name/setcheckname.py
```

可插入检查的开发流程

图 25 概述了在命令行环境中执行的开发流程。

图 25　开发流程



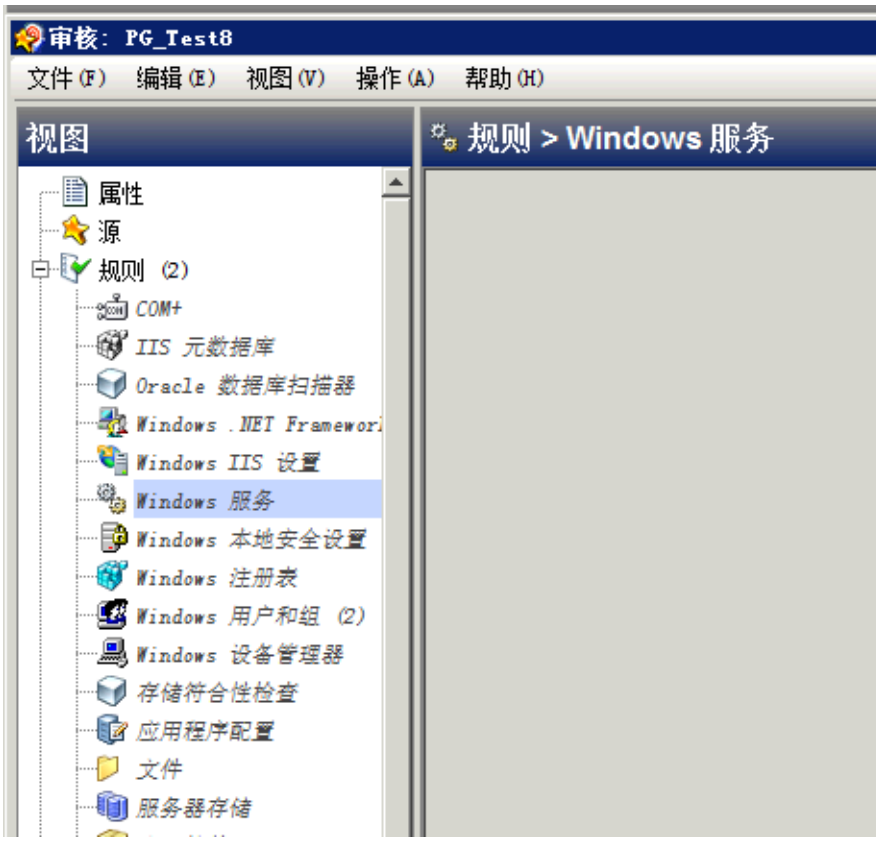
可插入检查配置 (config.xml)

config.xml 文件是可插入检查的规范文件，其中包含一些元素，用于控制该检查在 SA 客户端中的显示方式、默认值、要比较的值类型以及检查类别。例如，config.xml 文件中的以下元素确定 SA 客户端中的可插入检查的规则类别：

```
<checkCategory>Windows Services</checkCategory>
```

每个标准类别由其相应图标表示。标准类别包括硬件、软件、操作系统、用户和组、文件系统等，如图 26 所示。

图 26 规则层次结构中的可插入检查类别



以下列表显示了 config.xml 文件的模板：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE checkConfiguration SYSTEM "check.dtd">
<checkConfiguration version="1.0">
<checkName>$CHECKNAME</checkName>
<checkGUID>$CHECKGUID</checkGUID>
<checkDefaultDescription>$CHECKDESCRIPTION</checkDefaultDescription>
<checkRemediationDefaultDescription> $CHECKREMEDIATIONDESCRIPTION </
checkRemediationDefaultDescription>
<checkGetScriptName>$GETSCRIPTNAME</checkGetScriptName>
<checkGetScriptType>PY</checkGetScriptType><!-- Or SH for shell, BAT for Bat,
VBS for Visual Basic -->
<checkSetScriptName>$SETSCRIPTNAME</checkSetScriptName><!-- Optional -->
<checkSetScriptType>PY</checkSetScriptType><!-- Optional -->
<checkVersion>32b.0-1.0</checkVersion>
```

```
<checkReturnType>$RETURNTYPE</checkReturnType>   <!-- EXITCODE, STRING, or
NUMBER -->
<checkTestIDs>
<checkTestID>$CHECKTESTID</checkTestID> <!-- Optional -->
</checkTestIDs>
<checkPlatformTypes>
<checkPlatform>$PLATFORMTYPE</checkPlatform>  <!-- Currently Unix or Windows --
>
</checkPlatformTypes>
<checkCategories>
<checkCategory>$CATEGORY</checkCategory>  <!-- Top-level GUI category -->
</checkCategories>
<checkGetArguments>  <!-- All arguments are optional -->
<checkGetArgument>
<checkGetArgumentType>$GETARGTYPE</checkGetArgumentType> <!-- STRING or NUMBER
-->
     <checkGetArgumentDefaultLabel>$GETDEFAULTLABEL</
checkGetArgumentDefaultLabel>
            <checkGetArgumentDefaultDescription>$GETDEFAULTDESCRIPTION</
checkGetArgumentDefaultDescription>
          <checkGetArgumentDefaultValue>$GETDEFAULTVALUE</
checkGetArgumentDefaultValue>
     </checkGetArgument>
</checkGetArguments>
<checkSetArguments>  <!-- Also optional -->
<checkSetArgument>
<checkSetArgumentType>$SETARGTYPE</checkSetArgumentType>
     <checkSetArgumentDefaultLabel>$SETDEFAULTLABEL</
checkSetArgumentDefaultLabel>
            <checkSetArgumentDefaultDescription>$SETDEFAULTDESCRIPTION</
checkSetArgumentDefaultDescription>
     <checkSetArgumentDefaultValue>$SETDEFAULTVALUE</
checkSetArgumentDefaultValue>
</checkSetArgument>
</checkSetArguments>
<checkSuccessExitCodes>  <!-- Only for EXITCODE type checks, generally at least
two entries -->
     <checkSuccessExitCode>
<checkSuccessExitCodeValue>$EXITCODEVALUE</checkSuccessExitCodeValue>
            <checkSuccessExitCodeDefaultDescription>$EXITCODEDESCRIPTION
     </checkSuccessExitCodeDefaultDescription>
            <checkSuccessExitCodeDefaultDisplayName>$EXITCODEDISPLAYNAME
     </checkSuccessExitCodeDefaultDisplayName>
  </checkSuccessExitCode>
</checkSuccessExitCodes>
</checkConfiguration>
```

有关更多详细信息，请参见 config.xml 文件的文档类型定义 (DTD)（第 149 页）。

审核 (get) 脚本

您可以设计审核脚本（也称为 get 脚本），以从托管服务器中获取值。根据 config.xml 文件中的指定，可使用一些可选参数来执行该脚本。如果脚本运行 EXITCODE 检查，则会将脚本结果与 config.xml 文件中指定的退出代码进行比较。对于 STRING 和 NUMBER 返回类型的检查，会将结果与写入 STDOUT 的内容进行比较。

审核脚本具有一组预定义的返回代码。可以在检查的 config.xml 文件中定义其他返回代码。

审核脚本可显示参考消息。在对审核脚本失败进行故障排除时，这些消息将很有用。请查看下面的 Python 审核脚本示例：

```
import sys
import os
import string

if __name__ == "__main__":

  # If there are get arguments they will be loaded into sys.argv

  # Enter the desired check code here
  # Example:
  #   Looking for file "/usr/bin/ssh"

  if os.path.isfile("/usr/bin/ssh"):
     result = 1
  else:
     result = 0

  # Case A:
  #  If number/string check, the results are grabbed from    #  stdout.
  #   All debugging statements must be sent to stderr so as not
  #   to be picked up.

  sys.stderr.write("Debugging:Found result %s\n" % result)

  sys.stdout.write(result)

  # Case B:
  # If exitcode check, the results are returned by the argument
  #   passed to sys.exit().The exitcodes must match the
  #   ExitCodeValues defined in the config.xml file.

sys.exit(result)
```

修正 (set) 脚本

您可以设计修正脚本 （也称为 set 脚本），以制定托管服务器更改，该更改完成时审核脚本将返回成功。根据检查的 config.xml 文件中的指定，可使用一些可选参数来执行该脚本。

这些 set 脚本是可选的，并且可以具有与其对应 get 脚本非常相似的特性，也可以具有与对应 get 脚本完全不同 （并且更长）的特性。

从 shell 的角度来看，除了所使用的返回代码之外，脚本本身并无特殊之处。大多数检查会显示一些调试输出或信息消息。除了发生脚本失败时 （这些消息可用于执行故障排除），用户通常不会看到这些信息。

标准做法是始终在 set 脚本中至少包含一个参数。此外，请记住修改 config.xml 文件，以便在向现有检查添加 set 脚本时，该文件能够正常显示在 SA 客户端中。

确保在修正脚本退出时返回退出代码 0，这表示成功。所有其他退出代码将表示修正操作失败。

请查看下面的 Python set 脚本示例。

```
import sys
import os
import string
if __name__ == "__main__":

   # If there are set arguments they will be loaded into
   # sys.argv
   # Enter the desired set code here.Stdout may be used for
   # debugging.
   # Uses exitcode 0 for success, and all other values for
   # failure.
   # enter condition where set script if successful. for this
   # example, use 'if 1'

   if 1:
      sys.exit(0)

   else:
      sys.exit(-1)
```

可插入检查中的其他代码

除 get 或 set 脚本以外，可插入检查还可包含其他代码。可以向检查中添加库、可执行文件或其他脚本，以使其 set 或 get 脚本能够在执行时利用这些代码。

您还可以在 ZIP 文件中包含其他代码。

压缩可插入检查

在创建 config.xml 文件之后，审核 (get) 脚本和可选的修正 (set) 脚本将创建一个包含这些文件的 ZIP 存档。下面的 shell 历史记录显示了 UNIX 环境中的创建过程。

```
# ls
 check_name
# cd check_name
# zip ../checkname.zip *
 adding: config.xml
 adding: getcheckname.py
 adding: setcheckname.py
# unzip -t ../checkname.zip
 testing: config.xml      OK
 testing: getcheckname.py     OK
 testing: setcheckname.py     OK
No errors detected in compressed data of ../checkname.zip.
```

导入可插入检查

可使用 OCLI 1.0 实用程序将可插入检查导入 SA 核心或网状网络。《SA 内容实用程序指南》中说明了该实用程序。下面的 shell 历史记录提供了适用于 Linux 的导入过程示例：

```
# cp checkname.zip /var/tmp/checks
# cd /var/tmp/checks
# cp opsware_32.a.692.0-upload/disk001/packages/Linux/3AS/ocli-32a.2.0.5-
linux-3AS .
# chmod 755 ocli-32a.2.0.5-linux-3AS
# ./ocli-32a.2.0.5-linux-3AS
# ../ocli/login.sh
# export PATH=/opt/opsware/bin:$PATH
# oupload -C"Customer Independent" -t"Server Configuration Check" --
forceoverwrite --old -O"SunOS 5.8" your_Pluggable_check.zip
```

`oupload` 命令使用 “SunOS 5.8” 将检查定义为属于 SA 客户端中的通用 Unix 类别。要指定 Windows 类别的检查，请使用 “Windows 2003”。

审核策略创建

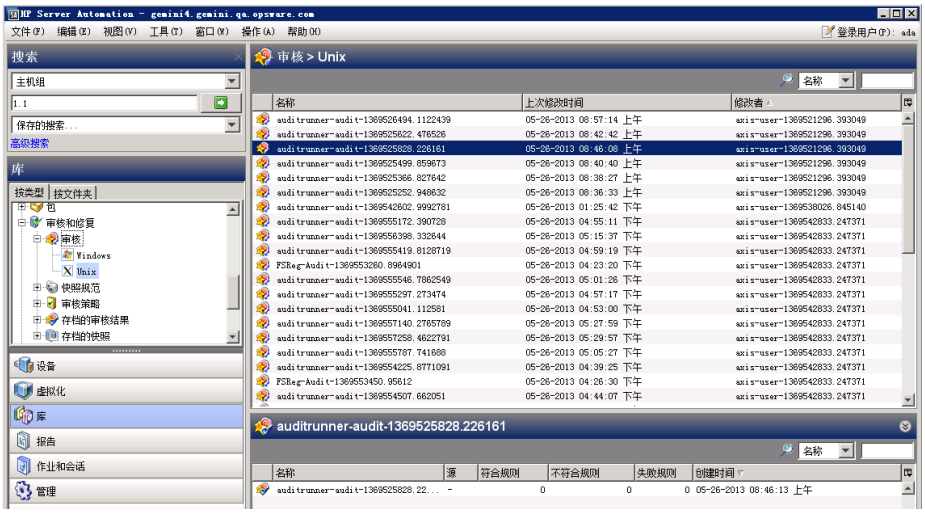
下面的图 27 显示了审核策略创建过程：

图 27 审核策略创建过程

创建审核策略

审核策略由规则组成。每个规则由一个或多个检查组成，其中可以包括用户创建的可插入检查。在 SA 客户端中显示、创建和编辑审核策略及规则。图 28 显示在模型系统上可用的审核规则的列表。

图 28 审核规则列表



有关如何创建审核策略的详细信息，请参见 《SA 用户指南：审核与符合性》。

导出审核策略

要将新审核策略移动到其他 SA 核心，请使用 DCML 交换工具 (DET) 命令行实用程序将其从一个核心导出，然后导入另一个核心。可通过此工具使用现有核心的内容 （如策略）来填充新安装的 SA 核心。有关此过程的详细说明，请参见 《SA 内容实用程序指南》。

config.xml 文件的文档类型定义 (DTD)

该文件管理 SA 客户端的显示名称和描述、默认值、要对检查代码返回的值执行的比较、显示这些值的 SA 客户端类别，等等。

默认 config.xml 文件中的两个元素 checkGetArguments 和 checkSetArguments 用于在执行时向脚本传递数据值。如果您的可编程检查不需要任何参数，请从 config.xml 文件中删除这些元素。

config.xml 的以下 DTD 由 SA 动态生成：

```
<!ELEMENT checkConfiguration (checkName, checkGUID, checkDefaultDescription,
checkRemediationDefaultDescription?, checkGetScriptName?,
checkGetScriptType?, checkSetScriptName?, checkSetScriptType?, checkVersion,
checkAllowRemediationOnFailure?, checkReturnType, checkTestIDs?,
checkPlatformTypes, checkExclusivePlatforms?, checkExcludePlatforms?,
checkCategories, checkGetArguments?, checkSetArguments?,
checkComparisonDefaults?, checkCompareValidValues?, checkSuccessExitCodes?)>
<!ATTLIST checkConfiguration version CDATA #REQUIRED>
<!ELEMENT checkName (#PCDATA)>
<!ELEMENT checkGUID (#PCDATA)>
<!ELEMENT checkDefaultDescription (#PCDATA)>
<!ELEMENT checkRemediationDefaultDescription (#PCDATA)>
<!ELEMENT checkGetScriptName (#PCDATA)>
```

```
<!ELEMENT checkGetScriptType (#PCDATA)>
<!ELEMENT checkSetScriptName (#PCDATA)>
<!ELEMENT checkSetScriptType (#PCDATA)>
<!ELEMENT checkVersion (#PCDATA)>
<!ELEMENT checkAllowRemediationOnFailure (#PCDATA)>
<!ELEMENT checkReturnType (#PCDATA)>
<!ELEMENT checkTestIDs (checkTestID+)>
<!ELEMENT checkTestID (#PCDATA)>
<!ELEMENT checkPlatformTypes (checkPlatform+)>
<!ELEMENT checkPlatform (#PCDATA)>
<!ELEMENT checkExclusivePlatforms (checkExclusivePlatform+)>
<!ELEMENT checkExclusivePlatform (#PCDATA)>
<!ELEMENT checkExcludePlatforms (checkExcludePlatform+)>
<!ELEMENT checkExcludePlatform (#PCDATA)>
<!ELEMENT checkCategories (checkCategory+)>
<!ELEMENT checkCategory (#PCDATA)>
<!ELEMENT checkGetArguments (checkGetArgument+)>
<!ELEMENT checkGetArgument (checkGetArgumentType,
checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription,
checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)>
<!ELEMENT checkGetArgumentType (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkGetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkGetArgumentValidValues (checkGetArgumentValidValue+)>
<!ELEMENT checkGetArgumentValidValue (checkGetArgumentValidValueItem,
checkGetArgumentValidValueDisplayName)>
<!ELEMENT checkGetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkGetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSetArguments (checkSetArgument+)>
<!ELEMENT checkSetArgument (checkSetArgumentType,
checkSetArgumentDefaultLabel, checkSetArgumentDefaultDescription,
checkSetArgumentDefaultValue?, checkSetArgumentValidValues?)>
<!ATTLIST checkSetArgument populateFromRule CDATA #IMPLIED>
<!ELEMENT checkSetArgumentType (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultLabel (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultDescription (#PCDATA)>
<!ELEMENT checkSetArgumentDefaultValue (#PCDATA)>
<!ELEMENT checkSetArgumentValidValues (checkSetArgumentValidValue+)>
<!ELEMENT checkSetArgumentValidValue (checkSetArgumentValidValueItem,
checkSetArgumentValidValueDisplayName)>
<!ELEMENT checkSetArgumentValidValueItem (#PCDATA)>
<!ELEMENT checkSetArgumentValidValueDisplayName (#PCDATA)>
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?,
checkComparisonDefaultValues)>
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>
<!ATTLIST checkComparisonDefaultOperator not CDATA #IMPLIED>
<!ATTLIST checkComparisonDefaultOperator caseInsensitive CDATA #IMPLIED>
<!ELEMENT checkComparisonDefaultValues (checkComparisonDefaultValue+)>
<!ELEMENT checkComparisonDefaultValue (checkComparisonDefaultValueItem,
checkComparisonDefaultValueDisplayName)>
<!ELEMENT checkComparisonDefaultValueItem (#PCDATA)>
<!ELEMENT checkComparisonDefaultValueDisplayName (#PCDATA)>
<!ELEMENT checkCompareValidValues (checkCompareValidValue+)>
```

```
<!ELEMENT checkCompareValidValue (checkCompareValidValueItem,
checkCompareValidValueDisplayName)>
<!ELEMENT checkCompareValidValueItem (#PCDATA)>
<!ELEMENT checkCompareValidValueDisplayName (#PCDATA)>
<!ELEMENT checkSuccessExitCodes (checkSuccessExitCode+)>
<!ELEMENT checkSuccessExitCode (checkSuccessExitCodeValue,
checkSuccessExitCodeDefaultDescription,
checkSuccessExitCodeDefaultDisplayName)>
<!ELEMENT checkSuccessExitCodeValue (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDescription (#PCDATA)>
<!ELEMENT checkSuccessExitCodeDefaultDisplayName (#PCDATA)>
```

下表描述了 config.xml DTD 的元素。

表 24　DTD 元素和特性

元素	特性
checkConfiguration version	设置为 1.0。仅在审核和修正框架需要时才进行更改。
checkName	检查 / 规则在 SA 客户端中的英文显示名称。
checkGUID	标准 GUID （例如 9500A4AE-EE9E-4383-87F2-BAD7DDC26C59） 可以使用“guidgen” Windows 实用程序生成、从网站下载，或通过其他方式获取。 GUID 必须唯一，否则无法将可插入检查上载到核心。在使用其唯一 GUID 上载检查之后，不能更改该 GUID。否则，在删除原始项之前，重新上载操作将失败并显示“数据库唯一约束错误”。检查由 GUID 进行唯一标识，但是在上载过程中则由其名称 （zip 文件名）标识。
checkDefaultDescription	在 SA 客户端描述框中显示。接受硬回车和 HTML。对于 HTML，需使用 < 和 > 转换 HTML 标记。
checkRemediationDefaultDescription	在 SA 客户端检查 / 规则的 “修正”部分下显示。
checkGetScriptName	get 脚本的文件名，例如 getUsersWithoutPasswordExpiration.sh。
checkGetScriptType	代码类型确定了要运行的解释程序。get 和 set 脚本可以是以下类型：SH、 VBS、 PY、 BAT。
checkSetScriptName	修正脚本的文件名。

表 24　DTD 元素和特性（续）

元素	特性
checkSetScriptType	代码类型确定了要运行的解释程序。 set （修正）脚本可以是 SH、 VBS、 PY 和 BA 类型。
checkVersion	该元素基于 SA 和框架内部版本号，例如 32b.0-1.0。
checkAllowRemediationOnFailure	某些脚本可能会在 get 阶段失败，但是您可以通过修正脚本更正该情况。这样，即使在脚本失败时，也可以执行修正。例如，如果未定义注册表项的不存在状态，您可以在 set 代码中创建和设置它。
checkReturnType	允许的值包括 EXITCODE、 STRING 或 NUMBER： EXITCODE - 通过 Wscript.Quit()、exit、return 等的标准脚本返回。 NUMBER - 审核和修正框架将从 stdout 抓取数据，并将其解释为数字类型。 STRING - 审核和修正框架将从 stdout 抓取数据，并将其解释为字符串类型。
checkTestIDs	测试 ID 的列表。
checkTestID	用于显示 CIS、MSFT、NSA 或其他策略的标准命名，例如 CIS-RHEL 8.4。这是显示在 SA 客户端中的自由格式字段，因此命名应一致，以便与 TON 内容对应。
checkPlatformTypes	检查的有效平台类型列表。
checkPlatform	WINDOWS | UNIX （或同时作为单独的元素）
checkExclusivePlatforms	专有平台的列表。目前，默认情况下审核和修正会按 Windows 或 Unix 划分内容，但是各种实际标准以及各操作系统中存在的限制和/或差异导致无法始终实现该理想状态。您可以将审核和修正限制为从 spin 检索的平台 ID 所指定的任何平台。 此参数可以引用 *SA 支持的平台* 文档中列出的一个受支持操作系统。
checkExclusivePlatform	单个平台 ID。
checkExcludePlatforms	排除的平台的列表。如果 PlatformType 声明了 UNIX，您可以提供要从 UNIX 集中排除的平台 ID （所有 Linux + 所有 Unix）。
checkExcludePlatform	单个平台 ID。

表 24　DTD 元素和特性 （续）

元素	特性
checkCategory	这是用于显示检查的 SA 客户端类别。目前，一个检查只能显示在一个类别中。如果类别不存在，将在上载检查时创建类别。应尽可能地对现有检查使用以下标准类别： 事件日志记录 文件系统 操作系统 操作系统	域控制器 （子类别） 操作系统	网络 （子类别） 注册表 服务 用户和组
checkGetArgument (checkGetArgumentType, checkGetArgumentDefaultLabel, checkGetArgumentDefaultDescription, checkGetArgumentDefaultValue?, checkGetArgumentValidValues?)>	指定 get 脚本的参数。
checkGetArgumentType	NUMBER | STRING
checkGetArgumentDefaultLabel	输入框或下拉选项旁的 SA 客户端标记。
checkGetArgumentDefaultDescription	包含详细说明信息的悬浮文本。
checkGetArgumentDefaultValue	此 get 参数的默认值。
checkGetArgumentValidValue (checkGetArgumentValidValueItem, checkGetArgumentValidValueDisplayName	checkGetArgumentValidValueItem (#PCDATA)> checkGetArgumentValidValueDisplayName (#PCDATA)>
checkGetArgumentValidValues (checkGetArgumentValidValue+)	（可选）用于限制参数，例如限制为 0/ 禁用和 1/ 启用。

表 24　DTD 元素和特性 （续）

元素	特性
checkSetArguments (checkSetArgument+)	checkSetArgument (checkSetArgumentType, checkSetArgumentDefaultLabel, checkSetArgumentDefaultDescription, checkSetArgumentDefaultValue?, checkSetArgumentValidValues?) setArgument 元素与 GetArguments 相同，但用于修正 /set 脚本 （如果存在）。 例外是： checkSetArgument populateFromRule - 指定该 set 参数默认值是否应用规则数据进行填充，这与在 config.xml 中提供任何默认值的方式形成对比。通常，该元素始终设置为 true。
checkSetArgumentType	NUMBER | STRING
checkSetArgumentDefaultLabel	输入框或下拉选项旁的 SA 客户端标记。
checkSetArgumentDefaultDescription	包含详细说明信息的悬浮文本。
checkSetArgumentDefaultValue	此 set 参数的默认值。
checkSetArgumentValidValues (checkSetArgumentValidValue+)	
checkSetArgumentValidValue (checkSetArgumentValidValue Item, checkSetArgumentValidValue DisplayName)>	checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)> checkSetArgumentValidValueItem (#PCDATA)> checkSetArgumentValidValueDisplayName (#PCDATA)>
checkSetArgumentValidValue Item	（可选）用于限制参数，例如限制为 0/ 禁用和 1/ 启用。
checkSetArgumentValidValueDisplayName	
<!ELEMENT checkComparisonDefaults (checkComparisonDefaultOperator?, checkComparisonDefaultValues)>	checkComparisonDefaultOperator not — negation of operator specified, TRUE | FALSE checkComparisonDefaultOperator caseInsensitive — only valid for STRING types.
<!ELEMENT checkComparisonDefaultOperator (#PCDATA)>	比较运算符的默认值列表。用于 TON 构建框架之外的字段或开发。
checkComparisonDefaultValues (checkComparisonDefaultValue+)	checkComparisonDefaultValue (checkComparisonDefaultValueItem, checkComparisonDefaultValueDisplayName).

表 24　DTD 元素和特性（续）

元素	特性
checkComparisonDefaultValueIte	默认传递给代码的值。
checkComparisonDefaultValueDisplayName	值在 SA 客户端中的显示名称。
checkCompareValidValues (checkCompareValidValue+)> checkCompareValidValue (checkCompareValidValueItem, checkCompareValidValueDisplayName)> checkCompareValidValueItem (#PCDATA)> checkCompareValidValueDisplayName (#PCDATA)>	
checkSuccessExitCodes (checkSuccessExitCode+) checkSuccessExitCode (checkSuccessExitCodeValue, checkSuccessExitCodeDefaultDescription, checkSuccessExitCodeDefaultDisplayName)>	对于 EXITCODE checkReturnType，必须定义有效值才能正确执行脚本操作，这通常同时包括符合条件和不符合条件的预期值。如果返回不同于此处所指定值的任何值，将视为脚本失败，这将在 SA 客户端和报告中以不同的形式显示。
checkSuccessExitCodeValue	脚本完成值，例如 0（通常表示*已禁用*）。
checkSuccessExitCodeDefaultDescription	DisplayName/Value 的悬浮文本。
checkSuccessExitCodeDefaultDisplayName	向用户显示的值或文本，例如 “已禁用”。

A 搜索筛选器语法

筛选器语法

搜索筛选器是一个用于 findServerRefs 等方法的参数。搜索筛选器中的表达式支持您根据对象特性值获取 SA 对象 （例如服务器和文件夹）引用。搜索筛选器的正式语法如下：

```
<filter>                  ::= (<expression-junction>)+

<expression-junction>     ::= <expression-list-open> <junction>
   (<expression>)+ <expression-list-close>

<expression>              ::= <expression-open> <attribute>
   <general-delimiter> <operator> <general-delimiter>
   <value-list> <expression-close>

<attribute>               ::= <resource_field>
<vo_member>               ::= <text>
<resource_field>          ::= <text>
<value-list>              ::= (<double-quote> <text> <double-quote>)* |
(<number>)*

<text>                    ::= [a-z] [A-Z] [0-9]
<number>                  ::= [0-9] [.]

<junction>                ::= <union-junction> |
   <intersect-junction>
<union-junction>          ::= '|'
<intersect-junction       ::= '&'
<expression-list-open>    ::= '('
<expression-list-close>   ::= ')'
<expression-open>         ::= '(' | '{'
<expression-close>        ::= '(' | '}'
<general-delimiter>       ::= <whitespace>
<whitespace>              ::= ' '
<double-quote>            ::= '"'
<escape-character>        ::= '\'

<operator>                ::= <equal_to> |...| <contains_or_above>

Valid operators for the preceding line:

<equal_to>                ::= '='     | 'EQUAL_TO'
<not_equal_to>            ::= '!='    | '<>' | 'NOT_EQUAL_TO'
<in>                      ::= '='     | 'IN'
<not_in>                  ::= '!='    | '<>' | 'NOT_IN'
```

```
<greater_than>           ::= '>'    | 'GREATER_THAN'
<less_than>              ::= '<'    | 'LESS_THAN'
<greater_than_or_equal>  ::= '>='   | 'GREATER_THAN_OR_EQUAL'
<less_than_or_equal>     ::= '<='   | 'LESS_THAN_OR_EQUAL'
<begins_with>            ::= '=*'   | 'BEGINS_WITH'
<ends_with>              ::= '*='   | 'ENDS_WITH'
<contains>               ::= '*=*'  | 'CONTAINS'
<not_contains>           ::= '*<>*' | 'NOT_CONTAINS'
<in_or_below>            ::= 'IN_OR_BELOW'
<in_or_above>            ::= 'IN_OR_ABOVE'
<between>                ::= 'BETWEEN'
<not_between>            ::= 'NOT_BETWEEN'
<not_begins_with>        ::= 'NOT_BEGINS_WITH'
<not_ends_with>          ::= 'NOT_ENDS_WITH'
<is_today>               ::= 'IS_TODAY'
<is_not_today>           ::= 'IS_NOT_TODAY'
<within_last_days>       ::= 'WITHIN_LAST_DAYS'
<within_last_months>     ::= 'WITHIN_LAST_MONTHS'
<within_next_days>       ::= 'WITHIN_NEXT_DAYS'
<within_next_months>     ::= 'WITHIN_NEXT_MONTHS'
<not_within_last_days>   ::= 'NOT_WITHIN_LAST_DAYS'
<not_within_last_months> ::= 'NOT_WITHIN_LAST_MONTHS'
<not_within_next_days>   ::= 'NOT_WITHIN_NEXT_DAYS'
<not_within_next_months> ::= 'NOT_WITHIN_NEXT_MONTHS'
<contains_or_below>      ::= 'CONTAINS_OR_BELOW'
<contains_or_above>      ::= 'CONTAINS_OR_ABOVE'
```

用法备注

必须在每个表达式接合内使用相同的接合类型：

- 有效：`((x = y) & (a = y) & (x = a))`
- 无效：`((x = y) & (a = y) | (x = a))`

文本值必须包含在双引号中，但数字值没有此要求。必须用反斜杠对值内的任何双引号进行转义：

- 有效数字：`123.456`
- 有效文本：`"abc"`
- 无效文本：`abc`
- 有效文本：`"ab\"c"`
- 无效文本：`"ab"c"`
- 无效文本：`ab"c`

将与其他表达式组接合的表达式组必须包含在括号内：

- 有效分组：`((x = y) & (a = b)) | (n = r)`
- 无效分组：`(x = y) & (a = b) | (n = r)`

B 重建 Apache HTTP 服务器和 PHP

此附录描述在 SA 中如何重建 Apache HTTP 服务器和 PHP 并替换它们。 SA 中包含一个 Apache HTTP 服务器和 PHP。因此，只有在需要使用其他 Apache HTTP 服务器版本，或者需要将其他库或模块编译到 PHP 中时，才需阅读本附录。

SA 使用 Apache HTTP 服务器和 PHP 进行 Web 自动化平台扩展 (APX)。有关详细信息，请参见创建自动化平台扩展 (APX)（第 59 页）。

扩展 APX HTTP 环境

本节描述如何通过重建 Apache HTTP 服务器和 PHP 来扩展 APX HTTP 环境。

必须在所有核心升级之后执行这些任务。

如果您具有多主控网状网络，则必须对所有核心中的每个切分执行这些任务。有关切片组件捆绑包的详细信息，请参见 《SA 管理指南》。

重建 PHP

执行以下任务重建 PHP。

1 从 http://www.php.net/ 下载 PHP 源代码。

2 将源代码放置在安装了 apxproxy 的服务器上的目录下，通常为 /opt/opsware/apxproxy 目录下。

3 输入以下命令，替换版本号 （如果您下载的是不同版本的 PHP）。

```
mkdir /build ; cp php-4.4.8.tar.gz /build; cd /build
gzip -dc php-4.4.8.tar.gz | tar xvf -
cd php-4.4.8
./configure --prefix=/opt/opsware/apxphp
--with-pear=/opt/opsware/apxphp/lib/pear
--with-config-file-path=/opt/opsware/apxphp/lib
--with-apxs2=/opt/opsware/apxhttpd/bin/apxs <any other options you>
make clean
make
```

4 备份 libphp4.so 的旧副本：

```
cp /opt/opsware/apxhttpd/modules/libphp4.so /opt/opsware/apxhttpd/modules/
libphp4.so.backup
```

5 将新的 libphp4.so 文件复制到 apxhhtps 目录：

```
cp libs/libphp4.so /opt/opsware/apxhttpd/modules/libphp4.so
```

6 确保 tool.list 中存在完整的引用库：

```
ldd ./libs/libphp4.so
```

对于输出中的每个条目，确保该文件存在于 /etc/opt/opsware/ogfs/tool.list 中。

如果条目不存在，请添加它。

7 备份 apxphp 文件夹：

```
mv /opt/opsware/apxphp /opt/opsware/apxphp.orig
```

8 安装 PHP：

```
make install
```

9 重新加载和链接 OGFS，确保您添加到 /etc/opt/opsware/ogfs/tools.list 中的任何内容都显示在 OGFS 中：

```
/opt/opsware/ogfs/tools/rewink && /opt/opsware/ogfs/tools/reload
```

10 重新启动 apxproxy：

```
/etc/opt/opsware/startup/apxproxy restart
```

重建 Apache

执行以下任务重建 Apache HTTP 服务器。

1 从 http://httpd.apache.org/ 下载 Apache HTTP 服务器的源代码。

2 将源代码放置在切片组件捆绑包所在的服务器上的目录中。有关切片组件捆绑包的详细信息，请参见《SA 管理指南》。

3 输入以下命令，替换版本号（如果您下载的是不同版本的 httpd）。

```
mkdir /build; cp httpd-2.2.8.tar.gz /build; cd /build
```

```
gzip -dc httpd-2.2.8.tar.gz | tar xf -
```

```
cd httpd-2.2.8
```

```
./configure --prefix=/opt/opsware/apxhttpd <any other options you want>.
```

SA 当前使用：

```
--enable-mods-shared="actions alias auth_basic auth_digest authn_file authz_user cgi deflate dir dumpio env expires headers ident logio log_config mime negotiation rewrite userdir vhost_alias imagemap status"
```

```
--disable-dav
```

```
--with-port=8021
```

```
--with-expat=builtin
```

```
--without-pgsql
```

（*仅适用于 SunOS*）输入以下命令：

```
perl -pi -e 's/#define HAVE_GETADDRINFO 1/#undef HAVE_GETADDRINFO/g' ./
srclib/apr/include/arch/unix/apr_private.h

make
```

4 生成 apxhttp 目录的备份：

```
mv /opt/opsware/apxhttpd /opt/opsware/apxhttpd.orig
```

5 安装 Apache：

```
make install
```

6 将新文件重新加载和链接到 OGFS 中：

```
/opt/opsware/ogfs/tools/rewink && /opt/opsware/ogfs/tools/reload
```

7 模块目录中的 HTTPD 和 .so 文件可能会引用外部库。这些库必须对 OGFS 可见或在 OGFS 中闪烁。

登录 OGFS，并在 /opt/opsware/apxhttpd/bin/httpd 以及 /opt/opsware/apxhttpd/modules 中的任何 .so 上运行 LDD，确保其中列出的所有文件都存在于 OGFS 中。如果不存在，则将文件添加到 /etc/opt/opsware/ogfs/tool.list（在 OGFS 外部），然后重新运行步骤 6，直到所有文件都对 /opt/opsware/apxhttpd/bin/httpd 可用。

8 此时，必须重建 PHP。请参见重建 PHP （第 159 页）。

索引