HP Server Automation

Enterprise 版

软件版本：10.0

内容实用程序

文档发布日期：2013 年 6 月 13 日
软件发布日期：2013 年 6 月

法律声明

担保

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

此处所含信息如有更改，恕不另行通知。

受限权利声明

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

版权声明

商标声明

Adobe® 是 Adobe Systems Incorporated 的商标。

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

Microsoft®、 Windows®、 Windows® XP 是 Microsoft Corporation 在美国的注册商标。

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

UNIX® 是 The Open Group 的注册商标。

支持

请访问 HP 软件联机支持网站：

http://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

支持列表

有关完整的支持和兼容性信息，请参见相关产品发布的支持列表。可在 HP 软件联机支持网站上查找所有支持列表和产品手册，地址为：

http://h20230.www2.hp.com/sc/support_matrices.jsp

您还可以从 HP 软件联机支持产品手册网站下载此版本的 《HP Server Automation Support and Compatibility Matrix》，地址为：

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

文档更新

要检查是否有最新的更新，或者验证是否正在使用最新版本的文档，请访问：

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

需要注册 HP Passport 才能登录此站点。要注册 HP Passport ID，请单击“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。有关任何版本的列表，请参见 “文档变更说明”。

产品版本

Server Automation 有两种版本：

- Server Automation (SA) 是 Server Automation Enterprise 版。有关 Server Automation 的信息，请参见《SA Release Notes》、《SA 用户指南：Server Automation》。
- Server Automation Virtual Appliance (SAVA) 是 Server Automation Standard 版。更多有关 SAVA 所包括内容的详细信息，请参见 《SAVA Release Notes》和 《SAVA 概览》指南。

文档变更说明

下表指明了自上一个发布版本以来对此文档所做的更改。

日期	更改
2013 年 5 月 20 日	SA 10.0 随附的此文档的原始发布。

目录

1 导入和导出 SA 内容

本章的目标读者包括负责指定 SA 内容的系统管理员。用户应熟悉脚本编程和 SA 基础知识。请参见 《SA 用户指南：Server Automation》。

DCML 交换工具 (DET)

DCML （数据中心标记语言）是一种基于 XML 的语言，用来描述数据中心环境内的元素和关系。DCML 交换工具 (DET) 是用来导出和导入 SA 内容的命令。使用它可以将新安装的 SA 多主控网状网络与现有网状网络中的内容融合在一起。此工具还可用于从一个网状网络导出部分内容，并将这些内容导入其他网状网络实例。

cbt 命令

DET 只是 SA 附带的一个命令 `cbt`。有关 `cbt` 命令的详细信息，请参见 cbt 命令用法 （第 13 页）和 cbt 命令参考 （第 55 页）。

利用 `cbtperm` 命令可以设置使用 DET 的权限。有关 cbtperm 命令的详细信息，请参见 DET 权限命令 cbtperm （第 60 页）。

对于 DET 来说，*内容* 表示用户创建的 SA 服务器管理信息，其中包括以下内容类型：

- 应用程序、应用程序配置、应用程序配置模板
- 自定义扩展
- 自定义字段和自定义特性
- 客户
- 分布式脚本
- 文件夹
- 包
- 修补程序和修补程序策略
- 服务器符合性条件
- 设备组
- 用户组。

内容不包括托管环境类型信息。例如，不包括设施信息和服务器属性。

DET 与 DCML 的关系

DET 导出的内容与 DCML 的第一版公开发行的规范 DCML Framework Specification v0.11 不兼容。DCML 交换工具使用专有扩展架构来描述从 Server Automation 导出的内容。导出的 data.rdf 是一个有效的 DCML 实例文档，可由兼容的 DCML 处理器解析。

自定义字段和自定义特性

每个自定义字段都位于命名空间中。DET 只能访问（因此也只能导出）用户可见的默认命名空间中的这些对象。其他命名空间（OPSWARE 等）中的对象将不会导出。如果需要导出其他命名空间中的对象（例如，操作系统序列），则通过特定于应用程序的 API（例如，操心系统序列 API）导出。

所有自定义特性都可以导出，包括对最终用户隐藏的特性 （以 `__OPSW` 开头的键）。

对于自定义字段和特性，导入的值 （包括 Null）会覆盖现有值。

2 cbt 命令用法

导出内容

cbt 命令将您指定的内容从目标 SA 网状网络导出到 RDF/XML 文件，而该文件又可以导入到另一个 SA 网状网络。请参见导入内容 （第 37 页）。

cbt 命令位于以下目录中：

/opt/opsware/cbt/bin

导出命令为：

```
cbt -e <content_dir> -f <filter_file> -cf <target_core_config>
```

此命令及其参数表示：

- 内容目录 - 将存储所导出内容的目录的路径。如果该目录事先不存在，导出功能将会创建该目录。
- filter_file - 一组规则，告知 DET 应从目标 SA 网状网络导出的内容。有关创建此文件的信息，请参见导出筛选器 （第 14 页）。
- target_core_config - 配置文件，告知 DET 各个 SA 组件的位置，以及用来访问这些组件的身份。有关创建此文件的说明，请参见创建目标网状网络配置文件 （第 51 页）。

导出命令可以使用相同的参数运行多次，附加说明如下：

- 如果指定了筛选器， DET 将忽略内容目录中任何先前的导出，并重新启动导出过程。
- 如果导出命令指定的内容目录包含有效的导出（先前成功的导出），DET 将提示用户是否确定要覆盖。如果用户选择不覆盖， DET 将退出。

在独立网状网络中开始导出或导入过程之前，关闭命令中心核心组件以防止用户更改 SA 内容，直至该过程完成。

在多主控网状网络中，首先使用多主控工具确保控制了网状网络并且没有冲突，然后关闭网状网络中的所有命令中心以防止用户更改 SA 内容，直至该过程完成。

有关停止和重新启动命令中心核心组件的信息，请参见 《SA 管理指南》。

导出筛选器

导出筛选器是用户指定的规则，告知 DET 要导出的内容 （随后将导入这些内容）。导出筛选器用于以下内容类型：

要导出软件策略和操作系统序列之类的文件夹内容，可使用文件夹导出筛选器导出父文件夹。

示例：导出筛选器文件

DET 读取指定筛选器文件中的导出筛选器。筛选器文件使用 RDF/XML 进行编码。下面是包含单个导出筛选器规则的简单筛选器文件的示例。

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE rdf:RDF [
3.  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4.  ]>
5.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6.           xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
7.  <ApplicationFilter rdf:ID="exportAppServers">
8.    <path>/Application Servers/Package Test</path>
9.    <directive rdf:resource="&filter;Descendants" />
10. </ApplicationFilter>
11. </rdf:RDF>
```

此示例在第 1 到第 6 行中显示标准筛选器标头。这些行以及第 11 行 （标准过滤器脚注）在每个筛选器中都相同。

第 7 到第 10 行在每个筛选器中都是不同的，并代表特定的筛选器功能。

在上例中，只有一条导出筛选器规则。但筛选器可以在标准标头和脚注行之间包含任意数量的唯一筛选器。例如，此筛选器包含三条导出筛选器规则：

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE rdf:RDF [
3.  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4.  ]>
5.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6.           xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
7.  <ApplicationFilter rdf:ID="exportAppServers">
8.    <path>/Application Servers/Package Test</path>
9.    <directive rdf:resource="&filter;Descendants" />
10. </ApplicationFilter>
11. <ApplicationFilter rdf:ID="exportAppServfoo">
12.   <path>/Application Servers/Foo</path>
13.   <directive rdf:resource="&filter;Node"/>
14. </ApplicationFilter>
15. <CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
16.   <scriptName>Bulk_Password_Changes</scriptName>
17. </CustomExtensionFilter>
18. </rdf:RDF>
```

示例筛选器位于 DET 安装目录下：

```
<install_dir>/filters
```

此目录包含每种筛选器类型的示例以及 `all.rdf` 筛选器，该筛选器从 SA 网状网络导出所有已知的 SA 数据类型。

下列各节描述了每种筛选器类型及其允许的参数。通常情况下，筛选器类型会映射到可以在 SA 客户端中操作的对象类型。例如，修补程序筛选器映射到 SA 客户端修补程序对象。筛选器及其特性的命名还会映射到 SA Web 客户端的命名结构，以便筛选器创建人可以很快熟悉筛选器以及筛选器与 SA 内容的关联。

应用程序导出筛选器

应用程序导出筛选器告知 DET 要导出的应用程序节点和关联的内容。通过单击导航面板中的 “软件” 链接，然后单击 “软件” 菜单上的 “应用程序” 链接，可以在 SA Web 客户端显示以下应用程序节点。

- 应用程序服务器
- 数据库服务器
- 操作系统附加设备
- 其他应用程序
- 系统实用程序
- Web 服务器

下表描述了应用程序筛选器元素的语法：

表 1　应用程序导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定： rdf:ID="unique name"

表 2　应用程序导出筛选器嵌套元素

元素	描述
path （必需）	从软件树顶层到待导出节点的绝对路径。路径分隔符为 “/”。
directive （必需）	带有一个 rdf:resource 参数的空内容元素。参数引用以下三个常量之一： • Descendants：导出给定路径的所有子级，包括路径的叶节点。 • Node：仅导出给定节点。 • Path：导出路径上的所有节点 （不包括其他节点）。 例如，给定以下路径： /Custom Applications/A/B/C/D 并且您的路径为 /Custom Applications/A/B 如果 rdf:resource 参数为 Node，则导出节点 B。 如果 rdf:resource 参数为 Path，则导出节点 A 和 B。 如果 rdf:resource 参数为 Descendants，则导出节点 B、 C 和 D。
customerName （可选）	这个可选元素限制仅导出此客户所拥有的位于指定路径中或路径下的节点。如果指定的客户不拥有该路径指定的节点，则不导出任何内容，并且会记录警告。 有关此元素如何在筛选器文件中工作的示例，请参见 customerName 元素示例 （第 35 页）。

应用程序导出筛选器示例

仅导出 /Application Servers/Foo 节点。

```
<ApplicationFilter rdf:ID="exportAppServfoo">
  <path>/Application Servers/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</ApplicationFilter>
```

导出给定路径上的 Bar 和 Baz 节点。（请注意，不会导出堆栈根。）

```
<ApplicationFilter rdf:ID="exportDBServBarBaz">
```

```
    <path>/DBServer/Bar/Baz</path>
    <directive rdf:resource="&filter;Path"/>
  </ApplicationFilter>
```

导出 patchtool 节点及其所有子级，包括叶节点。

```
  <ApplicationFilter rdf:ID="exportSUpatchtool">
    <path>/System Utilities/patchtool</path>
    <directive rdf:resource="&filter;Descendants"/>
  </ApplicationFilter>
```

导出所有属于 Acme 客户的 Apache Web 服务器：

```
  <ApplicationFilter rdf:ID="exportAcmeAppServApache">
    <path>/Application Servers/Web Servers/Apache Web</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Acme</customerName>
  </ApplicationFilter>
```

应用程序配置导出筛选器

应用程序配置导出筛选器告知 DET 要导出的应用程序配置。应用程序配置包含一个或多个应用程序配置模板文件。因此，如果导出应用程序配置，还会导出其中的所有模板文件。

表 3　应用程序配置导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"

表 4　应用程序配置导出筛选器嵌套元素

元素	描述
configurationName （可选）	可选元素，用于指定应用程序配置的名称。如果要按名称导出特定应用程序配置，则使用此元素。
customerName （可选）	可选元素，用于指定导出与指定客户关联的所有应用程序配置。
osPlatform rdf:resource （可选）	可选元素，用于指定导出与指定操作系统关联的所有应用程序配置。

应用程序配置导出筛选器示例

导出所有应用程序配置。

```
  <ApplicationConfigurationFilter rdf:ID="getAllAppConfigs"/>
```

仅导出独立于客户并且与 SunOS 5.8 操作系统关联的应用程序配置 “iPlanet”。

```
<ApplicationConfigurationFilter rdf:ID="getSpecificAppConfigs">
   <configurationName>iPlanet</configurationName>
   <customerName>Customer Independent</customerName>
   <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</ApplicationConfigurationFilter>
```

应用程序配置模板导出筛选器

应用程序配置模板导出筛选器告知 DET 要导出的应用程序配置模板文件。

表 5　应用程序配置模板导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"

表 6　应用程序配置模板导出筛选器嵌套元素

元素	描述
configurationFileName （可选）	可选元素，用于指定应用程序配置模板的名称。如果要按名称导出特定应用程序配置模板，则使用此元素。
osPlatform rdf:resource （可选）	可选元素，用于指定导出与指定操作系统关联的所有应用程序配置。
customerName （可选）	可选元素，用于指定导出与指定客户关联的所有应用程序配置模板。

应用程序配置模板导出筛选器示例

导出所有应用程序配置模板。

```
<ApplicationConfigurationFileFilter rdf:ID="getAllAppConfigTemps"/>
```

导出独立于客户并且与 Red Hat Enterprise Linux AS 3 X86_64 操作系统关联的特定应用程序配置模板 “iplanet6.1_mimetypes.tpl”。

```
<ApplicationConfigurationFileFilter rdf:ID="getSpecificAppConfigTemp">
  <configurationFileName>iplanet6.1_mimetypes.tpl</configurationFileName>
  <customerName>Customer Independent</customerName>
```

```
<osPlatform rdf:resource="&filter;Red_Hat_Enterprise_Linux_AS_3_X86_64"/>

</ApplicationConfigurationFileFilter>
```

审核筛选器

审核筛选器告知 DET 要从 SA 核心 / 网状网络导出的审核，以便之后可以将其导入到另一个 SA 核心 / 网状网络。

表 7　审核筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 8　审核筛选器嵌套元素

元素	描述
auditPolicyName	要导出的审核策略的名称。

示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<AuditPolicyFilter rdf:ID="apf1">
  <auditPolicyName>My Audit Policy</auditPolicyName>
</AuditPolicyFilter>
</rdf:RDF>
```

自定义扩展导出筛选器

自定义扩展导出筛选器告知 DET 是导出特定自定义扩展还是所有自定义扩展。如果要导出多个自定义扩展，但不是所有自定义扩展，则为每个要导出的自定义扩展创建一个筛选器。

表 9　自定义扩展导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 10　自定义扩展导出筛选器嵌套元素

元素	描述
scriptName（可选）	可选元素，用于指定要导出的脚本。脚本名称不包括帐户前缀。如果省略此元素，则导出所有自定义扩展脚本。

自定义扩展导出筛选器示例

仅导出 Bulk_Password_Changes 自定义扩展脚本。

```
<CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
  <scriptName>Bulk_Password_Changes</scriptName>
</CustomExtensionFilter>
```

导出所有自定义扩展脚本。

```
<CustomExtensionFilter rdf:ID="exportAllCustExtScripts"/>
```

自定义字段架构导出筛选器

自定义字段架构导出筛选器告知 DET 从网状网络导出所有自定义字段定义。

表 11　自定义字段架构导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

自定义字段架构导出筛选器示例

从网状网络导出所有自定义字段定义：

```
<CustomFieldSchemaFilter rdf:ID="getCustomFieldsSchema"/>
```

客户导出筛选器

客户导出筛选器告知 DET 从网状网络导出所有客户或特定客户。

表 12　客户导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 13　客户导出筛选器嵌套元素

元素	描述
customerName （可选）	可选元素，用于指定要导出的唯一客户。

客户导出筛选器示例

从网状网络导出所有客户：

```
<CustomerFilter rdf:ID="exportAllCustomers"/>
```

从网状网络导出客户 “Acme Computers”：

```
<CustomerFilter rdf:ID="exportAcmeCustomer">
  <customerName>Acme Computers</customerName>
</CustomerFilter>
```

文件夹导出筛选器

文件夹筛选器告知 DET 导出特定文件夹，包括下列与文件夹关联的项或文件夹中包含的项：

- 应用程序配置模板
- 应用程序配置
- 特性和自定义特性
- 包含的审核策略
- 包含的 OS 构建计划
- 包含的操作系统序列
- 包含的包
- 包含的脚本
- 包含的软件策略
- 按名称引用用户组的 FolderACLs （不导出用户组）。
- 到指定文件夹的路径上所有文件夹的占位符
- 子文件夹 （可选）

表 14　文件夹导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"

表 15　文件夹导出筛选器嵌套元素

元素	描述
path（必需）	必需元素，用于指定文件夹路径。
recursive（可选）	可选元素，用于指定子文件夹的导出。

文件夹导出筛选器示例

例如，假设文件夹层次结构如下。

```
/
/A
/A/B
```

下面的示例列出了给定上面的文件夹层次结构后导出的文件夹。

导出文件夹 A：

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;No"/>
</FolderFilter>
```

导出文件夹 B：

```
<FolderFilter rdf:ID="f1">
   <path>/A/B</path>
</FolderFilter>
```

导出文件夹 A 和 B：

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

导出文件夹 A 和 B：

```
<FolderFilter rdf:ID="f1">
   <path>/</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

操作系统导出筛选器

操作系统导出筛选器告知 DET 要导出的操作系统节点或操作系统类型。

表 16　操作系统导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 17　操作系统导出筛选器嵌套元素

元素	描述
osName （可选）	SA Web 客户端中的用户分配的操作系统的名称。
osPlatform （必需）	必需的嵌套元素。这个空元素具有一个 rdf:resource 参数。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。

操作系统导出筛选器示例

导出 “7.1 for mwp” Red Hat Linux 7.1 操作系统。

```
<OSFilter rdf:ID="exportOSRHLinux71">
  <osPlatform rdf:resource="&filter;Red_Hat_Linux_7.1"/>
  <osName>7.1 for mwp</osName>
</OSFilter>Export all Solaris 5.6 operating systems.
<OSFilter rdf:ID="exportOSSun56">
  <osPlatform rdf:resource="&filter;SunOS_5.6"/>
</OSFilter>
```

包导出筛选器

包导出筛选器告知 DET 从网状网络导出所有包或指定包。将导出包含文件夹的占位符。还会导出到包含文件夹的路径上的所有文件夹的占位符。

对于 Microsoft 修复程序和服务包，即使包显示为存在于网状网络中，您要导出的 Microsoft 包的二进制文件也可能尚未上载。例如，用户可能已将 Microsoft 修补程序数据库上载到网状网络，但尚未上载包的实际二进制文件。在这种情况下，已在 SA 模型中创建包的单元记录，但没有可导出的内容。这时如果要使用包导出筛选器导出包内容，将不会导出 Microsoft 包的内容。

表 18　包导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 19　包导出筛选器嵌套元素

元素	描述
packageType （必需）	必需元素，用于指定要导出的包类型。此参数可以引用下列一种包类型： • AIX_Base_Fileset • AIX_LPP • AIX_Update_Fileset • APAR • Build_Customization_Script • HPUX_Depot • HPUX_Fileset • HPUX_Patch_Fileset • HPUX_Patch_Product • HPUX_Product • Relocatable_ZIP • RPM • Solaris_Package • Solaris_Package_Instance • Solaris_Patch • Solaris_Patch_Cluster • 未知 • Windows_Hotfix • Windows_MSI • Windows_OS_Service_Pack • Windows_ZIP （已弃用） • ZIP
packageName （可选）	可选元素，用于指定已命名的包。包的名称是 SA Web 客户端“包属性”页面的 “名称”字段中显示的名称，而不是包的文件名。
osPlatform （可选）	可选元素，用于指定已命名包的操作系统。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。
customerName （可选）	可选元素，用于指定已命名包的客户。

包导出筛选器示例

导出独立于客户且在 SunOS_5.8 操作系统上运行的所有服务器的所有 RPM 程序包：

```
<PackageFilter rdf:ID="exportCIPackages">
    <packageType rdf:resource="&filter;RPM"/>
    <customerName>Customer Independent</customerName>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</PackageFilter>
```

导出属于 Acme Computers 客户的所有服务器的 RPM 程序包 “software1.0.0-1.rpm”：

```
<PackageFilter rdf:ID="exportAcmePackages">
    <packageType rdf:resource="&filter;RPM"/>
    <packageName>software1.0.0-1.rpm</packageName>
    <customerName>Acme Computers</customerName>
</PackageFilter>
```

可重定位的 ZIP 文件可以安装到单个服务器上的不同位置。由于可重定位的 ZIP 文件的名称与其父 ZIP 文件的名称相同，所以指定一个 ZIP 文件将会导出该 ZIP 文件的所有可重定位版本。例如，假设 ZIP 文件层次结构如下：

- ZIP `hmp.zip` (SunOS 5.8)
  - 安装在 `/foo` 中的可重定位 ZIP `hmp.zip`。
  - 安装在 `/bar` 中的可重定位 ZIP `hmp.zip`。

对于上述 ZIP 文件层次结构，使用以下筛选器将导出这两个可重定位的 ZIP 文件（`/foo` 和 `/bar`）。

```
<PackageFilter rdf:ID="p1">
    <packageType rdf:resource="&filter;Relocatable_ZIP"/>
    <packageName>hmp.zip</packageName>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</PackageFilter>
```

修补程序导出筛选器

修补程序导出筛选器告知 DET 要导出的修补程序或修补程序类型。

对于 DET 2.5 之前定义的 Windows 修补程序内容，确保 Windows MBSA 修补程序定义对于源和目标网状网络是相同的，否则将不导入未定义的 Windows 修补程序。

表 20　修补程序导出筛选器参数

参数	描述
`rdf:ID`	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：`rdf:ID="unique name"`。

表 21　修补程序筛选器嵌套元素

元素	描述
patchType （必需）	必需的嵌套元素。这个空元素具有一个 rdf:resource 参数。此参数可以引用下列一种修补程序类型： • APAR • APAR_FILESET • UPDATE_FILESET • AIX_Update_Fileset • HPUX_PATCH_PRODUCT • HPUX_Patch_Product • HPUX_PATCH_FILESET • HPUX_Patch_Fileset • SOL_PATCH • Solaris_Patch • SOL_PATCH_CLUSTER • Solaris_Patch_Cluster • HOTFIX • Windows_Hotfix • SERVICE_PACK • Windows_OS_Service_Pack • PATCH_META_DATA • Microsoft_Patch_Database
patchName （可选）	可选元素，用于指定特定修补程序的名称。此名称必须是 SA Web 客户端中显示的修补程序 unit_name。
patchLocale （可选）	用于标识 Windows 修补程序语言的区域设置。非 Windows 修补程序将忽略此元素。 此元素的值示例包括 en、ja 和 ko。这些值代表英语、日语和朝鲜语。默认值是英语。有关 Windows 修补程序当前支持的区域设置列表，请参见 《SA 用户指南：为服务器安装修补程序》。

修补程序筛选器示例

导出 IY13260 APAR。

```
<PatchFilter rdf:ID="exportAPARIY13260">
  <patchName>IY13260</patchName>
  <patchType rdf:resource="&filter;APAR"/>
</PatchFilter>
```

导出所有 Solaris 修补程序。

```
<PatchFilter rdf:ID="exportSolPatches">
  <patchType rdf:resource="&filter;SOL_PATCH"/>
</PatchFilter>
```

导出用于日语区域设置的修补程序 Q123456。

```
<PatchFilter rdf:ID="pf1">
  <patchName>Q123456</patchName>
```

```
  <patchLocale>ja</patchLocale>
</PatchFilter>
```

修补程序策略导出筛选器

修补程序策略导出筛选器告知 DET 要导出的用户定义修补程序策略。（将不导出供应商推荐的策略。）

可以指定可选的嵌套元素 <patchPolicyName> 和 <osPlatform> 来筛选特定的修补程序策略。如果未指定可选的嵌套元素，将导出目标网状网络中的所有修补程序策略。

表 22　修补程序策略导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 23　修补程序策略导出筛选器嵌套元素

元素	描述
patchPolicyName	可选元素，用于指定修补程序策略的唯一名称。
osPlatform	可选元素，用于使用 rdf:resource 参数指定修补程序策略的具体操作系统。此参数可以引用下列一种操作系统： • Windows_2000 • Windows_2003 • Windows_NT_4.0

修补程序策略导出筛选器示例

从目标网状网络导出所有修补程序策略：

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
```

仅导出 Windows NT 操作系统上的修补程序策略 “BestWindowsPoliciesNT” 和 Windows 2003 操作系统上的 “BestWindowsPolicies2003”：

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
  <patchPolicyName>BestWindowsPoliciesNT</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_NT/>
</PatchPolicyFilter>

<PatchPolicyFilter rdf:ID="PatchPolicies2"/>
  <patchPolicyName>BestWindowsPolicies2003</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

导出 Windows 2003 操作系统的所有修补程序策略：

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

服务器符合性条件（审核策略）导出筛选器

服务器符合性条件导出筛选器告知 DET 要导出的审核策略。

表 24　服务器符合性条件导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"

表 25　服务器符合性条件筛选器嵌套元素

元素	描述
selectionCriteriaNam（可选）	可选元素，用于指定审核策略的名称。如果要按名称导出特定审核策略，则使用此元素。
osType rdf:resource（可选）	可选元素，用于指定导出与指定操作系统关联的所有审核策略。此元素的两个可能值为 Windows 和 Unix。 例如： <osType rdf:resource="&filter;Windows"/> 或 <osType rdf:resource="&filter;Unix"/>

服务器符合性条件导出筛选器示例

导出所有审核策略。

```
<ComplianceSelectionCriteriaFilter rdf:ID="getAllSelectionCriteria"/>
```

导出与 Windows 操作系统关联的特定审核策略 “My Audit Policy”。

```
<ComplianceSelectionCriteriaFilter rdf:ID="getSpecificSelectionCriteria">
  <selectionCriteriaName>My Audit Policy</selectionCriteriaName>
  <osType rdf:resource="&filter;Windows"/>
</ComplianceSelectionCriteriaFilter>
```

服务器 （设备）组导出筛选器

服务器组导出筛选器告知 DET 从网状网络导出指定的服务器组。

表 26　服务器组导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 27　服务器组筛选器嵌套元素

元素	描述
path （必需）	必需元素，用于指定要导出的服务器组的名称。
directive （必需）	带有一个 rdf:resource 参数的必需空内容元素。可用于指定要导出的组的内容。参数引用以下三个常量之一： • Node：仅导出路径的叶节点，但如果路径事先不存在，则沿路径创建空占位符 （名称和描述，无规则）。 • Path：导出路径上的所有组 （名称、描述和规则），但不导出子级。 • Descendants：导出给定路径的所有子级，包括路径的叶节点。 例如，给定以下路径： /Group/A/B/C/D 并且您的路径为 /Group/A/B 如果 rdf:resource 参数为 Node，则导出服务器组节点 B。 如果 rdf:resource 参数为 Path，则导出服务器组节点 A 和 B。 如果 rdf:resource 参数为 Descendants，则导出服务器组节点 B、 C 和 D。
customerName （可选）	此可选元素限制仅导出附加的服务器组节点，以便仅导出此客户拥有的附加节点。 customerName 元素不影响动态服务器组规则所引用节点的导出。

备注

- 不会导出特定于核心的信息，例如组成员资格和 “Date last used”或 “History”属性。
- 静态组也会导出；但只会导出组的名称和描述。
- 如果动态组规则引用自定义字段，自定义字段架构将仅导出单个自定义字段，而不是整个架构。
- 路径定义组是公共还是私有。因此可以通过指定路径 /Group/Public （和 Descendants 指令）导出所有公共组。
- 私有组无法导出，所以使用路径 /Group/Private 将会导致在导出过程中发生错误。
- 导入的动态服务器组可以不具有任何规则。如果源组只有类似于“Facility is C07”或“Realm is Sat02”的规则，则会发生这种情况。由于 “设施”和 “领域”特定于核心，因此不会导出这些规则。
- 另外，也不会导出引用服务器 ID 的任何规则。例如，不会导出类似于 “Server ID equals 55500001”的规则。
- 所有附加的软件策略都会导出。

服务器组导出筛选器示例

从网状网络导出所有公共服务器组：

```
<ServerGroupFilter rdf:ID="exportPubServGroups">
    <path>/Group/Public/</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServerGroupFilter>
```

导出公共服务器组 “NT Servers”，包括属于该组的所有子组：

```
<ServerGroupFilter rdf:ID="exportNTServGroups">
    <path>/Group/Public/NT Servers</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServerGroupFilter>
```

仅导出公共服务器组 “Production Web Servers”（不导出其任何子组）：

```
<ServerGroupFilter rdf:ID="exportProdWebServGroups">
    <path>/Group/Public/Production Web Servers</path>
    <directive rdf:resource="&filter;Node"/>
</ServerGroupFilter>
```

导出公共组 “Production Web Servers”及其子组 “iPlanet”，但不导出其他子组。

```
<ServerGroupFilter rdf:ID="exportProdWebServGroupsIP">
  <path>/Group/Public/Production Web Servers/iPlanet</path>
  <directive rdf:resource="&filter;Path"/>
</ServerGroupFilter>
```

服务水平导出筛选器

服务水平导出筛选器告知 DET 要导出的服务水平节点。

表 28　服务水平导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 29　服务水平导出筛选器嵌套元素

元素	描述
path （必需）	从顶层节点到待导出节点的绝对路径。路径分隔符为 “/ ”。
directive （必需）	带有一个 rdf:resource 参数的空内容元素。参数引用以下三个常量之一： • Descendants：导出给定路径的所有子级，包括路径的叶节点。 • Node：仅导出给定节点。 • Path：导出路径上的所有节点 （不包括其他节点）。 例如，给定以下路径： /Service Level/A/B/C/D 并且您的路径为 /Service Level/A/B 如果 rdf:resource 参数为 Node，则导出节点 B。 如果 rdf:resource 参数为 Path，则导出节点 A 和 B。 如果 rdf:resource 参数为 Descendants，则导出节点 B、 C 和 D。
customerName （可选）	这个可选元素限制仅导出此客户所拥有的位于指定路径中或路径下的节点。如果指定的客户不拥有该路径指定的节点，则不导出任何内容，并且会记录警告。 有关此元素如何在筛选器文件中工作的示例，请参见 customerName 元素示例 （第 35 页）。

服务水平导出示例

仅导出 /Service Level/Foo 节点。

```
<ServiceLevelFilter rdf:ID="exportServLevfoo">
  <path>/Service Level/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</ServiceLevelFilter>
```

导出给定路径上的 Bar 和 Baz 节点。请注意，不会导出堆栈根。

```
<ServiceLevelFilter rdf:ID="exportServLevBarBaz">
  <path>/ServiceLevel/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</ServiceLevelFilter>
```

导出 Gold 服务水平节点及其所有子级，包括叶节点。

```
<ServiceLevelFilter rdf:ID="exportServLevGold">
  <path>/ServiceLevel/Gold</path>
  <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

快照筛选器

快照筛选器告知 DET 要从 SA 核心 / 网状网络导出的快照，以便之后可以将其导入到另一个 SA 核心 / 网状网络。

表 30　快照筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 31　快照筛选器嵌套元素

元素	描述
snapshotResultId	要导出的快照的 ID。快照名称*不*唯一，因此必须提供 ID。打开快照浏览器后，此 ID 会显示在第一个屏幕上的用户界面中。

示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<SnapshotResultFilter rdf:ID="srf1">
  <snapshotResultId>20001</snapshotResultId>
</SnapshotResultFilter>
</rdf:RDF>
```

模板导出筛选器

模板导出筛选器告知 DET 要导出的模板节点。

表 32　模板导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"。

表 33　模板导出筛选器嵌套元素

元素	描述
path （必需）	从顶层节点到待导出节点的绝对路径。路径分隔符为 “/”。
directive （必需）	带有一个 rdf:resource 参数的空内容元素。参数引用以下三个常量之一： • Descendants：导出给定路径的所有子级，包括路径的叶节点。 • Node：仅导出给定节点。 • Path：导出路径上的所有节点 （不包括其他节点）。 例如，给定以下路径： /Templates/A/B/C/D 并且您的路径为 /Templates/A/B 如果 rdf:resource 参数为 Node，则导出节点 B。 如果 rdf:resource 参数为 Path，则导出节点 A 和 B。 如果 rdf:resource 参数为 Descendants，则导出节点 B、 C 和 D。
customerName （可选）	这个可选元素限制仅导出此客户所拥有的位于指定路径中或路径下的节点。如果指定的客户不拥有该路径指定的节点，则不导出任何内容，并且会记录警告。 有关此元素如何在筛选器文件中工作的示例，请参见 customerName 元素示例 （第 35 页）。

模板导出筛选器示例

仅导出 /Templates/Foo 节点。

```
<TemplateFilter rdf:ID="exportTemplatesfoo">
  <path>/Templates/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</TemplateFilter>
```

导出给定路径上的 Bar 和 Baz 节点。请注意，不会导出堆栈根。

```
<TemplateFilter rdf:ID="exportTemplatesBarBaz">
  <path>/Templates/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</TemplateFilter>
```

导出 Alpha 模板节点及其所有子级，包括叶节点。

```
<TemplateFilter rdf:ID="exportTemplatesAlpha">
  <path>/Templates/Alpha</path>
  <directive rdf:resource="&filter;Descendants"/>
</TemplateFilter>
```

用户组导出筛选器

用户组导出筛选器告知 DET 要导出的用户组。用户组导出时会导出每个用户组的下列信息：

- 名称
- 描述
- SA Web 客户端用户组管理页面的 “功能” 选项卡中每个功能的选中状态
- “其他” 选项卡中每项权限的选中状态
- “节点堆栈” 选项卡中每个角色类堆栈的 “读取”、“读取和写入” 以及 “无” 状态
- “客户” 选项卡中每个客户的 “读取”、“读取和写入” 以及 “无” 状态
- “设备组” 选项卡中每个服务器组的 “读取”、“读取和写入” 状态
- “客户端功能” 选项卡中每项功能的 “读取”、“读取和写入”、“无”、“是” 或 “否” 状态。

表 34　用户组导出筛选器参数

参数	描述
rdf:ID	每个筛选器都有唯一名称，在筛选器文件中使用以下格式指定：rdf:ID="unique name"

表 35　用户组导出筛选器嵌套元素

元素	描述
groupName （可选）	可选元素，用于按名称导出特定用户组。如果未指定 groupName，则导出所有用户组。

备注

- 用户和设施权限的成员资格 （如 “用户” 和 “设施” 选项卡中所示）不会导出。
- “客户” 和 “设备组” 选项卡当前分别列出所有客户和服务器组，允许设置 “读取”、“读取和写入” 以及 “无” 状态。仅导出配置了 “读取” 或 “读取和写入” 权限的客户和设备组。

用户组导出筛选器示例

从网状网络导出所有用户组：

```
<UserGroupFilter rdf:ID="exportAllUserGroups"/>
```

导出组 “SuperUsers”：

```
<UserGroupFilter rdf:ID="exportUserGroups">
  <groupName>SuperUsers</groupName>
</UserGroupFilter>
```

导出三个用户组 “AdvancedUsers”、“OpswareAdministrators”和 “BasicUsers”：

```
<UserGroupFilter rdf:ID="exportAdvUsersGroup">
  <groupName>AdvancedUsers</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportOpsUsersGroup">
  <groupName>OpswareAdministrators</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportBasicUsersGroup">
  <groupName>BasicUsers</groupName>
</UserGroupFilter>
```

customerName 元素示例

这些示例说明 customerName 元素如何用于应用程序、服务水平、模板和服务器组导出筛选器。

本节包含两个主题：

针对应用程序、服务水平和模板的 customerName 示例

给定以下节点层次结构：

```
Service Levels (owned by Customer Independent)
      A (Customer Independent)
           B (Customer Independent)
           C (Nike & Adidas)
                D (Nike)
```

- 如果您的文件指定以下筛选器定义，则导出 A、 B、 C 和 D。也就是所有客户的服务水平。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

- 如果您的文件指定以下筛选器定义，则导出 A 和 B。
  C 和 D 将被跳过。

```
ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Customer Independent</customerName>
</ServiceLevelFilter>
```

- 如果您的文件指定以下筛选器定义，则导出 C 和 D。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Nike</customerName>
</ServiceLevelFilter>
```

- 如果您的文件指定以下筛选器定义，则仅导出 C。 D 将被跳过，因为它不属于 Adidas。

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Level/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Adidas</customerName>
</ServiceLevelFilter>
```

- 如果您的文件指定以下筛选器定义，则不导出任何内容。

```
<ServiceLevelFilter rdf:ID="a1">
<path>/Service Level/A</path>
<directive rdf:resource="&filter;Descendants"/>
<customerName>Nike</customerName>
</ServiceLevelFilter>
```

针对服务器组的 customerName 示例

这些示例说明 customerName 元素如何用于服务器组筛选器。

例如，如果您的核心具有以下服务器组层次结构：

```
    Server Groups
              Public
               SG1
                          + /Application Servers/A (owned by Customer
Independent)
                            + /System Utilities/B (Nike)
                            + /Web Servers/C (Adidas)
```

- 如果您的文件指定以下筛选器定义，则导出 SG1、 A、 B 和 C。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
</ServerGroupFilter>
```

- 如果您的文件指定以下筛选器定义，则导出 SG1 和 A。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Customer Independent</customerName>
</ServerGroupFilter>
```

- 如果您的文件指定以下筛选器定义，则导出 SG1 和 B。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Nike</customerName>
</ServerGroupFilter>
```

- 如果您的文件指定以下筛选器定义，则导出服务器组 SG1。

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Acme</customerName>
</ServerGroupFilter>
```

导入内容

cbt 命令将内容导入到目标 SA 网状网络。

仅基于向前兼容性支持内容导入。也就是说，无法将内容从任何 SA 版本导入到 SA 的旧版本。

cbt 命令可执行文件位于以下目录中：

/opt/opsware/cbt/bin

导入命令为：

```
cbt -i <content_dir> -p <policy> -cf <target_core_config> --noop
```

此命令及其参数表示：

- content_dir - 包含先前所导入内容的目录。
- policy - 在目标 SA 网状网络中检测到重复对象时，DET 应使用的导入策略。请参见内容类型导入策略 （第 38 页）。
- target_core_config - 配置文件，告知 DET 各个 SA 组件的位置，以及用来访问这些组件的身份。有关创建此文件的说明，请参见配置 cbt 命令 （第 50 页）。
- --noop：以 “空运行” 模式运行导入。也就是说，不修改任何数据。而在正常运行时，将输出变更摘要。

有关所有可用参数及其含义的完整列表，请参见cbt 命令参考 （第 55 页）。

使用 DET 导入应用程序时， SA 网状网络中的关联包名称采用 “cbt” 后缀。例如：

```
openssh-3.8p1-sol8-sparc-local_cbt796213986
```

内容类型导入策略

下表显示了在发现重复对象时，在命令行中指定的策略对于每种内容类型的影响。

选项包括：

- `overwrite` - 未指定策略时的默认值。此选项的影响对于每种内容类型都不同，具体如表中所述。
- `duplicate` - 此选项的影响对于每种内容类型都不同，具体如表中所述。
- `skip` - 对于所有内容类型，指定 “**skip**”意味着在发现重复对象时，会话日志中会输入一条消息，导入继续进行。

有关所有可用参数及其含义的完整列表，请参见 cbt 命令参考 （第 55 页）。

表 36　导入每种内容类型时 DET 使用的策略

内容类型	关联的内容类型	导入策略 （覆盖）	导入策略 （复制）
应用程序	• 自定义特性 • 配置跟踪策略 • 安装顺序 • 软件列表 • 客户	内容信息覆盖目标 SA 网状网络中的现有节点，而不改变其节点 ID。内容信息在现有节点上重叠。	通过对应用程序名称应用 “cbt<random>”后缀，重命名内容信息。
应用程序配置	帐户 应用程序配置文件	以覆盖模式更新所有特性。	创建新应用程序配置并将其命名为 “Oldname-cbt<random>”
应用程序配置模板	帐户	以覆盖模式更新所有特性。	创建新应用程序配置模板并将其命名为 “Oldname-cbt-<random>”
配置跟踪策略	NA	创建并覆盖现有策略。	与 “安装顺序关系”相同。
自定义特性	NA	创建并覆盖现有密钥。结果是将导入的密钥与现有密钥合并。	与 “安装顺序关系”相同。
自定义扩展	NA	创建脚本的新版本。	与覆盖策略相同。

表 36　导入每种内容类型时 DET 使用的策略 （续）

内容类型	关联的内容类型	导入策略 （覆盖）	导入策略 （复制）
自定义字段架构	NA	显示名称是唯一更新的字段。	对重复对象不执行任何操作。
客户	NA	对重复对象不执行任何操作。	对重复对象不执行任何操作。 有关导入客户的重要信息，请参见使用增量同步多主控网状网络 （第 46 页）。
分布式脚本	NA	创建脚本的新版本。	与覆盖策略相同。
文件夹	• 包 • 软件策略 • 操作系统序列 • 客户	创建到此文件夹的路径上所有文件夹的占位符。更新此文件夹的所有特性。文件夹内容和关联客户的现有数据将被覆盖。如果指定 --folderacls，将创建到预先存在的用户组的 folderACLs。	跳过：将不复制文件夹。
安装挂接	NA	请参见 “单元”。	请参见 “单元”。
安装顺序关系	NA	创建关系并且无论如何都覆盖现有关系。	由于这在父节点的上下文中完成，并且始终都会创建父节点，因此会始终创建新关系 - 尽管会重命名。
MRL	NA	始终使用源网状网络中的名称在目标网状网络中创建 MRL。	与覆盖策略相同。

表 36　导入每种内容类型时 DET 使用的策略 （续）

内容类型	关联的内容类型	导入策略 （覆盖）	导入策略 （复制）
操作系统	• 自定义特性 • 配置跟踪策略 • 客户 • 软件列表 • 安装挂接 • MRL	内容信息覆盖目标 SA 网状网络中的现有节点，而不改变其节点 ID。内容信息在现有节点上重叠。	通过对应用程序名称应用“cbt<random>”后缀，重命名内容信息。
包	NA	包将会上载并覆盖现有包，同时会覆盖“容器”包类型：LPP、HPUX 软件仓库和 Solaris 包。如果新内容（包含的包）是旧内容的一个超集，这些包类型将被新数据覆盖。否则，DET 将恢复到现有 “重命名”模式。 如果包已在另一个文件夹中，它将作为新包导入到新文件夹中 -- 现有包将不会移动到新文件夹。	与覆盖策略相同。
修补程序	NA	上载物理修补程序包，并在软件数据库中创建包含的单元。 如果新内容 （包含的包）是旧内容的一个超集， AIX LPP 和 HPUX 软件仓库包类型将被新数据覆盖。否则，DET 将恢复到现有“重命名”模式。	与覆盖策略相同。这是因为 Server Automation 无法可靠、高效地确定软件数据库中的包是否与所上载的包等同。

表 36　导入每种内容类型时 DET 使用的策略 （续）

内容类型	关联的内容类型	导入策略 （覆盖）	导入策略 （复制）
修补程序知识 (PATCH_META_DATA)	NA	修补程序数据库将导入到 Server Automation ，并覆盖现有数据库 （如果有）。导入时创建的知识将取决于目标 SA 网状网络中的修补程序首选项设置。	与覆盖策略相同。
修补程序策略	修补程序	更新修补程序的描述和列表。	创建新修补程序策略并将其命名为 “Oldname-cbt < 随机 >”
服务器符合性条件	NA	以覆盖模式更新所有特性。	创建新服务器符合性条件并将其命名为 “Oldname-cbt<random>”
服务器 （设备）组	• 应用程序 • 软件策略 • 自定义特性 • 自定义字段架构 • 修补程序 • 服务器组 • 服务水平	更新组描述和类型。覆盖动态组规则。“if any rules are met” 和 “if all rules are met” 匹配设置将会更新以反映导出中定义的内容。自定义特性将被覆盖。修补程序、应用程序和服务水平的附件也会被覆盖。	创建新服务器组并将其命名为 “Oldname-cbt < 随机 >”。
服务水平	• 自定义特性 • 配置跟踪策略 • 客户	内容信息覆盖目标 SA 网状网络中的现有节点，而不改变其节点 ID。内容信息在现有节点上重叠。	通过对模板名称应用 “cbt<random>” 后缀，重命名内容信息。
软件列表	• 单元	创建并覆盖现有列表。	与 “安装顺序关系” 相同。

表 36　导入每种内容类型时 DET 使用的策略 （续）

内容类型	关联的内容类型	导入策略 （覆盖）	导入策略 （复制）
模板	• 自定义特性 • 客户 • 应用程序 • 修补程序 • 操作系统 • 服务水平	内容信息覆盖目标 SA 网状网络中的现有节点，而不改变其节点 ID。内容信息在现有节点上重叠。	通过对模板名称应用“cbt<random>”后缀，重命名内容信息。
单元	单元脚本	单元与物理包关联，请参见上面的 “包” 内容类型。虚拟单元始终与目标 SA 网状网络中的现有单元相关联 - 上载也属于同一导入会话的物理包时，这可能会带来负面影响。	与覆盖策略相同。
单元脚本	NA	创建并覆盖现有单元脚本。	与覆盖策略相同。
用户组	客户和服务器（设备）组	更新用户组描述。此外，将更新功能的选中状态（如 “功能” 选项卡和 “其他” 选项卡中所示）以反映导出内容。并更新客户、节点堆栈和客户端功能的 “读取” 、“读取和写入” 以及 “无” 设置以反映导出内容。另外还会更新服务器组的 “读取” 以及 “读取和写入” 设置。	创建新用户组并将其命名为 “Oldname-cbt-<random>”。

导入删除条件

如果已指定将内容标记为在导出过程中删除，则在导入时运行 `--delete` 选项将会从目标网状网络中删除所标记的项。

但在某些情况下，如果目标网状网络中标记为删除的内容正被 SA 模型的部件使用， DET 将采用“无害”方法，即重命名内容项而不删除它。或者，如果在导出时使用 `-del` 选项，但在导入时不使用 `-del` 选项，则标记为在导出时删除的内容项不会在目标网状网络中删除，而是进行重命名。

在目标网状网络中重命名内容项时，会对重命名的项使用以下命名约定：

```
<item_name>-cbtDeleted<12345>
```

例如，如果在一次 DET 运行中重命名应用程序配置 “foo”，它将会重命名为 “`foo-cbtDeleted134234`”。

表 37 描述了在导入时删除内容项必须符合的所有条件，以及将要重命名内容项的情况。如果不满足删除条件，则根据重命名约定对项进行重命名。

对于某些内容项，不存在任何限制；在为导入和导出均使用删除选项时它们将始终标记为删除。对于其他内容项，则不允许删除。

重命名找不到的对象

因为某种原因重命名内容项后（无 `-del` 或“无害”）， DET 在后续的导入中可能会找不到该内容项。原因在于该项在目标网状网络中的名称已经因为重命名而发生更改。

例如，如果应用程序配置 “foo” 在一次 DET 运行中重命名为 “foo-cbtDeleted134234”，在后续的运行中 DET 尝试查找应用程序配置 “foo” 将失败。这会导致 DET 无法再次重命名或删除应用程序配置。

重命名后会找不到的具有依赖关系的对象类型包括应用程序、应用程序配置、应用程序配置文件、符合性选择条件、自定义扩展、分布式脚本、操作系统、修补程序策略、服务器 （设备）组、服务水平、模板。

表 37　使用 -del 选项的情况下，在导入时删除内容项的条件

对象类型	删除条件
应用程序	没有附加设备。 没有子节点。 没有模板或设备组包括此节点。
应用程序配置	未被任何服务器或设备组使用。 未被任何软件策略使用。
应用程序配置文件	未被任何应用程序配置使用。
符合性选择条件	始终允许删除。
自定义扩展	绝不允许删除；始终重命名。
自定义字段架构	始终允许删除。
客户	没有应用程序、服务水平和模板节点。 没有未停用的设备。 没有包 （包括状态为 DELETED 的包）。 没有 IP 范围组。 没有文件夹。 注意：如果客户仍有包在 SA 中 （包括状态为 DELETED 的包），则无法删除该客户。当对象状态为 DELETED 时，这意味着 a) 至少在一个服务器上仍需要该包来进行修正操作 b) 卫星软件数据库缓存尚未刷新该包。如果是这样，则不会删除标记为删除的客户，而是进行重命名。
部署阶段值	没有使用该值的设备。
分布式脚本	始终允许删除。
文件夹	不包含包、软件策略和子文件夹。
操作系统	没有附加设备。 没有子节点。 没有模板或设备组包括此节点。

表 37　使用 -del 选项的情况下，在导入时删除内容项的条件 （续）

对象类型	删除条件
包	是可删除的单元类型 （如下所示） 没有 Solaris 修补程序群集或 MRL 使用该包。 没有软件策略使用该包。 如果是 ZIP 包，则没有任何可重定位的子 ZIP。 没有操作系统定义或应用程序节点使用该包。 没有软件策略使用该包。 如果是修补程序： — 没有附加到修补程序节点的设备。 — 没有模板或设备组包括修补程序节点。 — 没有修补程序策略或修补程序异常包括修补程序节点。 如果是 LPP、 HPUX 软件仓库或 Solaris 包： — 没有软件策略在使用任何子包。 — 没有操作系统定义或应用程序节点使用任何子包。 — 对于属于修补程序的任何子包： – 没有附加到修补程序节点的设备。 – 没有模板或设备组包括修补程序节点。 – 没有修补程序策略或修补程序异常包括修补程序节点。 可删除的包单元类型 *（请参见此表格后面的列表）
修补程序策略	没有附加设备。 没有附加设备组。
服务器 （设备）组	没有附加设备。 没有子节点。 未被访问控制使用。 没有动态绑定作业。
服务器使用值	没有使用该值的设备。
服务水平	没有附加设备。 没有子节点。 没有模板或设备组包括此节点。
模板	没有子节点。
用户组	始终允许删除。

* 可删除的包单元类型：

— HOTFIX
— HPUX_DEPOT

- LPP
- MRL
- MSI
- PROV_INSTALL_HOOKS
- RPM
- SERVICE_PACK
- SOL_PATCH
- SOL_PATCH_CLUSTER
- SOL_PKG
- SP_RESPONSE_FILE
- 未知
- UPDATE_ROLLUP
- WINDOWS_UTILITY
- ZIP

导入客户时的注意事项

目前 DET 不支持导出与客户关联的用户组权限，但所导出的客户与目标网状网络 （要导入客户的网状网络）中的客户同名时例外。

例如，假设在源网状网络中具有软件应用程序节点 iPlanet，并且该软件应用程序节点 iPlanet 可以读取和写入到与客户 Computing Machines 关联的所有组。其中一个与客户 Computing Machines 关联的组名为 groupA。

接下来，从源网状网络导出软件应用程序节点 iPlanet，然后将该节点导入到新的网状网络 - 该网状网络没有客户 Computing Machines。结果是 groupA 中的所有用户都无法查看目标网状网络中的软件应用程序节点 iPlanet。

但如果导入客户 Computing Machines 的网状网络中已有完全同名的客户，则所有权限在新网状网络中都不会改变，并且所有用户 groupA 都能够访问软件应用程序节点 iPlanet - 也就是说，所有与 Computing Machines 客户关联的权限 （读取和写入软件节点 iPlanet 的能力）将保持不变。

导入客户的解决方法

如果用户组无权访问对象 （例如与客户关联的服务器），则使用 SA Web 客户端重新分配权限。在执行此操作前，仅管理员用户能看到这些客户及其关联的对象。

使用增量同步多主控网状网络

本 DET 发布提供执行“增量”导出和导入的方法，可帮助您同步和更新多主控网状网络中的内容。

例如，您可以从 “源”网状网络运行常规导出，该网状网络包含您希望其他网状网络包含的所有内容。使用新选项可以仅导出已修改或删除的内容，以使目标网状网络与源网状网络保持一致。

增量导出

以下命令行选项可用于执行增量导出：

- `--baseline` （短格式：`-b`）

  指定用来与当前导出进行比较的基线导出。这要求在导出过程中指定 `--incremental` 或 `--delete`。

- `--incremental` （短格式：`-incr`）

  对于筛选器文件指定的内容，仅导出在基线后增加或修改的内容。如果未指定此选项，则导出筛选器文件指定的所有内容。必须与 `--baseline` 一起使用。

- `--delete` （短格式：`-del`）

  将筛选器文件未指定的基线中的所有内容包括在导出中，并标记 “as deleted”。如果未指定此选项，则不会将任何内容导出并标记 “as deleted”。必须与 `--baseline` 一起使用。

下面说明了在导出过程中将 `--delete` 和 `--incremental` 与 `--baseline` 结合使用的情况：

- 没有增量导出选项。

  导出筛选器文件指定的所有内容。

- `-incr`

  导出筛选器文件指定的在基线之后新建或更改的所有内容。

- `-del`

  导出筛选器文件指定的所有内容 （因为未指定 `-incr`），加上筛选器文件未指定的基线中的所有内容 （“as deleted”）。

- `-incr -del`

  导出筛选器文件指定的在基线之后新建或更改的所有内容，加上筛选器文件未指定的基线中的所有内容 （“as deleted”）。

增量导入

此命令行选项允许您执行增量导入 （如果在导出时指定了某些选项）：

- `--delete` （短格式：`-del`）

  如果在导出时一起指定了 `--baseline` 选项和 `--delete`，则在导入时使用 `--delete` 选项将会删除导出中标记为删除的对象。

  如果在导出时一起指定了 `--baseline` 选项和 `--delete`，但在导入时未使用 **--delete**，则不会删除标记为删除的项，而是进行重命名。要进一步了解某些内容永远不会删除，而是始终重命名的情况 （例如，对象在网状网络中有其他依赖关系），请参见导入删除条件 （第 43 页）。

网状网络同步使用场景

当源网状网络中的内容发生了删除和修改时，典型的增量导出和导入周期如下所示：

- 刚开始，完全导出用来导出应用程序配置内容的筛选器：

  ```
  cbt -e content/appConfig.0 -f ac_Filter.rdf -cf meshA_Config
  ```

- 将导出的内容导入到另一个网状网络：

  ```
  cbt -i content/appConfig.0 -p overwrite -cf meshB_Config
  ```

  源网状网络中的内容发生更改和删除。

- 使用 -b、 -incr 和 -del 从源网状网络中导出修改和删除的内容：

  ```
  cbt -e content/appConfig.1 -f ac_Filter.rdf -b content/appConfig.0 -incr -
  del -cf meshA_Config
  ```

- 将增量内容导入目标网状网络，更新已修改的内容并删除已删除的内容：

  ```
  cbt -i content/appConfig.1 -p overwrite -cf meshB_Config -del
  ```

- 每次要更新内容时重复步骤 4 和步骤 5，并使用最新的导出作为基线。例如，在下一轮您将：
  - 使用 -b content/appConfig.1 导出 content/appConfig.2。
  - 导入 content/appConfig.2。

内容目录

内容目录是导出的 SA 内容的永久存储。内容目录包含：

- `data.rdf` - 导出的 SA 配置内容的数据库。
- `filter.rdf` - 用户提供的和 DET 生成的筛选器的数据库。
- `blob/` - 包含所导出的软件包和脚本的目录。
- `var/` - 包含前十个导入和导出会话的日志的目录。日志名为 `cbtexport {0-9}.log` 和 `cbtimport {0-9}.log`。第 0 个日志始终为最新，第 9 个日志文件始终是十个会话日志中存在时间最久的一个。

下面是内容目录示例。

```
% ls -R
.:
blob
data.rdf
filter.rdf
var
```

```
./blob:
unitid_140270007.pkg
unitid_166510007.pkg
unitid_166540007.pkg
unitid_2090007.pkg

./var:
cbtexport0.log
cbtexport0.log.lck
cbtimport0.log
```

会话示例

cbt 命令预先配置为以根用户身份在托管服务器上执行。如果在此配置中使用，您必须提供 SA 用户名和密码来执行导出或导入。

下面是会话示例。下面的示例假定用户已被授予导入和导出权限。有关详细信息，请参见配置 cbt 命令 （第 50 页）。

下面是命令中心服务器上 csh 会话的示例。

```
% setenv JAVA_HOME <j2re 1.4.x installation>
% /opt/opsware/cbt/bin/cbt -e /tmp/foo -f \
/opt/opsware/cbt/filters/app.rdf \
--spike.username hermaime
Enter password for hermaime:********
...
```

cbt 命令位于以下目录中：

```
/opt/opsware/cbt/bin
```

安装 cbt 命令

cbt 命令预安装在 SA 核心服务器上，可以随时使用。

如果要在任何其他 Unix 服务器上使用 cbt 命令，则需要按照本节中的描述对其进行手动安装。也可通过创建软件策略，将 Cbt 自动安装到任意基于 Unix 的托管服务器上。

cbt 命令可以在任何 Unix 计算机上运行，并且具有 SA 网状网络的网络访问权。尽管 cbt 命令在 Windows 平台上不受支持，但它却支持导入和导出 Windows 内容。

要在 Unix 服务器而非 SA 核心安装 cbt 命令，请执行以下操作：

1 以 root 身份登录到要安装 cbt 命令的 Unix 服务器。该服务器必须能够访问 SA 网状网络。

2 可通过 HPSA 安装介质或在 SA 核心上获取 cbt 的存档 cbt-<version>.zip。

   a 在 SA 核心，在 SA 库下找到 cbt 存档：Opsware/Tools/CBT

   b 在 HP Server Automation 安装介质 上找到位于 packages 子目录中的 cbt-<version>.zip 文件。

3 将该 cbt 存档复制到要安装的目录。

4 解压缩此存档。

5 配置 Java 运行时环境 (JRE)：

   a 如果没有此文件，可从 www.oracle.com 下载 JRE 1.4.x、JDK 1.6.x 或更高版本，然后将其安装在您登录的服务器上。

   b 将 JAVA_HOME 环境变量设置为指向您的 Java 安装。例如，在 csh 中发出以下命令：
   ```
   % setenv JAVA_HOME <java installation>
   ```

   c 也可以将 PATH 环境变量设置为包含 cbt 安装目录：<cbt_install_directory>/bin。

6 要验证是否可以运行 cbt 命令，请输入以下命令：

   ```
   % cd <cbt_install_directory>/bin
   % ./cbt -v
   ```

   -v 选项显示命令的版本字符串。

   在某些服务器上， cbt 命令显示以下错误：

   ```
   Error occurred during initialization of VM
   Could not reserve enough space for object heap
   ```

   如果发生此错误，则编辑 cbt 脚本，将 jargs 中 -Xmx 选项的值更改为更小的值，例如：-Xmx512m。

配置 cbt 命令

本节说明如何配置 cbt 命令。 cbt 命令安装在 SA 核心服务器上的以下目录中：

```
/opt/opsware/cbt/bin
```

cbt 命令可以在任何 Unix 计算机上运行，并且具有 SA 网状网络的网络访问权。尽管 cbt 命令在 Windows 平台上不受支持，但它却支持导入和导出 Windows 内容。

cbt 命令是用 Java 编写的，并使用 OWL 和 RDF 作为其架构定义和永久存储。它使用 SA API 导入和导出 SA 内容，以提取配置和大型二进制内容，例如包和脚本。

1 要验证是否可以运行 cbt 命令，请输入以下命令：

   ```
   % cd <cbt_install_directory>/bin
   % ./cbt -v
   ```

   -v 选项显示命令的版本字符串。

   在某些服务器上， cbt 命令显示以下错误：

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

如果发生此错误，则编辑 cbt 脚本，将 jargs 中 -Xmx 选项的值更改为更小的值，例如：-Xmx512m。

2 为每个要导入或导出内容的网状网络执行以下步骤。

a 从以下位置获取 opsware-ca.crt 信任证书的副本
/var/opt/opsware/crypto/twist/opsware-ca.crt
并将其保存在 cbt 命令可以访问的位置。如果您是从安装了 SA 命令中心核心组件的服务器运行 cbt，则此步骤为可选。

b 从以下位置获取 spog.pkcs8 客户端证书的副本
/var/opt/opsware/crypto/twist/spog.pkcs8
并将其保存在 cbt 命令可以访问的位置。如果您是从安装了 SA 命令中心核心组件的服务器运行 cbt，则此步骤为可选。

c 通过 SA 管理员获取 Web 服务数据访问引擎 (twist) 用户名和密码。这是在安装 Web 服务数据访问引擎 (twist) 时设置的。

d 创建目标网状网络配置文件，其中包含访问 SA 网状网络组件所需的位置和身份信息。有关此任务的详细信息，请参见下一节。

创建目标网状网络配置文件

创建目标网状网络配置文件以简化 DET 的使用。示例默认配置文件随 DET 一起安装，位于以下位置：

```
cbt/cfg/default.properties
```

网状网络配置文件是 “键 = 值” 对文本文件，其中包含需要在 DET 命令行上提供的 SA 组件访问信息。要定义 DET 网状网络配置文件的参数，请复制该文件并将其保存到已知位置。

因为配置文件包含用户名和密码，所以请务必保证它的安全。

表 38 包含所有与 DET 配置相关的可能属性。这些属性可以在 DET 命令行上提供，也可以在配置文件中指定。

表 38 中列出的默认配置属性值假定您正在运行命令中心核心组件的 SA 网状网络上运行 DET。（因此 .host 属性才会显示 localhost 值。）另外，twist.certpaths、ssl.trustcerts 和 ssl.keypairs 假定路径在命令中心服务器上。

如果网状网络配置文件中未特别提及与配置相关的属性，则使用下面 “配置属性”表中显示的默认值。

表 38　配置属性

属性名称	默认值	描述
cbt.numthreads	1	用于导出的并发线程数。 在导出内容时，您可以指定任意数量的线程。 但在导入内容时， DET 仅支持一个线程。
spike.enabled	true	使用 Spike 在所有基于 XML-RPC 的服务器上进行身份验证和授权。
spike.host	way	Spike 的主机名或 IP。
spike.path	wayrpc.py	Spike 的基本 URL 路径。
spike.password	< 无默认值 >	用于 Spike 身份验证的用户密码。这是 OCC 用户的密码，在安装网状网络期间设置。请联系 SA 管理员 （或网状网络的安装人员）了解此信息。
spike.port	1018	Spike 的侦听器端口。
spike.protocol	https	Spike 的侦听器协议。通常是 HTTPS。
spike.username	admin	用于 Spike 身份验证的用户密码。这是由 cbtperm 工具授权的用户。 此用户名必须是有权创建或修改对象的 admin 帐户，但不能是 detuser 帐户。 DET 默认配置将 spike.username 设置为帐户：admin
spin.host	spin	数据访问引擎的主机名或 IP。
spin.path	spinrpc.py	数据访问引擎的基本 URL 路径。
spin.port	1004	数据访问引擎的侦听器端口。

表 38　配置属性（续）

属性名称	默认值	描述
spin.protocol	http	数据访问引擎的侦听器协议。如果 DET 与 SA 命令中心在同一个服务器上，并且正在多服务器网状网络中运行纯文本 spin，则为 HTTP，若是采用其他配置，则为 HTTPS。
ssl.keyPairs	/var/opt/opsware/crypto/twist/spog.pkcs8	用来与基于 XML-RPC 的服务器通信的客户端证书的逗号分隔列表。
ssl.trustCerts	/var/opt/opsware/crypto/twist/opsware-ca.crt	用来与基于 XML-RPC 的服务器通信的信任证书文件的逗号分隔列表。
ssl.useHttpClient	true	使用 HTTP 客户端库而不是 JDK 的内置 HTTP 客户端。
twist.certPaths	/var/opt/opsware/crypto/twist/opsware-ca.crt	用来与 Web 服务数据访问引擎通信的信任证书的逗号分隔列表。
twist.host	localhost	Web 服务数据访问引擎的主机名或 IP。
twist.password	< 无默认值 >	Web 服务数据访问引擎的密码。此密码在安装网状网络期间设置。请联系 SA 管理员（或网状网络的安装人员）了解此信息。
twist.port	1032	Web 服务数据访问引擎的侦听端口。
twist.protocol	t3s	Web 服务数据访问引擎的协议。该协议应为 t3 或 t3s。
twist.username	detuser	Web 服务数据访问引擎的用户名。应为“detuser”。此帐户为系统帐户，密码在安装网状网络时设置。
way.host	way	命令引擎的主机名或 IP。
way.path	wayrpc.py	命令引擎的基本 URL 路径。
way.port	1018	命令引擎的侦听器端口。

表 38　配置属性（续）

属性名称	默认值	描述
way.protocol	https	命令引擎的侦听器协议。通常是 HTTPS。
word.host	word	软件数据库的主机名或 IP。从 SA 7.80 开始，软件数据库已成为切分组件捆绑包的一部分。
word.path	wordbot-new.py	软件数据库的基本 URL 路径。
word.port	1003	软件数据库的侦听器端口。
word.protocol	https	软件数据库的侦听器协议。该协议为 HTTPS。
mail.transport.protocol	smtp	用于邮件服务器的邮件传输协议。
mail.smtp.host	smtp	邮件服务器主机名。
mail.smtp.port	25	邮件服务器使用的端口号。
mail.from	<currentuser>@<currenthost>	用于通知电子邮件中的“发件人”字段的电子邮件地址。

下面是目标网状网络配置文件的示例，其中仅包含基本网状网络配置信息。

```
twist.host=twist.c07.dev.opsware.com
twist.port=1032
twist.protocol=t3s
twist.username=<detuser>
twist.password=<twist_password>
twist.certPaths=<absolute path to opsware-ca.crt>

spike.username=<OCC_user>
spike.password=<OCC_user_password>
spike.host=way.c07.dev.opsware.com
way.host=way.c07.dev.opsware.com
spin.host=spin.c07.dev.opsware.com
word.host=theword.c07.dev.opsware.com

ssl.keyPairs=<absolute path to spog.pkcs8>
ssl.trustCerts=<absolute path to opsware-ca.crt>

mail.transport.protocol=smtp
mail.smtp.host=mail
mail.smtp.port=44
mail.from=joe_user@yourcompany.com
```

3 cbt 命令参考

本章描述了 cbt 命令及其选项。 cbt 命令位于 SA 核心服务器上的以下目录中：

/opt/opsware/cbt/bin

导出选项 (-e)

导出选项使用以下语法：

```
cbt -e <content_dir> [<options>]
```

表 39 导出选项

短选项	长选项	描述
-e <content_dir>	--export <content_dir>	从 SA 核心导出 SA 数据，并将数据存储在给定内容目录中。
-f <filter_file>	--filter <filter_file>	首次导出时，必须指定描述导出数据的筛选器文件。此后，如果未指定筛选器，则使用内容目录中先前使用过的筛选器。有关 DET 筛选器文件的详细信息，请参见示例：导出筛选器文件 （第 14 页）。
-b <content_dir>	--baseline <content_dir>	指定用来与当前导出进行比较的基线导出。它要求在导出过程中指定 --incremental 或 --delete。
-incr	--incremental	执行增量导出。对于筛选器文件指定的内容，仅导出在基线后增加或修改的内容。如果未指定此选项，则导出筛选器文件指定的所有内容。

表 39　导出选项（续）

短选项	长选项	描述
-cf <file>	--config <file>	指定 DET 配置文件。有关详细信息，请参见创建目标网状网络配置文件（第 51 页）。
-c	--clean	从 -e 指定的内容目录删除先前导出的数据。
-d	--debug	显示更详细的调试信息。
-del	--delete	将筛选器文件未指定的基线中的所有内容包括在导出中，并标记 “as deleted” 。如果未指定此选项，则不会将任何内容导出并标记 “as deleted”。 如果使用此选项，还必须使用 --baseline 指定基线导出。
-np	--noprogress	不在控制台上显示进度。
-nd	--nodownload	不从软件数据库下载单元（单词）。重要提示：使用此选项导出的内容无法导入。
-lx	--logxml	创建 XML 格式的日志文件。
-em	--email	将导出摘要通过电子邮件发送到这个以逗号分隔的地址列表。要使此选项生效，必须已将电子邮件通知参数添加到 DET 配置文件中。
(none)	--emaillog	在电子邮件中包括整个日志文件。

导入选项 (-i)

导入选项使用以下语法：

```
cbt -i <content_dir> [<options>]
```

表 40　导入选项

短选项	长选项	描述
`-i <content_dir>`	`--import <content_dir>`	从给定内容目录导入 SA 数据。
`-p overwrite|duplicate|skip`	`--policy overwrite|duplicate|skip`	导入策略。默认值为“overwrite”。 “overwrite”表示覆盖目标 Server Automation 上同一命名空间中的对象，而不影响其对象 ID。 “duplicate”表示在目标 Server Automation 上检测到重复项时，使用组合名称创建对象的副本。 “skip”是最保守的策略。如果在目标 Server Automation 上检测到相同的对象，它会中止导入该对象。 有关具体内容类型的导入策略的详细信息，请参见内容类型导入策略（第 38 页）。
`-del`	`--delete`	删除导出时标记为删除的对象。也就是说，只有在导出过程中指定 `-del` 选项的情况下，此选项才生效。如果未指定此选项，将重命名对象。

表 40　导入选项（续）

短选项	长选项	描述
-fa	--folderacls	将导入的文件夹与现有用户组关联。 如果未指定此选项，请将文件夹以及从父文件夹继承的 ACL 导入到目标网状网络中。 如果指定了此选项，DET 将会尝试在导入文件夹时导入 ACL。仅当源网状网络中已存在同名用户组或该用户组已在当前 DET 运行过程中导入时，ACL 才会导入。ACL 将与同名的现有用户组进行关联。插入文件夹时，导入的 ACL 将替换目标网状网络中从父文件夹继承的 ACL。更新文件夹时，ACL 将覆盖现有 ACL。
-n	--noop	以“空运行”模式运行导入。也就是说，不修改任何数据。而在正常运行时，将输出变更摘要。
-cf <file>	--config <file>	从给定文件读取配置。
-d	--debug	显示更详细的调试信息。
-np	--noprogress	不在控制台上显示进度。
-nu	--noupload	不将未更改的包上载到软件数据库（单词）。 实用程序报告包已被覆盖，但包并未发生任何更改。仅更新了它的单元记录。

表 40　导入选项（续）

短选项	长选项	描述
-lx	--logxml	创建 XML 格式的日志文件。
-em <addrs>	--email <addrs>	将导入摘要通过电子邮件发送到这个以逗号分隔的地址列表。要使此选项生效，必须已将电子邮件通知参数添加到 DET 配置文件中。
(none)	--emaillog	在电子邮件中包括整个日志文件。

显示导出状态选项 (-t)

显示导出状态选项使用以下语法：

```
cbt -t <content_dir>
```

表 41　显示导出状态选项

短选项	长选项	描述
-t	--showstatus	显示给定内容目录的导出状态。

配置文件选项 (-s)

配置文件选项使用以下语法：

```
cbt -s [-cf <file>]
```

表 42　配置文件选项

短选项	长选项	描述
-s	--showconfig	显示当前配置值。
-cf <file>	--confi <file>	从给定文件读取配置。

显示版本选项 (-v)

显示版本选项使用以下语法：

```
cbt -v
```

表 43　显示版本选项

短选项	长选项	描述
-v	--version	显示 DET 工具的版本。

显示帮助选项 (-h)

显示帮助选项使用以下语法：

```
cbt -h
```

表 44　显示帮助选项

短选项	长选项	描述
-h	--help	显示此帮助消息。

DET 权限命令 cbtperm

利用 cbtperm 命令可以设置使用 DET 的权限。 cbtperm 权限命令使用以下语法：

```
cbtperm -u [user] -a [spike.username] -p [spike.port] -s [spike.host] -c
[ssl.trustCerts] -k [ssl.keyPairs]
```

表 45　DET 权限命令选项

短选项	长选项	描述
-u	N/A	您要向其授予 DCML 交换工具使用权限的用户。
-a	--spike.username	Spike 身份验证的用户名，例如 SA Administrator。
-p	--spike.port	Spike 的侦听器端口。

表 45　DET 权限命令选项 （续）

短选项	长选项	描述
-s	--spike.port	Spike 的主机名或 IP。
-c	--ssl.trustCerts	用来与 XML-RPC 服务器通信的信任证书文件的逗号分隔列表。
-k	--ssl.keyPairs	用来与 XML-RPC 服务器通信的客户端证书的逗号分隔列表。

4 IDK 概述

IDK 和 ISM 概述

Server Automation 包括智能软件模块 (ISM) 开发套件 (IDK)。 IDK 包含用于创建、构建和上载 ISM 的命令行工具和库。 ISM 是一组文件和目录，其中包含应用程序位、安装脚本和控制脚本。您在本地文件系统中构建一个 ISM，然后将该 ISM 上载到 Server Automation 应用程序策略。上载 ISM 后，可以使用 HP Server Automation 客户端将 ISM 的应用程序安装到托管服务器上。

IDK 的优势

IDK 具有以下优势：

- 封装了最佳实践，用于管理软件产品，支持标准团队在复杂的数据中心环境中提供稳定、一致的软件内部版本并管理变更。
- 将模块上载到 Server Automation 中，使其可以立即用于托管服务器上的安装。
- 将应用程序的安装脚本和控制脚本与要安装的位分开。更新脚本时不必重新安装应用程序位。
- 通过向 Server Automation 查询自定义特性，支持动态配置。
- 自动根据二进制存档构建本地包 （例如 RPM）。
- 在 Unix 平台上支持从采用常见规范格式的源代码构建。
- 为喜欢在 shell 环境中构建包和编写安装脚本的开发人员和管理员提供命令行工具。

IDK 工具和环境

IDK 包含以下内容：

- ISMTool - 用于创建、构建和上载 ISM 的命令行工具。
- ISMUserTool - 用于指定允许上载 ISM 的用户的命令行工具。
- 环境变量 - ISMTool 访问的 Shell 环境变量。
- 运行时库 - 支持 IDK 工具的 Server Automation 例程。

支持的包类型

您可以使用 IDK 创建以下类型的包：

- AIX LPP
- HP-UX 软件仓库
- RPM
- Solaris 包
- Windows MSI
- ZIP （Windows 和 Unix）

安装 IDK

建议您在托管服务器 （运行服务器代理的服务器）上安装和运行 IDK。有关说明，请参见在托管服务器上安装 IDK （第 64 页）。有关服务器代理的详细信息，请参见 《SA 用户指南：Server Automation》。

可以在核心服务器上安装 IDK，但要小心执行。核心组件与 IDK 工具共享 CRYPTO_PATH 环境变量。如果 CRYPTO_PATH 环境变量设置不正确，核心组件可能会停止运行。

可以在非托管服务器 （不运行核心组件或代理的服务器）上安装 IDK，但 IDK 的功能将受限。在此类服务器上，可以构建 ISM，但不能将其上载到核心，除非设置了 CRYPTO_PATH 环境变量。有关此变量的信息，请参见 CRYPTO_PATH （第 114 页）。请参见在非托管服务器上安装 IDK （第 65 页）。

在托管服务器上安装 IDK

要在托管服务器上安装 IDK 和 ISMTool，请执行以下步骤：

1 选择要运行 IDK 的托管服务器。

2 确保安装 IDK 的主机与将安装 ISM 应用程序的托管服务器运行相同版本的操作系统。

例如，如果要为将安装在 Redhat Linux 7.3 托管服务器上的应用程序创建 ISM，则在 Redhat Linux 7.3 系统上安装 IDK。

3 如果在 Windows 托管服务器上安装 IDK，则如 CRYPTO_PATH （第 114 页）中所述设置 CRYPTO_PATH 环境变量。

4 如果要在 Redhat Linux Application Server、 Enterprise Server 或 Workstation 上安装 IDK，则确保已安装 `rpm-build` 包。要验证是否已安装此包，请输入以下命令：

```
rpm -qa | grep rpm-build
```

5 如果要在 Solaris 区域上安装 IDK，则确保 `/usr/local` 目录已存在，并且具有写入权限。（此目录在稀疏根区域中可能不存在。）您可以使用 Server Automation 或下列 `zonecfg` 命令执行此任务，其中 *path* 是全局区域上的文件系统：

```
zonecfg:zone-name:fs> add fs
```

```
zonecfg:zone-name:fs> set dir=/usr/local
zonecfg:zone-name:fs> set special=path
zonecfg:zone-name:fs> set type=lofs
```

6 在 SA 客户端中搜索名称包含 “ismtool” 的软件策略。

7 在显示的软件策略列表中，右键单击适用于将运行 IDK 的平台的策略，然后选择 “附加服务器”。

8 在 “附加服务器” 窗口中，选择将运行 IDK 的托管服务器。

9 确保选中 “立即修正服务器” 复选框。

10 单击 “附加”。

11 Unix：在终端窗口中，以 root 身份登录到将安装 IDK 的主机，并将 PATH 环境变量设置为包括以下值。

```
/usr/local/ismtool/bin
```

（在 Windows 上会自动设置 PATH，但直到您再次登录后才会生效。）

12 在终端窗口中，通过输入以下命令检查 IDK 安装：

```
ismtool --myversion
```

在非托管服务器上安装 IDK

建议您在托管服务器（运行服务器代理的服务器）上安装和运行 IDK。有关说明，请参见在托管服务器上安装 IDK （第 64 页）。

可以在非托管服务器 （不运行核心组件或代理的服务器）上安装 IDK，但 IDK 的功能将受限。在此类服务器上，可以构建 ISM，但不能将其上载到核心，除非设置了 CRYPTO_PATH 环境变量。有关此变量的信息，请参见 CRYPTO_PATH （第 114 页）。

要在非托管服务器上安装 IDK 和 ISMTool，请执行以下步骤：

1 选择要运行 IDK 的托管服务器。

2 确保安装 IDK 的主机与将安装 ISM 应用程序的托管服务器运行相同版本的操作系统。

例如，如果要为将安装在 Redhat Linux 7.3 托管服务器上的应用程序创建 ISM，则在 Redhat Linux 7.3 系统上安装 IDK。

3 如果在 Windows 托管服务器上安装 IDK，则如 CRYPTO_PATH （第 114 页）中所述设置 CRYPTO_PATH 环境变量。

4 如果要在 Redhat Linux Application Server、Enterprise Server 或 Workstation 上安装 IDK，则确保已安装 rpm-build 包。要验证是否已安装此包，请输入以下命令：

```
rpm -qa | grep rpm-build
```

5 如果要在 Solaris 区域上安装 IDK，则确保 /usr/local 目录已存在，并且具有写入权限。（此目录在稀疏根区域中可能不存在。）您可以使用 Server Automation 或下列 zonecfg 命令执行此任务，其中 *path* 是全局区域上的文件系统：

```
zonecfg:zone-name:fs> add fs
zonecfg:zone-name:fs> set dir=/usr/local
zonecfg:zone-name:fs> set special=path
```

```
zonecfg:zone-name:fs> set type=lofs
```

6 在 SA 客户端中搜索名称包含 “ismtool” 的软件策略。

7 找到与目标服务器平台匹配的策略并将其打开。

8 选择 “策略项目” 视图，查看策略中的所有包。

9 找到与目标服务器的操作系统和体系结构匹配的包并将其打开。

10 使用 “操作” > “导出软件” 菜单将包下载到目标服务器。

11 将包手动安装到目标服务器。

12 Unix：在终端窗口中，以 `root` 身份登录到将安装 IDK 的主机，并将 PATH 环境变量设置为包括以下值。

```
/usr/local/ismtool/bin
```

（在 Windows 上会自动设置 PATH，但直到您再次登录后才会生效。）

13 在终端窗口中，通过输入以下命令检查 IDK 安装：

```
ismtool --myversion
```

IDK 快速入门

本节说明如何创建、构建和上载简单的 ISM。完成上载操作后，可以运行 SA 客户端并检查包含上载的 ISM 的软件策略。

在安装 IDK 的主机的终端窗口中执行以下步骤。除非另有说明，否则 Unix 和 Windows 上的命令相同。

1 Unix：以 `root` 身份登录到安装了 IDK 的服务器。

如果无法以 `root` 身份登录，则以其他 Unix 用户身份登录，并按照 CRYPTO_PATH（第 114 页）中的描述设置 CRYPTO_PATH 环境变量。

2 Windows：打开终端窗口并确保设置了 CRYPTO_PATH 环境变量。

3 通过输入 `ismusertool` 命令，授予用户上载 ISM 的权限，例如：

```
ismusertool --addUser johndoe
```

此命令要求您确认是否通过代理网关访问核心：

```
Using an agent gateway to reach an Opsware Core.
Is this correct?[y/n]: y
```

接下来，此命令提示您输入 Opsware `admin` 用户名和密码：

```
Enter Opsware Admin Username: admin
Enter admin's Opsware Password:
```

有关详细信息，请参见 ISMUserTool（第 117 页）。

4 创建新 ISM。

例如，要创建名为 `foo` 的 ISM，可以输入以下命令：

```
ismtool --new foo
```

此命令在当前目录级别创建一个目录 foo。 ISM 由 foo 目录的内容组成。您将在后续的 ismtool 命令中指定 foo ISM。

5 向 ISM 添加应用程序文件。

一种添加应用程序文件的方法是将一个或多个存档复制到 bar 子目录。例如，如果应用程序位在 mytest.zip 文件中，则可以如下所示将其添加到 ISM：

Unix：

```
cp /tmp/mytest.zip foo/bar
```

Windows：

```
copy c:\temp\mytest.zip foo\bar
```

6 设置软件策略的路径，该策略中包含要在后面步骤中上载的 ISM。

注意：对于包含软件策略的文件夹，您必须具有 “在文件夹中写入对象” 权限。文件夹权限在 SA 客户端的 “文件夹属性” 窗口上设置。

以下 ismtool 命令设置了软件策略 Quote Policy 的路径：

Unix：

```
ismtool  --opswpath '/My Kit/Service/Quote Policy' foo
```

Windows：

```
ismtool  --opswpath "/My Kit/Service/Quote Policy" foo
```

在 Unix 上要将路径包含在单引号内，但在 Windows 上要使用双引号。对于 Unix 和 Windows，路径都包含正斜杠。

7 通过输入以下命令，在 ISM 中构建包：

```
ismtool --build foo
```

此命令在 foo/pkg 子目录中创建三个包。在 Linux 系统上，这三个包如下所示：

```
foo-1.0.0-1.i386.rpm
foo-ism-1.0.0-1.i386.rpm
ismruntime-rpm-3.0.0-1.i386.rpm
```

foo-1.0.0-1.i386.rpm 包内含应用程序位，在本例中这些应用程序位已复制到 step 5 中的 foo/bar 子目录。foo-ism-1.0.0-1.i386.rpm 包保存安装挂接和控制脚本。（因为本例比较简单，所以没有控制脚本。） ismruntime-rpm-3.0.0-1.i386.rpm 包内含 SA 共享运行时，服务器代理在托管服务器上安装包时将使用这些运行时。

请注意，包类型 (RPM) 与 Linux 系统的本地打包引擎相对应。在 Windows 上，--build 命令在 foo\pkg 子目录中创建以下 MSI 包：

```
foo-1.0.0-1.msi
foo-ism-1.0.0-1.msi
ismruntime-msi-3.0.0-1.msi
```

8 通过输入以下命令，将 ISM 上载到软件策略：

```
ismtool --upload foo
```

此命令生成多个提示。首先，它要求您确认要将 ISM 上载到哪个核心：

```
Using the following Opsware Core:

  Data Access Engine : d02      192.168.198.91:1004
  Software Repository: d02      192.168.198.91:1003
  Command Engine     : d02      192.168.198.91:1018

  Is this correct?[y/n]: y
```

接下来， `--upload` 命令会提示您输入 Opsware 用户名和密码：

```
  Enter Opsware Username: johndoe
  Enter johndoe's Opsware Password:
.. .
Success!
```

9 在 SA 客户端中，打开软件策略并验证它是否包含前面的步骤中上载的 ISM。

平台差别

通常情况下， IDK 在不同平台（操作系统）的包上具有相同功能。但也有一些差别，如以下各节所述。

Solaris 差别

Solaris 包名称不能超过 9 个字符。按照约定，格式为一组大写字母后跟一组标识应用程序的小写字母。（可选）最后的字符可以有特殊含义。请注意，此格式为约定，而不是强制要求。下面是一些 Solaris 包名称示例：

```
SPROcc
SPROcmpl
SPROcodmg
SUNWgssx
SUNWgzip
SUNWhea
SUNWhiu8x
SUNWhmd
SUNWhmdu
SUNWhmdx
```

在 ISMTool 创建 Solaris 包时，它使用的包名称长度不得超过 9 个字符。ISMTool 构造的包名称以 `ISM` 开头，后跟 ISM 名称的前五个字符，然后是字母 `c`（代表控制包）或数字 `0`（代表应用程序包的第一部分）、数字 `1`（代表应用程序包的第二部分，以此类推）。例如，如果 ISM 名称为 `foobar`，则包名称如下：

```
ISMfooba0
ISMfoobac
```

如果出现截断情况，ISMTool 将生成一个警告，以便开发人员可以重命名 ISM 以避免命名冲突。要查看包名称，请使用 Solaris `pkginfo` 命令。

如果上载 Solaris 中继包，则不会上载响应文件。必须手动上载响应文件。

Windows 差别

在 Windows 上，当 ISMTool 创建应用程序和控制 Windows Installer (MSI) 包时，会如下所示对 `ProductName` 和 `ProductVersion` 进行编码：

```
ProductName:     <name>-<version>
ProductVersion:  0.0.<app|ctl release>
```

`<name>`、`<version>` 和 `<release>` 对应于 ISM 的内部信息，这些信息可使用 ISMTool 的 `--info` 命令进行查看。这种编码方案是特意设计的，是修正过程正常进行所必需的。

5 IDK 构建环境

ISM 文件系统结构

ISMTool --build 和 --upload 命令在 ISM 目录中运行，该目录可以使用 --unpack 命令或 --new 命令创建。 --unpack 命令解压缩先前使用 --pack 压缩的文件 （包含 ISM 目录内容）。 --new 命令最初会创建 ISM 目录。例如，以下命令创建名为 ntp-4.1.2 的新目录：

```
ismtool --new ntp-4.1.2
```

此命令在 ntp-4.1.2 目录下创建以下子目录：

- bar - 包含二进制存档，其内容用于创建应用程序包。
- doc - 在构建 ISM 期间自动生成的文档 (HTML) 的位置。您还可以在目录中创建其他文档文件。
- ism - 包含创建 ISM 的控制包所需的全部文件。在 ism 目录中，您可以编辑默认包挂接 （预安装、后安装、预卸载、后卸载），还可以向 ism/control 添加控制脚本。
- log - 保存跟踪源转换 （编译或本地安装）输出、本地打包引擎 （如 msi、 rpm、 pkgtrans、 swpackage）输出或 SA 上载的文件。
- pad - 包含 ISMTool --addPkgProp 选项指定的安装脚本（预安装、后安装、预卸载、后卸载）。
- pkg - 包含 --build 生成的所有应用程序包、控制包和共享运行时包。此子目录还包含中继包的副本。
- tmp - 用作 ISMTool 操作的暂存空间。
- src - 可以选择包含能控制源编译成二进制存档的文件。

下面的列表显示了在执行以下命令后， ISM 子目录的内容：

```
ismtool --build ntp-4.1.2
```

源构建的输出位于二进制存档目录下，生成的名称为 __ntp-4.1.2_src_ntp.spec.cpio。除了名称以两个下划线开头的其他文件外，构建过程还在 log、 pkg 和 tmp 子目录中创建文件。

```
ntp-4.1.2/
            src/
                ntp-4.1.2.tar.gz
                ntp.spec
            bar/
                __ntp-4.1.2_src_ntp.spec.cpio
```

```
    __ntp-4.1.2_src_ntp.spec.cpio.meta
pkg/
   ntp-4.1.2-3.i386.rpm
   ntp-ism-4.1.2-7.i386.rpm
   ismruntime-rpm-2.0.rpm
log/
    ...
doc/
    index.html
    index/
        ntp-4.1.2-3.i386.rpm.html
        ntp-ism-4.1.2-7.i386.rpm.html
tmp/
   ...
ism/
    ism.conf
    bin/
        ismget
        parameters
        platform
        python
    env/
        ism.sh
        ism.py
        ism.pl
    pkg/
        ism_check_install
        ism_post_install
        ism_post_uninstall
        ism_pre_install
        ism_pre_uninstall
    control/
pad/
    ismruntime-rpm-2.0.0.i386.rpm
     ...
    ntp-4.1.2-3.i386.rpm/
                          pkg.conf
                          scripts/
    ntp-ism-4.1.2-7.i386.rpm/
   ...
```

构建过程

本节描述了以下内容：

- 何时调用 --build 命令
- 多个命令行选项
- --build 命令执行的操作
- --build 命令创建的包

何时调用 --build 命令

在 --new 之后、 --upload 之前运行 ISMTool --build 命令。只要使用选项更改了 ISM，就必须在 --upload 前调用 --build，以使更改生效。例如，如果指定 --opswpath，则必须调用 --build，从而在上载 ISM 前使新的软件策略路径生效。

多个命令行选项

您可以在同一个命令行上调用多个 ISMTool 选项，也可以单独调用每个选项。在以下 Unix 示例中，命令将 ISM 目录 apache 的本地包引擎更改为 rpm3，版本更改为 2.0.47b，默认安装用户更改为 root，默认安装组更改为 root：

```
ismtool --pkgengine rpm3 --version 2.0.47b --user root --group root apache
```

它相当于下面的一系列命令：

```
ismtool --pkgengine rpm3     apache
ismtool --version   2.0.47b  apache
ismtool --user      root     apache
ismtool --group     root     apache
```

ISMTool 按照正确的逻辑执行顺序对命令操作进行排序。例如，以下命令在执行构建前将 apache 的版本更改为 3.0。

```
ismtool --build --version 3.0 apache
```

--build 命令执行的操作

ISMTool --build 命令执行以下步骤。

1. 通过删除所有构建副产品，执行构建前清理。但这一步会使先前的构建过程中生成的所有 cpio 存档成为构建缓存形式。
2. 运行可选脚本 ism/build/ism_clean。 ism/build 子目录中的脚本会挂接到构建过程中。要运行这些脚本，必须手动创建它们。
3. 对应用程序源运行校验和，如果当前校验和与先前的校验和不匹配，则增加应用程序发布版号。
4. 对控件源 （ism 子目录的内容）运行校验和，如果当前校验和与先前的校验和不匹配，则增加控件发布版号。
5. 运行可选脚本 ism/build/ism_pre。
6. 对于源构建，递归搜索 src 子目录中的 .spec 文件，编译并执行每个文件。
7. 创建共享运行时包。
8. 创建控制包。
9. 创建应用程序包。
10. 生成自动 HTML 文档 doc/index/index.html。
11. 运行可选脚本 ism/buid/ism-post。

--build 命令创建的包

--build 命令在 pkg 子目录中创建以下包：

- 应用程序包 - 根据 bar （二进制存档）子目录的内容创建，此包内含应用程序位。在调用 --build 命令前，将应用程序存档复制到 bar 子目录。应用程序包的文件名具有以下语法。<version> 针对整个 ISM，<release> 特定于应用程序包。有关详细信息，请参见 ISM 名称、版本号和发布版号 （第 78 页）。

  <name>-<version>-<release>.<package-extension>

- 控制包 - 此包内含来自 ism 子目录的控制脚本和安装脚本。控制包文件名具有以下语法：

  <name>-ism-<version>-<release>.<package-extension>

- 共享运行时包 - 此包保存服务器代理 （在安装期间）和任何控制脚本调用的共享运行时例程。这些运行时例程用于 Server Automation ，而不是用于应用程序本身。共享运行时包的文件名具有以下语法。（只有使用 --ctlprefix 选项指定了非默认值时，文件名中才包含 <ctl-prefix>。）

  ismruntime-<ctl-prefix>-<package-type>-<idk-version>.<package-extension>

- 中继包 - 可以使用 --addPassthruPkg 选项指定这些包，将它们原样复制到 pkg 子目录。

指定 ISM 的应用程序文件

本节讨论将应用程序文件放入 ISM 的方法：

- 将存档放入 bar 子目录中
- 指定中继包
- 编译源 （仅限 Unix）

将存档放入 bar 子目录中

在运行 --build 之前，可以将文件存档手动复制到 ISM 的 bar（二进制存档）子目录。或者，bar 子目录中的存档也可以由规范文件的 %files 部分中的指令作为 cpio 文件生成。另请参见编译源（仅限 Unix）（第 75 页）。

--build 命令将 bar 子目录中的存档重新打包成 pkg 子目录的应用程序包。下表列出了可以保存在 bar 子目录中的存档类型。

表 46　有效的二进制存档类型

文件扩展名	存档类型
.cpio	Unix CPIO 存档
.msi	Microsoft 安装程序
.rpm	RPM 程序包管理器
.tar	磁带存档
.tar.bz2	bzip2 压缩磁带存档
.tar.gz	gzip 压缩磁带存档
.zip	Info-Zip 兼容的 Zip

指定中继包

与 bar 子目录中的存档不同，中继包不会提取并重新打包。--addPassthruPkg 命令将中继包原样复制到 pkg 子目录中。 --addPassthruPkg 指定的包不能存放在 ISM 目录中。以下示例将中继包添加到 ISM 中，并指定要添加到软件策略的包：

```
ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

编译源 （仅限 Unix）

--build 命令在 src 子目录中递归搜索规范文件（以 .spec 结尾的文件）。如果找到规范文件，则将其编译成 Bourne Shell 并执行。规范文件是用 RPM 规范文件语言的简化衍生版编写的。ISMTool 的类似于规范文件的语言编译器允许您使用经过略微修改的现有 RPM 规范文件。

有关规范文件语言的详细信息，请参见位于以下 URL 的 Maximum RPM 文档：

http://www.rpm.org/max-rpm/index.html

示例规范文件

下面是 NTP 4.1.2 的一个简单 ISM 规范文件示例：

```
################################################
# Common Preamble
################################################

%define ismname %(../ism/bin/ismget name)
%define version %(../ism/bin/ismget version)
%define prefix %(../ism/bin/ismget prefix)

Name: %{ismname}
Version: %{version}
```

```
################################################
# prep, build, install, files
################################################

Source: http://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.1.2.tar.gz

%prep

%setup -n ntp-4.1.2

%build

%ifos Solaris2.7
echo ''do something Solaris2.7 specific''
%endif

%ifos Linux
echo ''do something Linux specific''
%endif

./configure --prefix=%prefix
make

%install
/bin/rm -rf $ISM_BUILD_ROOT
make install prefix=$ISM_BUILD_ROOT/%{prefix}

%files
%defattr(-,root,root)
%prefix
```

规范文件前言

前言指定要使用程序 ismget 从 ISM 获取的信息。以下行将获取 ISM 的名称、版本和前缀。

```
%define ismname    %(../ism/bin/ismget name)
%define version    %(../ism/bin/ismget version)
%define prefix     %(../ism/bin/ismget prefix)
```

在设置和编译源时这些获取的信息非常有用。不过 %define 命令是可选命令。前言中唯一要求的标记是 Name 和 Version。

%prep

%prep 部分专用于准备供编译的源。这包括解压缩源分发。单个源文件使用一个 Source 标记进行标识。源文件列表使用一系列标记进行标识：Source0, Source1, ... 同样，修补程序也使用一个 Patch 标记或一系列标记进行标识：Patch0, Patch1, ...ISMTool 如 Maximum RPM 中所述复制宏功能。 %setup 宏控制源文件的解压缩方式。 %prep 部分也可以使用 %patch 宏管理修补程序。

%build

%build 部分的 shell 脚本命令将源转换为二进制。从源编译通常包括运行 ./configure -prefix=%{prefix} 和 make 。可以根据平台（操作系统）执行配置切换。平台标记设计用于向后兼容实际安装中的 RPM。下列平台字符串是 ISMTool 规范文件中可用于平台分支的一些示例：

```
Linux
RedHat
RedHat-Linux-7.2
RedHat-Linux-AS2.1
Solaris
Solaris2.8
Solaris-2.8
SunOS
SunOS5.7
SunOS-5.7
hpux
hpux11.00
hpux-11.00
HPUX
HPUX11.00
HPUX-11.00
aix
aix4.3
aix-4.3
AIX
AIX4.3
AIX-4.3
```

%install

%install 部分指定将文件从构建位置复制到虚拟安装位置。例如，如果 %prefix 设置为 /usr/local，则以下行将 NTP 安装到 /usr/local/bin：

```
make install prefix=$ISM_BUILD_ROOT/%{prefix}
```

变量 $ISM_BUILD_ROOT（或与之相当的 $RPM_BUILD_ROOT）是 ISM 的 tmp 目录中一个临时目录的位置。该临时目录充当将应用 %files 部分指令的虚拟安装根目录。

%install 部分还指示二进制安装中文件的可提取位置。在二进制安装中，因在开发服务器上进行二进制安装而生成的文件可以打包到虚拟安装位置。但如果无法执行此操作，则二进制安装程序可以传输到终端系统，并利用 ISM 后安装挂接进行安装。在这种情况下，将会创建安装程序的二进制存档并将其复制到 ISM 的 bar 子目录。

%files

在规范文件中，源转换阶段的输出是 %files 部分的指令所指示的一组文件。这些文件存档到 ISM 的 bar 子目录中的 cpio 中。

源转换的最后阶段是选择安装到 $ISM_BUILD_ROOT 中的文件。 %files 部分的指令是 Maximum RPM 中记录的选择机制的一部分。这些指令指定与 $ISM_BUILD_ROOT 相关的一列文件或目录（按递归方式收集在一起）。在此示例中，安装路径为 $ISM_BUILD_ROOT/%{prefix}。要选择这些文件进行打包，只需提供 %prefix 作为打包目录。

除了通过命名文件或目录来选择文件外，还可以描述元信息。 %defattr(-,root,root) 行告知存档引擎使用它在文件系统中找到的模式，但要创建存档来使用 root,root 替换它在文件系统中找到的文件所有权。有关 %defattr() 和 %attr() 的完整文档，请参见 Maximum RPM。

ISM 名称、版本号和发布版号

本节包含以下内容：

ISM 名称、版本和发布版的初始值

--new 命令为新 ISM 创建目录，并指定 ISM 的内部基础名称。例如，以下命令在文件系统中创建 mystuff 目录、将内部基础名称设置为 mystuff，并将版本号设置为 1.0.0。

```
ismtool --new mystuff
```

在大多数情况下，使用 --new 指定版本号。以下命令创建目录 ntp-1.4.2、将内部基础名称设置为 ntp，并将版本号设置为 1.4.2：

```
ismtool --new ntp-1.4.2
```

要查看内部基础名称、版本号和发布版号，请使用 --info 命令：

```
ismtool --info ntp-1.4.2.
```

上面的命令生成的输出包括以下内容：

```
...
name:                   ntp
version:                4.2.1
appRelease:             0
...
ctlRelease:             0
...
```

ISM 版本号和发布版号比较

ISM 版本号和发布版号有多处不同。您可以使用 --new 或 --version 命令指定版本号。ISMTool 会自动生成发布版号，您无法指定。版本号应用于整个 ISM。每个应用程序包和控制包都有单独的发布版号。每次重新生成包时， --build 命令都会增加发布版号。由于应用程序包和控制包可以独立构建，所以不同的包可能具有不同的发布版号。

应用程序包和控制包的名称包括内部基础名称、版本号和发布版号。例如，ntp-4.1.2-3.i386.rpm 应用程序包的版本号为 4.1.2，发布号为 3。另请参见 --build 命令创建的包 （第 74 页）。

要显示 IDK （而不是 ISM）的版本，请输入以下内容：

```
ismtool --myversion
```

升级 ISM 版本

尽管您可以修改内部基础名称 （使用 --name）和版本号 （使用 --version），但却不建议您这样做，因为系统不会自动更改目录名称。如果您更改内部基础名称或版本，为避免冲突，还应重命名包含 ISM 的目录。

推荐的做法是使用匹配的内部基础名称、版本号、目录名称和软件策略路径。例如，要将 foo-1.2.7 升级到 foo-1.2.8，您需要执行以下步骤：

1 在与 foo-1.2.7 相同的目录级别，创建新的 ISM 目录：

```
ismtool --new foo-1.2.8
```

2 将存档复制到 foo-1.2.8/bar 目录或指定中继包。

3 在与前一版本相同的级别，将路径设置为软件策略。

Unix：

```
ismtool --opswpath 'MyFolder/{$NAME}/{$VERSION}'
```

Windows：

```
ismtool --opswpath "MyFolder/{$NAME}/{$VERSION}"
```

命令 --opswpath 使用 foo 替代 NAME 变量并使用 1.2.8 替代 VERSION 变量。要查看这些变量的当前值，请使用 --info 命令。有关变量替换的详细信息，请参见 ISMTool 变量 （第 106 页）。

4 使用 ISMTool 构建和上载 foo-1.2.8 ISM。

5 在 SA 客户端中，将 foo-1.2.7 策略与托管服务器分离。

6 （可选）删除 foo-1.2.7 策略。

7 根据新的软件策略修正托管服务器。

6 IDK 脚本

ISM 脚本概述

ISM 脚本是位于 ISM 目录中的 Unix shell 或 Windows 命令行脚本。下面的各节描述了不同类型的 ISM 脚本：

- **安装挂接**：安装挂接由 ISMTool `--build` 命令捆绑到 ISM 的控制包，由托管服务器上的本地打包引擎（如 `rpm`）运行。安装挂接可以调用控制脚本。
- **控制挂接**：控制脚本也捆绑到 ISM 的控制包，负责执行应用程序的特定日常任务，例如启动软件服务器。
- **安装脚本**：安装脚本不包含在控制包中，而是存储在软件数据库中，可以在 SA 客户端中的包“属性”上查看。

开发和运行安装挂接及控制脚本的整个流程如下所示：

1. 调用 ISMTool `--new` 命令，创建默认安装挂接。
2. 使用文本编辑器创建控制脚本。
3. 使用文本编辑器修改默认安装挂接，该挂接可以调用控制脚本。
4. 使用 ISMTool 构建和上载 ISM。
5. 在 SA 客户端中，将 ISM 中包含的应用程序安装到托管服务器上。在安装期间，预安装和后安装挂接在托管服务器上运行。
6. 在应用程序的生产生命周期中，运行或计划控制脚本。
7. 在应用程序的生命周期结束时，使用 SA 客户端卸载应用程序。在卸载期间，预卸载和后卸载挂接在托管服务器上执行。

安装脚本的整个流程与安装挂接和控制脚本的整个流程不同。有关详细信息，请参见安装脚本（第 92 页）。

ISM 脚本无法调用锁定与脚本关联的包的程序（如 `rpm` 或 `pkgadd`）。

安装挂接

安装挂接是位于 `ism/pkg` 子目录中的脚本。（有些文档将安装挂接称为“打包脚本”。）安装挂接在托管服务器上安装和卸载应用程序期间的特定阶段运行。

创建安装挂接

ISMTool --new 命令创建以下安装挂接：

Unix：

```
ism/pkg/
        ism_check_install
        ism_post_install
        ism_post_uninstall
        ism_pre_install
        ism_pre_uninstall
```

Windows：

```
ism\pkg\
        ism_post_install.cmd
        ism_post_uninstall.cmd
        ism_pre_install.cmd
        ism_pre_uninstall.cmd
```

要自定义安装挂接，可以使用文本编辑器对其进行修改。尽管您可以编辑安装挂接，但无法更改其文件名。

默认 ism_pre_install 和 ism_post_uninstall 挂接只是存根，它们不执行任何操作。默认 ism_post_install 挂接调用 ism_configure 和 ism_start 控制脚本。默认 ism_pre_uninstall 挂接调用 ism_stop 控制脚本。请注意，ISMTool 不会自动创建控制脚本；您必须使用文本编辑器创建控制脚本。（请参见控制脚本 （第 86 页）。）

有关 --build 命令创建的默认安装挂接的内容，请参见以下各节：

- Unix 的默认安装挂接 （第 84 页）
- Windows 的默认安装挂接 （第 85 页）

检查安装挂接

有些本地打包引擎直接支持 ism_check_install 挂接；其他引擎隐式支持 ism_pre_install 挂接。 ISMTool 将 check_install 功能映射到本地打包引擎。如果 check_install 脚本返回非零代码，则表示安装停止。

调用安装挂接

将 ISM 的应用程序安装到托管服务器（或将其卸载）时，服务器上的本地打包引擎调用安装挂接。（您不直接运行安装挂接。）例如，在 Linux 系统上， rpm 实用程序在安装应用程序位之前立即调用 ism_pre_install，并在删除应用程序位之后立即调用 ism_post_uninstall。

另请参见安装脚本与安装挂接的调用 （第 93 页）。

安装挂接和 ZIP 包

与其他一些打包引擎不同，Server Automation 使用的 ZIP 打包引擎不支持安装挂接。如果指定了 ZIP 打包引擎，并且安装挂接文件不为空，则 ISMTool 将发出警告并忽略安装挂接文件。

ZIP 包和安装目录

IDK 创建的 ZIP 包无法重定位。也就是说，无法使用同一个 ZIP 包在单个托管服务器上的不同目录中安装应用程序的多个实例。因此，如果最终用户更改了 ZIP 包在 SA 客户端中的 “安装路径” 字段，包安装将会失败。要更改安装目录， ISM 开发人员需要使用 `--prefix` 或 `--ctlprefix` 选项指定新路径、构建新 ISM，并将新 ISM 上载到核心。（对于 Windows NT4，这些选项是必需的，无法指定变量。）

作为 ZIP 包的最佳实践， ISM 开发人员应在 ISM 的描述中包含如下所示的警告：“警告：禁止更改此包的安装路径。”

安装挂接功能

您可以自定义安装挂接来执行类似于下表中所列的操作。

表 47　安装挂接功能

安装挂接	一般功能
`ism_pre_install`	创建必需的目录，创建用户，设置目录权限
`ism_post_install`	调用 `ism_configure` 控制脚本，调用 `ism_start` 控制脚本 （例如，用于启动 Web 服务器）
`ism_pre_uninstall`	调用 `ism_stop` 控制脚本 （用于停止服务器）
`ism_post_uninstall`	执行任何必需的清理工作

仅控制 ISM 的脚本

如果指定 `--skipApplicationPkg` 选项， ISMTool 将不构建应用程序包，并且支持创建仅控制 ISM。您可以使用此功能为不使用 ISMTool 安装或打包的应用程序构建控制器。例如核心操作系统功能的控制器、无法打包的当前运行应用程序和专用硬件的控制器。

在安装和卸载仅控制 ISM 期间，将运行 `ism_ctl_post_install` 和 `ism_ctl_pre_uninstall` 脚本。（脚本为所有 ISM 运行，但通常只为仅控制 ISM 指定这些脚本。）因为这些脚本不是 ISMTool 生成的，所以必须在运行 `--build` 命令前创建。下面的列表显示了这些脚本必需的名称和位置：

Unix：

```
ism/pkg/
        ...
        ism_ctl_post_install
        ism_ctl_pre_uninstall
```

Windows：

```
ism\pkg\
        ...
        ism_ctl_post_install.cmd
        ism_ctl_pre_uninstall.cmd
```

安装挂接在托管服务器上的位置

在开发系统上， `--build` 命令将安装挂接捆绑到 ISM 的控制包。在托管服务器上，控制包的内容安装到 ISM 的 `ctlprefix` 所指示的目录。默认情况下，安装挂接安装到以下目录：

Unix：

```
/var/opt/OPSWism/<ism-name>/pkg
```

Windows：

```
%ProgramFiles%\OPSWism\<ism-name>\pkg
```

要更改安装挂接的默认目录，请在构建和上载 ISM 之前指定 `--ctlprefix` 选项。例如，如果如下所示指定 `ctlprefix`，则安装挂接将安装到 `/usr/local/ntp-4.1.2/pkg`：

```
ismtool --ctlprefix /usr/local ntp-4.1.2
```

Unix 的默认安装挂接

默认 `ism_pre_install` 挂接：

```
#!/bin/sh
#
# ISM Pre Install Script
#
. `dirname $0`/../env/ism.sh
```

默认 `ism_post_install` 挂接：

```
#!/bin/sh
#
# ISM Post Install Script
#
. `dirname $0`/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_configure ]; then
${ISMDIR}/control/ism_configure
fi
if [ -x ${ISMDIR}/control/ism_start ]; then
${ISMDIR}/control/ism_start
fi
```

默认 ism_pre_uninstall 挂接：

```
#!/bin/sh
#
# ISM Pre Uninstall Script
#
. `dirname $0`/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_stop ]; then
${ISMDIR}/control/ism_stop
fi
```

默认 ism_post_unininstall 挂接：

```
#!/bin/sh
#
# ISM Post Uninstall Script
#
. `dirname $0`/../env/ism.sh
```

Windows 的默认安装挂接

默认 ism_pre_install.cmd 挂接：

```
@echo off
REM
REM ISM Pre Install Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
ENDLOCAL
```

默认 ism_post_install.cmd 挂接：

```
@echo off
REM
REM ISM Post Install Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
REM
REM Call the ISM's configure script
REM
IF EXIST "%ISMDIR%\control\ism_configure.cmd"
call "%ISMDIR%\control\ism_configure.cmd"
REM
REM Call the ISM's start script
REM
```

```
IF EXIST "%ISMDIR%\control\ism_start.cmd"
call "%ISMDIR%\control\ism_start.cmd"
ENDLOCAL
```

默认 ism_pre_uninstall.cmd 挂接：

```
@echo off
REM
REM ISM Pre Uninstall Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
REM
REM Call the ISM's stop script
REM
IF EXIST "%ISMDIR%\control\ism_stop.cmd"
call "%ISMDIR%\control\ism_stop.cmd"
ENDLOCAL
```

默认 ism_post_unininstall.cmd 挂接：

```
@echo off
REM
REM ISM Post Uninstall Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
```

控制脚本

ISM 控制脚本位于 ism/control 目录中。控制脚本在应用程序安装后执行日常管理或维护任务。

安装挂接可以运行控制脚本。如果任务在安装（或卸载）期间执行，但可能还会定期执行，则应该编码为控制脚本。例如， ism_post_install 挂接可以调用 ism_start 控制脚本，以在应用程序安装后立即启动该应用程序。另外， ism_pre_uninstall 挂接还可以调用 ism_stop 控制脚本来关闭应用程序。

用户可以从 SA 客户端的“ISM 控制”窗口运行控制脚本。高级用户可以从全局 Shell 中的命令行运行控制脚本。

创建控制脚本

与安装挂接不同，ISMTool 不会创建控制脚本；您必须使用文本编辑器创建控制脚本。可以向 ism/control 子目录添加任意数量的控制脚本。按照约定，控制脚本的文件名如下所示：

Unix：

```
ism/control/
           ism_start
           ism_stop
           ism_configure
           ism_reconfigure
```

Windows：

```
ism\control\
           ism_start.cmd
           ism_stop.cmd
           ism_configure.cmd
           ism_reconfigure.cmd
```

控制脚本名称在 SA 客户端的“ISM 控制”窗口中的显示可能与此不同。“ISM 控制”窗口的“操作”字段显示控制脚本的名称，但不使用前导 ism_ 或文件类型扩展名。例如，名为 ism_start.cmd 的控制脚本在 “操作”字段中显示为 start。“操作”字段仅显示控制脚本名称的前 25 个字符。因此，名称的前 25 个字符应该是唯一的。对于 Unix 和 Windows，前导 ism_ 必须是小写字符，否则“操作”字段会显示前缀。

对于 Unix，确保 ism/control 下的控制脚本可执行。否则它们不会显示在 SA 客户端中。

控制脚本功能

控制脚本适合用于管理应用程序的重复性任务。下表概述了控制脚本的典型用途。

表 48　控制脚本功能

控制脚本	一般功能
ism_start	通知任何相关或依赖服务器并启动应用程序
ism_stop	通知任何相关或依赖服务器并停止应用程序
ism_configure	执行配置操作
ism_reconfigure	类似于 ism_configure，但首先调用 ism_stop，然后再调用 ism_start

控制脚本在托管服务器上的位置

与安装挂接一样，控制脚本也通过 --build 命令捆绑到控制包。在托管服务器上，控制脚本位于 ISM ctlprefix 值所指示的目录中。默认情况下，控制脚本安装在托管服务器上的以下目录中：

Unix：

/var/opt/OPSWism/<ism-name>/control

Windows （NT4 除外）：

%ProgramFiles%\OPSWism\<ism-name>\control

要更改默认目录，请使用 ISMTool 指定 --ctlprefix 选项。对于 Windows NT4，必须指定 --ctlprefix，并且不能包含变量。

动态配置 ISM 参数

ISM parameter 实用程序支持控制脚本和安装挂接访问 SA 自定义特性的值。 ISM 参数的键与其对应自定义特性的名称匹配。自定义特性的值确定参数的值。自定义特性的源是设施、客户、服务器或设备组之类的 SA 对象。

自定义特性使用 SA 客户端设置，是一个包含配置信息的名称 - 值对。例如，要指定 Apache Web 服务器的端口号，自定义特性 APACHE_1.3_PORT 的值可以为 80。如果 ISM 拥有参数 APACHE_1.3_PORT，则控制脚本可以访问自定义特性的当前值。

使用 SA 客户端的 “ISM 控制”窗口，最终用户可以查看参数的源 （SA 对象）、查看参数值，还可以覆盖参数值。

ISM 参数的开发流程

开发和使用 ISM 参数的整个流程如下所示：

1. 使用 ISMTool 添加新参数。
2. 使用文本编辑器编写控制脚本 （或者修改安装挂接）以访问参数。
3. 使用 ISMTool 构建和上载 ISM。
4. 在 SA 客户端中，将 ISM 中包含的应用程序安装到托管服务器上。
5. 在 SA 客户端中，使用与参数相同的名称创建自定义特性。
6. 在 SA 客户端中，在托管服务器上运行控制脚本。在运行时，脚本从 Server Automation 检索参数 （控制特性）。

添加、查看和删除 ISM 参数

ISMTool `--addParam` 命令创建新参数，ISM 中的任何脚本都可以获取此参数。参数是具有四个字段的元组，每个字段由 ISMTool 选项指定。下表列出了各个字段及其对应的选项。

表 49　ISM 参数字段

参数字段	ISMTool 选项	描述
名称	`--addParam`	ISM 参数的名称，必须与自定义特性的名称匹配。
默认值	`--paramValue`	参数的默认值。如果找不到匹配的自定义特性，脚本将使用默认值。

类型	`--paramType`	参数的数据类型。允许的值：
		`'String'`
		`'Template'`

描述	`--paramDesc`	描述参数的文本。

以下 Unix 命令将参数 `NTP_SERVER` 添加到 `ntp-4.2.1` ISM：

```
ismtool --addParam NTP_SERVER \
     --paramValue 127.0.0.1 \
     --paramType 'String' \
     --paramDesc 'NTP server, default to loopback' ntp-4.2.1
```

要查看已添加到 `ntp-4.2.1` ISM 的参数，请输入以下命令：

```
ismtool --showParams ntp-4.2.1
```

要删除本例中添加的参数，请输入以下命令：

```
ismtool --removeParam NTP_SERVER ntp-4.2.1
```

访问脚本中的参数

使用 ISMTool 添加参数后，可以编写 ISM 控制脚本来访问参数。支持的脚本语言如下：

- Bourne Shell
- Korn Shell
- Windows command shell
- Python
- Perl

Shell 脚本通过环境变量访问参数， Python 脚本通过字典访问， Perl 脚本通过哈希表访问。

ISM parameters 实用程序

为获取参数，控制脚本运行位于 ISM 共享运行时包中的 parameters 实用程序。只能获取使用 --addParam 命令定义的参数。

parameters 实用程序具有以下语法：

```
parameters [options]
--scope <scope> ; server|servergroup|customer|facility|
                ; servicelevel|os|custapps|webserver|appserver|
                ; dbserver|systemutilities|osextras|install|
                ; default (default is all)
-s/--sh                 ; Bourne Shell syntax
-k/--ksh                ; Korn-Shell syntax
-p/--python             ; Python repr'ed dictionary
-l/--perl               ; PERL map
-c/--cmd                ; Windows Cmd syntax
-b/--vbscript           ; Windows VBScript syntax
-h/--help               ; Help
-v/--version            ; Version
```

--scope 选项限制只能在 Server Automation 的指定区域搜索自定义特性。例如，如果已经为设施和客户指定了 --scope facility 和自定义特性，则不考虑客户的自定义特性。另请参见：自定义特性的搜索顺序 （第 91 页）。

如果 parameters 实用程序在检索期间遇到错误，则返回特殊参数 _OPSW_ISMERR，其中包含所遇到错误的简要描述。

示例脚本

以下 Bourne Shell 示例是在 Unix 上配置 NTP 时间服务的控制脚本。 parameters 实用程序检索为 ISM 定义的两个参数：NTP_CONF_TEMPLATE 和 NTP_SERVER。

```
#!/bin/sh
. `dirname $0`/../env/ism.sh
eval `${ISMDIR}/bin/parameters`
echo $NTP_CONF_TEMPLATE | \
sed "s/NTP_SERVER_TAG/$NTP_SERVER/" > /etc/ntp.conf
```

下面用 Python 编写的控制脚本也配置 NTP。

```
#!/usr/bin/env python
import os
import sys
import string
ismdir=os.path.split(sys.argv[0])[0]
cmd = '%s --python' % (os.path.join(ismdir,'bin','parameters'))
params = eval(os.popen(cmd,'r').read())
template = params['NTP_CONF_TEMPLATE']
value = params['NTP_SERVER']
conf = string.replace(template,'NTP_SERVER_TAG',value)
fd=open('/etc/ntp.conf','w')
fd.write(conf)
fd.close()
```

以下示例显示了适用于 Windows 的配置控制脚本。在本例中，对于 32 位 Windows 操作系统，每个参数都以 “名称 = 值”（每行一个）的形式输出。

Windows FOR 命令将每个参数设置为环境变量。（在下面的列表中， FOR 命令分成了两行，但在实际脚本中， FOR 命令必须在一行中。）最后，参数被传递给名为 WindowsNTPConfigureScript.cmd 的 NTP 配置脚本。

```
@echo off
SETLOCAL
cd /d %ISMDIR%
for /f "delims== tokens=1,2" %%i in ('""bin\parameters.cmd""') do set
%%i=%%j WindowsNTPConfigureScript.cmd %NTP_CONF_TEMPLATE% %NTP_SERVER%
ENDLOCAL
```

自定义特性的搜索顺序

利用 SA 客户端，可以在多个位置设置自定义特性。例如，对于托管服务器 foo.hp.com，可以将自定义特性 APACHE_1.3_PORT 设置为 8085，对于与 foo.hp.com 服务器关联的 Widget Corp. 客户，可以将这个自定义特性设置为 80。在运行时，如果 foo.hp.com 上的控制脚本访问 APACHE_1.3_PORT 参数，它将获取哪个值？在这种情况下，值将为 8085，因为服务器的自定义特性在搜索顺序中排在首位。

请注意，如果找不到自定义特性，脚本将使用通过 ISMTool --paramValue 选项设置的默认参数值。

默认搜索顺序

自定义特性的默认搜索顺序如下：

1. 服务器
2. 设备组
3. 客户
4. 领域
5. 设施
6. 操作系统
7. ISM （上载操作期间在软件策略中创建）
8. 修补程序策略
9. 软件策略

多个设备组和服务水平按字母顺序搜索。例如，如果一个服务器同时属于 ABC 和 XYZ 组，则先在 ABC 组中搜索自定义特性，然后在 XYZ 组中搜索。作为子组的服务器组不继承其父组的自定义特性。

安装脚本

安装脚本位于 pad 子目录中。与安装挂接相同，安装脚本在托管服务器上的应用程序安装和卸载期间的特定阶段运行。

安装脚本与安装挂接的差别

尽管它们的用途相似，但安装脚本与安装挂接有几个方面的差别，下表列出了这些差别。

表 50　安装脚本与安装挂接的差别

安装脚本	安装挂接
按照 SA 客户端中包的属性显示。	按照 SA 客户端中包的内容显示。（仅显示 RPM。）
位于 pad 子目录中。	位于 ism/pkg 子目录中。
存储在模型库中 （在上载后）。	捆绑在控制包中，安装在托管服务器上 ctlprefix 指定的目录中。
由服务器代理运行。	由本地打包引擎运行。
可以为 ISM 中的每个包定义。	为整个 ISM 定义。

创建安装脚本

虽然 ISMTool 创建 pad 子目录结构，但它不创建默认安装脚本。对于使用 --build 创建的或使用 --addPassthruPkg 添加的每个包， ISMTool 如下所示创建子目录：

```
pad/<package-name>/scripts
```

例如，在 Linux 上 --build 命令将为 ISM ntp-1.4.2 创建以下子目录：

```
pad/ismruntime-rpm-2.0.0-1.i386.rpm/scripts
pad/ntp-ism-4.2.1-1.i386.rpm/scripts
pad/ntp-4.2.1-1.i386.rpm/scripts
```

使用文本编辑器，可以在 scripts 子目录中创建安装脚本。例如，可以为 ntp-4.2.1-1.i386.rpm 包创建安装脚本，如下所示：

```
pad/ntp-4.2.1-1.i386.rpm/scripts/
                                  preinstallscript
                                  pstinstallscript
                                  preuninstallscript
                                  pstuninstallscript
```

安装脚本的文件名必须与前面的示例匹配。例如，安装后立即调用的脚本必须命名为 pstinstallscript。

安装脚本与安装挂接的调用

如果一个 ISM 同时具有安装脚本和安装挂接，则在托管服务器上安装应用程序时， Server Automation 将按以下顺序执行任务：

1. 安装 ISM 运行时包。
2. 安装 ISM 控制包。
3. 运行 `preinstallscript` （安装脚本）。
4. 运行 `ism_pre_install` （安装挂接）。
5. 安装应用程序包 （应用程序位）。
6. 运行 `ism_post_install` （安装挂接）。
7. 运行 `ism_configure` （控制脚本）。
8. 运行 `ism_start` （控制脚本）。
9. 运行 `pstinstallscript` （安装脚本）。

在托管服务器上卸载应用程序时， Server Automation 按照以下顺序执行操作：

1. 运行 `preuninstallscript` （卸载脚本）。
2. 运行 `ism_pre_uninstall` （卸载挂接）。
3. 运行 `ism_stop` （控制脚本）。
4. 卸载应用程序包 （应用程序位）。
5. 运行 `ism_post_uninstall` （卸载挂接）。
6. 运行 `pstuninstallscript` （卸载脚本）。
7. 卸载 ISM 控制包。
8. 卸载 ISM 运行时包。

7 IDK 命令

ISMTool 参数类型

表 51 定义了本章其余部分定义的 ISMTool 命令中所使用的参数类型。例如，ISMTool --new 命令的语法所指定的 ISMNAME 参数类型。

表 51 ISMTool 参数类型

参数类型	描述	示例
路径	文件系统的绝对路径。	/foo/bar
字符串	不含空格的文本字符串。	foobar
文本	任意带引号的文本。在 Unix 上要将文本包含在单引号内，但在 Windows 上要使用双引号。	'This is some text'
布尔值	布尔值。	true 或 false
ISMFILE	文件系统中的有效 .ism 文件的路径。此文件将解压缩到 ISMDIR 中。	/foo/bar/name.ism
ISMDIR	提取的有效 ISMFILE 或新创建的 ISM 的路径。	xyz /home/sam/xyz
ISMNAME	新创建的 ISM 的名称。 ISMNAME 可以采用 STRING 或 STRING-VERSION 格式。	ntp ntp-4.1.2
VERSION	表示 ISM 版本的字符串。 VERSION 不得包含空格，并且必须是本地打包引擎的合法版本字符串。	1.2.3 4.13 0.9.7b
HOST[:PORT]	主机和可选端口。	www.foo.com www.foo.com:8000 192.168.1.2:8000
BYTES	整数字节。	42
SECONDS	整数秒数。	300
PARAMTYPE	预期的参数数据类型。仅允许 'String' 和 'Template' 常量值。在 Unix 上要将值包含在单引号内，但在 Windows 上要使用双引号。	'String' 'Template'

ISMTool 信息命令

本节描述了提供构建环境相关信息的 ISMTool 命令。

--help

显示 ISMTool 命令行帮助。

--env

显示环境中系统级别工具的位置。在调查构建问题和验证环境变量 ISMTOOLBINPATH 的设置是否正确时此命令非常有用。例如，在 Unix 系统上， --env 可能显示以下内容：

```
% ismtool --env
bzip2:/usr/local/ismtool/lib/tools/bin/bzip2
cpio:/usr/local/ismtool/lib/tools/bin/cpio
gzip:/usr/local/ismtool/lib/tools/bin/gzip
install:/usr/local/ismtool/lib/tools/bin/install
17
patch:/usr/local/ismtool/lib/tools/bin/patch
python:/usr/local/ismtool/lib/tools/bin/python
pythonlib:/usr/local/ismtool/lib/tools/lib/python1.5
rpm2cpio:/usr/bin/rpm2cpio
rpm:/bin/rpm
rpmbuild:/usr/bin/rpmbuild
tar:/usr/local/ismtool/lib/tools/bin/tar
unzip:/usr/local/ismtool/lib/tools/bin/unzip
wget:/usr/local/ismtool/lib/tools/bin/wget
zip:/usr/local/ismtool/lib/tools/bin/zip
zipinfo:/usr/local/ismtool/lib/tools/bin/zipinfo
pkgengines:['rpm4']
```

--myversion

显示 ISMTool 的版本。

--info ISMDIR

显示目录 ISMDIR 中包含的 ISM 的内部信息概述。在构建完成后，将会提供更多详细信息，可以在浏览器中通过以下 URL 查看：

```
<ISMDIR>/doc/index/index.html
```

--showParams ISMDIR

显示每个控制参数的名称、默认值、类型和描述。

--showPkgs ISMNAME

显示 ISM 管理的所有包的列表。此列表包括控制包、应用程序包、所有中继包以及中继包所含的所有内部包。内部包的示例包括 Solaris 包内的 Solaris 包实例或 AIX LPP 包内的更新文件集。对于每个托管包，将列出包名称、类型、附加的状态以及所有可以设置的元数据。

--showOrder ISMNAME

显示 ISM 托管的附加包的当前安装顺序。

--showPathProps ISMNAME

Server Automation 6.0 已弃用此选项。

显示当前为软件策略元数据指定的值。

ISMTool 创建命令

本节描述了生成 ISM 目录结构的 ISMTool 命令。

--new ISMNAME

创建新的 ISM，由包含子目录和文件的目录组成。 ISMNAME 的值指定新创建的 ISM 目录的名称。内部 ISM 名称随 ISNAME 的格式而变化。

例如，以下命令创建名为 foobar 的 ISM 目录。ISM 的内部名称为 foobar，ISM 的初始版本默认为 1.0.0。

```
% ismtool --new foobar
```

下面的命令创建名为 ntp-4.1.2 的 ISM 目录。 ISM 的内部名称为 ntp， ISM 的初始版本为 4.1.2。请注意， ISM 的内部名称不包含 -VERSION。

```
% ismtool --new ntp-4.1.2
```

ISM 目录的名称独立于内部 ISM 名称。例如，如果开发人员将 ntp-4.1.2 目录重命名为 myntp，则 ISM 的内部名称仍为 ntp， ISM 的版本仍为 4.1.2。

--pack ISMDIR

为 ISMDIR 中包含的 ISM 创建 ZIP 存档。该存档的名称将为 ismname-version.ism。请注意，ISMDIR 的内容必须小于 2GB。（如果大小超过 2GB，则使用 zip 或 tar 实用程序。） --pack 的示例如下：

Unix：

```
% ismtool --new tick
% ismtool --version 3.14 tick
% ls
tick/
% mv tick spooon
% ls
spooon/
% ismtool --pack spooon
% ls
spooon/ tick-3.14.ism
```

Windows：

```
% ismtool --new tick
% ismtool --version 3.14 tick
% dir
11/21/2003 10:17a <DIR> tick
% move tick spoon
% dir
11/21/2003 10:17a <DIR> spoon
% ismtool --pack spoon
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

--unpack ISMFILE

解压缩名为 ISMFILE 的 ZIP 文件中包含的 ISM。 ISM 将解压缩到在使用 `--pack` 命令创建 `ISMFILE` 时指定的 `ISMDIR`。以下示例使用在 `--pack` 示例中创建的 `ISMFILE`：

Unix：

```
% ls
spooon/ tick-3.14.ism
% rm -rf spooon
% ls
tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% ls
spooon/ tick-3.14.ism
```

Windows：

```
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% rmdir /s /q spoon
% dir
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

ISMTool 构建命令

本节描述了用于构建和修改 ISM 的 ISMTool 命令。

--verbose

显示额外调试信息。

--banner

禁止显示输出标题。

--clean

清理因构建而生成的所有文件。这会删除临时文件和所有构建产品。

--build

构建 ISM，在 `pkg` 子目录中创建包。

构建命令的主要用途是创建包含在 ISM 中的包。构建命令还可以选择调用源代码编译并运行构建前和构建后脚本。

--upgrade

升级 ISM 使其与当前安装的 ISMTool 版本相匹配。

ISMTool 的新发布版可以修复缺陷，或者修改它对提取的 ISMDIR 的操作方式。如果当前安装的 ISMTool 版本不同于创建 ISM 的 ISMTool 版本，则开发人员可能需要执行一些操作。请注意，不允许发生次要和主要降级。例如，如果 ISMTool 2.0.0 版本创建了 ISM，则 ISMTool 1.0.0 版本将不能处理该 ISM。表 52 列出了如果当前安装的 ISMTool 版本与之前的版本不同，则开发人员应采取的操作。

表 52　ISMTool 升级操作

当前安装的 ISMTool 版本	用于创建 ISM 的 ISMTool 版本	开发人员操作
1.0.1	1.0.0	修补程序号增大。不需要开发人员执行操作。这被视为简单的自动向前和向后兼容升级。
1.0.0	1.0.1	修补程序号减小。自动降级。不需要任何操作。
1.1.0	1.0.0	次要版本号增大。开发人员必须向 ISM 应用 `--upgrade` 命令。操作上可能略有差异，或者功能可能会略有增强。警告：此操作不可撤消。次要升级会设计得尽量透明。
2.0.0	1.0.0	主要版本号增大。开发人员必须向 ISM 应用 `--upgrade` 命令。操作上可能会有很大差异。开发人员可能需要执行发布说明中指定的其他操作。
1.0.0	2.0.0 或 1.1.0	主要版本号或次要版本号减小。不允许这种降级路径。无法使用安装的 ISMTool 版本处理 ISM。

--name STRING

将 ISM 的内部名称更改为 `STRING`。提取的 ISM 的顶级目录 `ISMDIR` 的名称可以与 ISM 的内部名称不同。要更改这两个名称，可以使用 ISMTool `--name` 命令更改内部名称，并使用文件系统命令更改目录名称。如果 STRING 格式对于本地打包引擎无效，直到发出 `--build` 命令并且打包引擎抛出一个错误，您才会发现问题。

--version STRING

更改 ISM 的内部版本字段。 STRING 不能包含空格。 `--version` 命令不对 STRING 格式执行任何其他检查。如果 STRING 格式对于本地打包引擎无效，直到发出 `--build` 命令并且打包引擎抛出一个错误，您才会发现问题。

--prefix PATH

更改 ISM 的安装前缀。 `PATH` 由 ISMTool 的 “从源代码构建” 功能使用，还由打包引擎的驱动程序使用。在托管服务器上进行安装时， ISM 中打包的应用程序文件安装在相对于 `PATH` 的位置。在 SA 客户端中， `PATH` 显示在包属性的 “安装路径” 字段中。在以下 Unix 示例中，开发人员使用这个 `.tar` 文件开始：

```
% tar tvf ntp/bar/ntp.tar
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 etc/ntp.conf
drwxr-xr-x ntp/ntp           0 2003-07-08 16:22:38 etc/ntp/
-rw-r--r-- root/root        22 2002-11-22 09:22:08 etc/ntp/step-tickers
-rw-r--r-- ntp/ntp           7 2003-07-08 16:22:38 etc/ntp/drift
-rw------- root/root       266 2001-09-05 03:54:42 etc/ntp/keys
-rwxr-xr-x root/root    252044 2001-09-05 03:54:43 usr/sbin/ntpd
-rwxr-xr-x root/root     40460 2001-09-05 03:54:43 usr/sbin/ntpdate
-rwxr-xr-x root/root     70284 2001-09-05 03:54:43 usr/sbin/ntpdc
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntp-genkeys
-rwxr-xr-x root/root     66892 2001-09-05 03:54:43 usr/sbin/ntpq
-rwxr-xr-x root/root     12012 2001-09-05 03:54:43 usr/sbin/ntptime
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntptimeset
-rwxr-xr-x root/root     19244 2001-09-05 03:54:43 usr/sbin/ntptrace
-rwxr-xr-x root/root      1019 2001-09-05 03:54:39 usr/sbin/ntp-wait
```

在本例中， `--prefix '/'` 将构建一个应用程序包，使所有文件都安装在相对于文件系统根目录的路径中。

```
% ismtool --build --prefix '/' --pkgengine rpm4 ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-1.i386.rpm
drwxr-xr-x   2 ntp   ntp        0 Jul  8 16:22 /etc/ntp
-rw-r--r--   1 root  root    1808 Nov 22  2002 /etc/ntp.conf
-rw-r--r--   1 ntp   ntp        7 Jul  8 16:22 /etc/ntp/drift
-rw-------   1 root  root     266 Sep  5  2001 /etc/ntp/keys
-rw-r--r--   1 root  root      22 Nov 22  2002 /etc/ntp/step-tickers
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/sbin/ntp-genkeys
-rwxr-xr-x   1 root  root    1019 Sep  5  2001 /usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root  252044 Sep  5  2001 /usr/sbin/ntpd
-rwxr-xr-x   1 root  root   40460 Sep  5  2001 /usr/sbin/ntpdate
-rwxr-xr-x   1 root  root   70284 Sep  5  2001 /usr/sbin/ntpdc
-rwxr-xr-x   1 root  root   66892 Sep  5  2001 /usr/sbin/ntpq
-rwxr-xr-x   1 root  root   12012 Sep  5  2001 /usr/sbin/ntptime
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/sbin/ntptimeset
-rwxr-xr-x   1 root  root   19244 Sep  5  2001 /usr/sbin/ntptrace
```

将安装前缀更改为 '/usr/local' 非常容易：

```
% ismtool --build --prefix '/usr/local' ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-2.i386.rpm
drwxr-xr-x   2 ntp   ntp        0 Jul  8 16:22 /usr/local/etc/ntp
-rw-r--r--   1 root  root    1808 Nov 22  2002 /usr/local/etc/ntp.conf
-rw-r--r--   1 ntp   ntp        7 Jul  8 16:22 /usr/local/etc/ntp/drift
-rw-------   1 root  root     266 Sep  5  2001 /usr/local/etc/ntp/keys
-rw-r--r--   1 root  root      22 Nov 22  2002 /usr/local/etc/ntp/step-
tickers
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/local/usr/sbin/ntp-
genkeys
-rwxr-xr-x   1 root  root    1019 Sep  5  2001 /usr/local/usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root  252044 Sep  5  2001 /usr/local/usr/sbin/ntpd
-rwxr-xr-x   1 root  root   40460 Sep  5  2001 /usr/local/usr/sbin/ntpdate
```

```
-rwxr-xr-x   1 root  root   70284 Sep  5  2001 /usr/local/usr/sbin/ntpdc
-rwxr-xr-x   1 root  root   66892 Sep  5  2001 /usr/local/usr/sbin/ntpq
-rwxr-xr-x   1 root  root   12012 Sep  5  2001 /usr/local/usr/sbin/ntptime
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/local/usr/sbin/
ntptimeset
-rwxr-xr-x   1 root  root   19244 Sep  5  2001 /usr/local/usr/sbin/ntptrace
```

在 Windows 上，没有标准方法来告知 MSI 安装到何处。因此，从 bar 目录中的 MSI 文件构建的应用程序包将忽略 --prefix 设置。但对于从 ZIP 文件构建的 Windows 应用程序包，ISMTool 将使用 --prefix 设置。在 Windows 上，前缀必须采用以下格式：driveletter:\directoryname（例如， D:\mydir）。在 Windows NT4 上， --prefix 是必需的，并且不得包含变量。

在 Unix 上， PATH 的默认值是 /usr/local。

--ctlprefix PATH

更改控制文件的安装前缀。请注意，不推荐使用此命令，您应使用默认值。在托管服务器上进行安装时，ISM 中打包的控制文件安装在相对于 PATH 的位置。在 SA 客户端中，PATH 显示在包属性的“安装路径”字段中。在 Windows 上，前缀必须采用以下格式：driveletter:\directoryname（例如， D:\mydir）。在 Windows NT4 上， --ctlprefix 是必需的，并且不得包含变量。

PATH 的默认值如下：

Unix：

/var/opt/OPSWism

Windows：

%ProgramFiles%\OPSWism

在 Solaris 上，如果指定 --ctlprefix，系统将提示您输入共享运行时包的名称。

--user STRING （仅限 Unix）

将应用程序包中文件的 Unix 用户所有者更改为 STRING。在托管服务器上安装包中的文件时，这些文件将归指定的 Unix 用户所有。

--group STRING （仅限 Unix）

将应用程序包中文件的 Unix 组所有者更改为 STRING。

--ctluser STRING （仅限 Unix）

将控制包中文件的 Unix 用户所有者更改为 STRING。默认值为 root。在托管服务器上安装包中的文件时，这些文件将归指定的 Unix 用户所有。

--ctlgroup STRING （仅限 Unix）

将控制包中文件的 Unix 组所有者更改为 `STRING`。默认值为 `bin`。

--pkgengine STRING （仅限 Unix）

更改本地打包引擎。在提供多个打包引擎的系统上，使用此命令在这些引擎之间切换。要查看可用引擎，则发出 `--help` 或 `--env` 命令。

请注意，如果更改本地打包引擎，则在 `--upload` 操作期间不会将任何包添加到软件策略。

--ignoreAbsolutePaths BOOL （仅限 Unix）

忽略存档中的绝对路径。例如，下面是具有绝对路径的二进制存档：

```
% tar tvf test/bar/foo.tar
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 /foo/bar/baz.conf
```

如果 `--prefix` 设置为 `/usr/local`，安装路径就会引起歧义：ISMTool 应该将 `baz.conf` 安装为 `/foo/bar/baz.conf` 还是 `/usr/local/foo/bar/baz.conf`？如果答案是 `/foo/bar/baz.conf`，则开发人员必须将 ISM 的 `--prefix` 设置为 `'/'`。但如果答案是 `/usr/local/foo/bar/baz.conf`，则开发人员必须指定 `--ignoreAbsolutePaths` 命令。

--addCurrentPlatform （仅限 Unix）

将当前平台添加到 ISM 的支持列表中。注意：此命令不会使 ISM 跨平台。ISM 可以在 SA 支持的不同平台上构造。平台是操作系统类型与版本的组合。示例平台包括：Redhat-Linux-7.2、SunOS-5.9、 Windows-2000。要查看 ISM 当前支持的平台，请使用 `--info` 命令。

--removeCurrentPlatform （仅限 Unix）

从 ISM 支持的平台列表中删除当前平台。

--addPlatform TEXT （仅限 Unix）

将 TEXT 指定的平台添加到 ISM 支持的平台列表中。因为平台支持和标识是动态的，因此不会对 `--addPlatform` 进行错误检查。所以建议使用 `--addCurrentPlatform` 而不是 `--addPlatform`。

--removePlatform TEXT （仅限 Unix）

将 TEXT 指定的平台从 ISM 支持的平台列表中删除。

--target STRING （仅限 Unix）

警告：此命令只应由专家使用。

允许为 RPM 打包引擎跨平台打包应用程序包。`--target` 命令必须与 `--skipControlPkg` 一起使用。 `STRING` 的格式为 `<arch-os>`，例如 `i686-linux` 或 `sparc-solaris2.7`。

--skipControlPkg BOOL

禁止构建控制包。此命令允许 ISMTool 支持打包不需要结构化应用程序控制包的文件。

--skipApplicationPkg BOOL

禁止构建应用程序包。此命令允许 ISMTool 支持创建仅控制 ISM 包。此功能可用于为不使用 ISMTool 安装或打包的应用程序构建控制器。例如核心操作系统功能的控制器、无法打包的当前运行应用程序和专用硬件的控制器。

--chunksize BYTES （仅限 Unix）

限制插入到应用程序包的字节数。（使用启发式算法来补偿压缩因子。）二进制存档 (`bar`) 目录可以包含许多用于构建应用程序包的存档。如果超出块区大小，则应用程序存档将分组到几个 bin 中，而每个 bin 又成为一个子应用程序包。算法是标准的启发式装箱算法。可移动的单元是 `bar` 目录中的二进制存档。

例如，假设输出包格式为 RPM 并且有 5 个二进制存档：`a.tgz` (100M)、 b.tgz (100M)、 `c.tgz` (200M)、`d.tgz` (300M) 和 e.tgz (50M)。如果块区大小设置为 314572800 (300M)，则输出应用程序 bin 为：

```
part1( a.tgz, b.tgz, e.tgz ) == 250M
part2( c.tgz )               == 200M
part3( d.tgz )               == 300M
```

这会生成三个应用程序包：

```
foobar-part0-1.0.0.i386.rpm
foobar-part1-1.0.0.i386.rpm
foobar-part2-1.0.0.i386.rpm
```

一般情况下，除非应用程序包大小接近千兆字节，否则块区大小都不会有问题。如果应用程序包大小接近千兆字节，一些包引擎会开始中断。默认块区大小为 1 千兆字节 （2 ^30 字节）。

--solpkgMangle BOOL （仅限 SunOS）

禁止 ISMTool 为符合 Solaris 要求而更改应用程序包的名称。有关详细信息，请参见 Solaris 差别（第 68 页）。

在创建 Solaris 包时，ISMTool 必须使用不超过 9 个字符的包名称。但在 --build 过程中，可能需要禁止 ISMTool 更改（“改变”）包名称。如果将 --solpkgMangle 指定为 false，则在创建应用程序包时， ISMTool 将使用 ISM 名称。将继续改变控制包名称。请注意，当 --solpkgMangle 为 false 时， ISM 名称不得超过 9 个字符，并且不能有多个应用程序包。

--embedPkgScripts BOOL

将 ISM 打包脚本 （安装挂接）的内容嵌入到应用程序包中。此选项必须与 --skipControlPkg 和 --skipRunTimePkg 一起使用。

默认情况下，构建应用程序包以调用控制包安装的 ISM 打包脚本。--embedPkgScripts 选项通过将 ism/pkg 目录中脚本的内容嵌入到应用程序包来覆盖此行为。在应用程序包的安装 / 卸载前以及安装 / 卸载后，将会调用这些脚本。

如果不需要 ism/pkg 目录中的一个或多个脚本，则在 --build 过程前将其删除。请注意， RPM 和 LPP 打包引擎没有 checkinstall 阶段，所以在构建 RPM 和 LPP 时会忽略 ism_check_install 文件。

--skipRuntimePkg BOOL

指定是否在后续 --build 操作中构建运行时包。

默认情况下会构建运行时包。如果将 --skipRuntimePkg 指定为 **true**，则不会在后续操作中构建运行时包，除非将 --skipRuntimePkg 指定为 **false**。如果找不到运行时包， ISM 实用程序 （如 parameters 接口）将会失败。除非您确定运行时包在将安装 ISM 的托管服务器上已存在，否则不要将 --skipRuntimePkg 指定为 **true**。

ISMTool 接口命令

本节描述了与 SA 交互的 ISMTool 命令。

--upload

将 ISMDIR 中包含的 ISM 上载到 --opswpath 指定的软件策略。如果指定的软件策略不存在，在上载过程中会自动创建此软件策略。要指定要连接的 SA 核心，可使用命令行参数 （如 --softwareRepository）或表 53 中列出的环境变量。

--upload 命令会提示您输入 SA 用户名和密码。在执行上载操作之前，必须使用 ismusertool 向该用户授权。另外，此用户必须对包含软件策略的文件夹具有写入权限。

--noconfirm

禁止显示要求 y 或 n 回复的确认提示。例如， ISMTool 具有以下确认提示：

```
Do you wish to proceed with upload?[y/n]:
```

如果设置了 --noconfirm，则不显示提示，并且 ISMTool 按照答案为 y 运行。--noconfirm 选项仅影响 ISMTool 的当前调用。

--opswpath STRING

指定将包含上载的 ISM 的软件策略的路径。请注意，路径始终包含正斜杠，即使在 Windows 上也是如此。

如果指定的软件策略不存在，在上载过程中会自动创建此软件策略。如果指定文件夹 （不是由策略结束的路径），由于您无法将 ISM 上载到文件夹中，将会发生错误。

ISMTool 支持构造跨平台 ISM。此类 ISM 的示例是网络时间协议 (NTP) 守护程序，它可以在各种平台上从源代码构建。为使跨平台 ISM 的上载更加轻松，ISMTool 支持在 --opswpath STRING 中进行变量替换。这些变量代表 ISM 的内部设置。表 53 列出了 ISMTool 识别的变量。

表 53　ISMTool 变量

变量	示例
${NAME}	ntp
${VERSION}	4.1.2
${APPRELEASE}	3
${CTLRELEASE}	7
${PLATFORM}	Redhat Linux 7.2
${OSTYPE}	Redhat Linux
${OSVERSION}	7.2

Unix 示例：

```
% ismtool --opswpath '/System Utilities/${NAME}/${VERSION}/${PLATFORM}'
```

可能的扩展：

```
'/System Utilities/ntp/4.1.2/Redhat Linux 7.2'
```

Windows 示例：

```
% ismtool --opswpath "/System Utilities/${NAME}/${VERSION}/${PLATFORM}"
ntp
```

可能的扩展：

```
"/System Utilities/ntp/4.1.2/Windows 2000"
```

--commandCenter HOST[:PORT]

要上载到文件夹，请使用位于 HOST[:PORT] 的 Opsware 命令中心核心组件。

--dataAccessEngine HOST[:PORT]

要进行上载，可使用位于 HOST[:PORT] 的 SA 数据访问引擎。

--commandEngine HOST[:PORT]

要进行上载，可使用位于 HOST[:PORT] 的 SA 命令引擎。

--softwareRepository HOST[:PORT]

要进行上载，可使用位于 HOST[:PORT] 的 SA 软件数据库。

--description TEXT

提供 ISM 的描述性文本。在上载时，此文本复制到关于软件策略的描述字段。

--addParam STRING

将参数 STRING 添加到 ISM。通常还指定命令 --paramValue、--paramDesc 和 --paramType。例如：

```
% ismtool --addParam NTP_SERVER \
          --paramValue 127.0.0.1 \
          --paramType 'String' \
          --paramDesc 'NTP server, default to loopback' ntp

% ismtool --addParam NTP_CONF_TEMPLATE \
          --paramValue /some/path/ntp.conf.template \
          --paramType 'Template' \
          --paramDesc 'Template for the /etc/ntp.conf file' ntp
```

--paramValue TEXT

设置参数的默认值。还必须指定 --addParam 命令。如果参数类型为 'String'，则值为 TEXT 指定的字符串。如果参数类型为 'Template'，则 TEXT 解释为配置模板文件的 PATH。模板文件中的数据作为默认值加载。如果未指定 --paramValue 和 --paramType，则默认值为空字符串。

--paramType PARAMTYPE

设置参数的类型。还必须指定 `--addParam` 命令。`PARAMTYPE` 必须为 `'String'` 或 `'Template'`。默认类型为 `'String'`。

--paramDesc TEXT

设置参数的描述性文本。还必须指定 `--addParam` 命令。默认值为空字符串。

--removeParam STRING

删除参数 STRING。

--rebootOnInstall BOOL

使用 SA 包控制标志 `reboot_on_install` 为应用程序包做标记。如果 `--rebootOnInstall` 设置为 true，则将在安装包后重新启动托管服务器。如果 ISM 有多个应用程序包，则对列表中的最后一个包做标记。

--rebootOnUninstall BOOL

使用 SA 包控制标志 `reboot_on_uninstall` 为应用程序包做标记。如果 `--rebootOnUninstall` 设置为 true，则将在卸载包后重新启动托管服务器。如果 ISM 有多个应用程序包，则对列表中的第一个包做标记。

--registerAppScripts BOOL （仅限 Windows）

向应用程序包注册 ISM 打包脚本 （安装挂接）。

默认情况下， ISM 打包脚本使用应用程序 MSI 进行编码，在安装前、安装后、卸载前和卸载后运行。当指定 `--registerAppScripts` 时，在上载过程中将 ISM 打包脚本注册为 SA 包控制脚本。包控制脚本注册在模型库中，可从 HP Server Automation 客户端查看。

如果 ISM 打包脚本包含与应用程序 MSI 安装冲突的操作，则需要使用 `--registerAppScripts` 命令。例如，如果后安装脚本包含对 `msiexec.exe` 的调用，则会发生冲突。由于 Microsoft 安装程序不允许并行安装，因此包含 `msiexec.exe` 调用的脚本将无法成功完成。通过将 ISM 打包脚本注册为 SA 包控制脚本，将在 MSI 安装和卸载外调用这些脚本。

--endOnPreIScriptFail BOOL （仅限 Windows）

向应用程序包注册，结束后续安装。

如果 `--endOnPreIScriptFail` 和 `--registerAppScripts` 均设置为 true，在 ISM 预安装脚本返回非零退出代码时安装将中止。

--endOnPstIScriptFail BOOL （仅限 Windows）

向应用程序包注册，结束后续安装。

如果 --endOnPstIScriptFail 和 --registerAppScripts 均设置为 **true**，在 ISM 后安装脚本返回非零退出代码时安装将中止。

--endOnPreUScriptFail BOOL （仅限 Windows）

向应用程序包注册，结束后续卸载。

如果 --endOnPreUScriptFail 和 --registerAppScripts 均设置为 **true**，在 ISM 预卸载脚本返回非零退出代码时卸载将中止。

--endOnPstUScriptFail BOOL （仅限 Windows）

向应用程序包注册，结束卸载。

如果 --endOnPstUScriptFail 和 --registerAppScripts 均设置为 **true**，在 ISM 卸载后脚本返回非零退出代码时卸载将中止。

--addPassthruPkg {PathToPkg} --pkgType {PkgType} ISMNAME

指定 {PathToPkg} 标识的包应被视为中继包。支持的包类型 {PkgType} 取决于平台，如表 54 所示。

{PathToPkg} 可以是包的完整或相对路径，但在指定 --addPassthruPkg 选项时该包必须存在。{PathToPkg} 无法在 ISM 的当前目录结构中指定包。例如，控制包、应用程序包或 bar 目录中的包无法指定为中继包。

请注意，默认情况下，上载操作不将中继包 （由 --addPassthruPkg 指定）添加到软件策略。要添加中继包，必须指定 --attachPkg 选项。

如果上载 Solaris 中继包，则不会上载响应文件。必须手动上载响应文件。

下表列出了每种平台允许的 {PkgType} （包类型）值。

表 54　中继选项支持的包类型

平台 （操作系统）	{Pkgtype} 允许的值
AIX	lpp rpm zip
HP-UX	depot zip
Linux	rpm zip

表 54　中继选项支持的包类型 （续）

平台 （操作系统）	{Pkgtype} 允许的值
SunOS	rpm solcluster solpatch solpkg ips zip
Windows	hotfix msi sp zip

以下示例显示了如何将中继包添加到 ISM 并指定要添加到软件策略的包：

```
% ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
Inspecting specified package:...
bos.rte.libs.5.1.0.50.U (lpp)
  bos.rte.libs-5.1.0.50 (update fileset)
  IY42527 (apar)
Done.
% ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

--removePassthruPkg {PassthruPkgFileName} ISMNAME

指定已注册的中继包不再是中继包。

ISMTool 将执行以下操作：

1. 从 ISM 目录结构删除 {PassthruPkgFileName}。
2. 在 ism.conf 中记录 {PassthruPkgFileName} 不再是中继包。
3. 在下一次上载及所有后续上载中，如果包添加到 --opswpath 软件策略，包将会被删除。

请注意，ISM 会记住所有作为中继包删除的包。如果包通过 SA 客户端或先前的上载操作添加到软件策略，该包将在下一次上载操作中从策略中删除。

--attachPkg {PkgName} --attachValue BOOLEAN ISMNAME

指定 ISM 托管的包是否应添加到 --opswpath 标识的软件策略。

默认情况下，在构建控制包或应用程序包时，这些类型的包会标记为添加到软件策略。但在指定 --attachPkg 选项之前，中继包和内部包不会标记为添加。

{PkgName} 是 --showPkgs 命令所列的包的名称。如果 --attachValue 为 true，包将标记为添加到软件策略。如果 --attachValue 为 false，包将上载到软件数据库中，但不会添加到软件策略。如果 --attachValue 为 false，并且包已经位于软件策略中，则包将标记为从策略中删除。将在 --upload 操作中添加或删除包。下表列出了可以添加到软件策略的包类型。

表 55　包类型属性

包类型	这种包类型是否可以包含脚本？	这种包类型是否可以添加到软件策略？
AIX LPP	否	否
AIX 基础文件集	是	是
AIX 更新文件集	是	是
AIX APAR	否	是
HP-UX 软件仓库	否	否
HP-UX 文件集	是	是
HP-UX 修补程序文件集	否	否
HP-UX 产品	否	是
HP-UX 修补程序产品	否	是
IPS 包	否	是
RPM	是	是
Solaris 包	否	否
Solaris 包实例	是	是
Solaris 修补程序	是	是
Solaris 修补程序群集	否	是
Windows 修复程序	是	是
Windows MSI	是	是
Windows Service Pack	是	是
Windows ZIP 文件	是	是

--orderPkg {PkgName} --orderPos {OrderPos} ISMNAME

更改 ISM 托管的附加包的安装顺序。

{OrderPos} 是一个整数，指定 {PkgName} 标识的包的新安装顺序。 {OrderPos} 是 1（而不是 0）或第一个要安装的包。要显示安装顺序，请使用 `ismtool --showOrder` 命令。

以下示例显示了如何显示和更改安装顺序：

```
% ismtool --showOrder ISMNAME
[1] test-ism-1.0.0-1.rpm
[2] test-1.0.0-1.rpm
[3] bos.rte.libs-5.1.0.50
```

```
[4] IY42527

% ismtool --orderPkg IY42527 --orderPos 1 ISMNAME
[1] IY42527
[2] test-ism-1.0.0-1.rpm
[3] test-1.0.0-1.rpm
[4] bos.rte.libs-5.1.0.50
```

--addPathProp {PathProp} --propValue {PropValue} ISMNAME

指定软件策略的属性（元数据）值。

要显示当前值，请使用 --showPathProps 命令。下表列出了 --addPathProp 命令允许的值和类型。

表 56 {PathProp} 允许的值

{PathProp} 允许的值	{PropValue} 类型	示例
description	文本	'This does something important'
已弃用： notes	文本	'And so does this'
已弃用： allowservers	布尔值	false

以下示例命令显示了如何设置 description 属性：

```
% ismtool --addPathProp description --propValue 'This policy does something'
ISMNAME
% ismtool --showPathProps ISMNAME
description:   This policy does something
```

--editPkg {PkgName} --addPkgProp {PkgProp} --propValue {PropValue} ISMNAME

为给定的包元数据属性指定值。

{PkgName} 标识要更新的包；它可以是使用 --showPkgs 命令列出的任意包名称。下表列出了 {PkgProp} 允许的值。

表 57 {PathProp} 允许的值

{PkgProp} 允许的值	描述	{PropValue} 类型
deprecated	包的已弃用状态	布尔值
description	包的描述	文本
endonpreiscriptfail	预安装脚本失败时修正结束	布尔值
endonpreuscriptfail	预卸载脚本失败时修正结束	布尔值

表 57　{PathProp} 允许的值

{PkgProp} 允许的值	描述	{PropValue} 类型
endonpstiscriptfail	后安装脚本失败时修正结束	布尔值
endonpstuscriptfail	后卸载脚本失败时修正结束	布尔值
installflags	包的安装标志	文本
notes	包的备注	文本
rebootoninstall	包要求在安装后重新启动	布尔值
rebootonuninstall	包要求在卸载后重新启动	布尔值
uninstallflags	包的卸载标志	文本

只有在为包定义了预安装 / 后安装 / 预卸载 / 后卸载脚本时，才会设置 endonXXXscriptfail 值。这些脚本位于 ISMNAME/pad 子目录中。

请注意，并非所有包类型都支持前面表中列出的所有 {PkgProp} 值。每种包类型支持的 {PkgProp} 值可以通过查看 SA 客户端中的包属性详细信息看到。此外，下表还列出了特定包类型支持的 {PkgProp} 值。

表 58　包类型允许的 {PkgProp} 值

{PkgProp} 允许的值	包类型	描述	{PropValue}
upgradeable	RPM	包可以升级	布尔值
productname	Windows MSI	MSI 产品名称	字符串
productversion	Windows MSI	MSI 版本号	字符串
servicepacklevel	Windows OS Service Pack	Service Pack 版本号	整数
installdir	Windows ZIP	安装目录	字符串
postinstallscriptfilename	Windows ZIP	后安装脚本文件名	字符串
postinstallscriptfilenamefail	Windows ZIP	后安装脚本失败时修正结束	布尔值
preuninstallscriptfilename	Windows ZIP	预卸载脚本文件名	字符串

表 58　包类型允许的 {PkgProp} 值

{PkgProp} 允许的值	包类型	描述	{PropValue}
preuninstallscriptfilenamefail	Windows ZIP	预卸载脚本失败时修正结束	布尔值

在执行 --upload 操作之前，必须设置 productversion、 productname 和 servicepacklevel。在 --upload 操作后 productname 和 productversion 无法更改。如果在修改 productname 或 productversion 后再执行其他 --upload 操作，将不会应用修改的值。

以下示例显示了如何指定包的描述：

```
% ismtool --editPkg bos.rte.libs.5.1.0.50 --addPkgProp description --
propValue 'This is a fileset' ISMNAME
```

ISMTool 环境变量

ISMTool 引用本节描述的 shell 环境变量。

CRYPTO_PATH

此环境变量指示包含文件 ismtool/token.srv 的目录。

只有从 Windows 托管服务器或非 SA 管理的服务器 （即没有服务器代理的服务器）上载 ISM 时，才需要 CRYPTO_PATH 和 token.srv。要在上载 ISM 时连接到 SA 核心，ISMTool 需要安装 Server Automation 时生成的客户端证书 （token.srv 文件）。

请记住，将此证书与 ISMTool 一起使用会调用另一种安全机制，而不是 SA 客户端使用的安全机制。因此，您的权限可能会增加，也可能会减少。您可能有权访问属于客户的一些服务器，而通常情况下您是无权访问这些服务器的。另外，您可能能够执行无法使用 SA 客户端执行的操作。因此，在这种情况下使用 ISMTool 要小心谨慎，以免安全权限的可能变更引起意想不到的后果。

要获取 token.srv 文件并设置 CRYPTO_PATH 环境变量，请执行以下步骤：

1. 以 root 用户身份登录到核心服务器并找到以下文件：

   /var/opt/opsware/crypto/agent/agent.srv

2. 将 agent.srv 复制到安装了 IDK 的服务器上的以下文件中：

   <some-path>/ismtool/token.srv

   目录路径的 <some-path> 部分由您选择，但包含 token.srv 的子目录必须为 ismtool。

3. 将 CRYPTO_PATH 环境变量设置为 <some-path>，即 ismtool/token.srv 上层的目录。

例如，在 Unix 服务器上，假设 token.srv 的完整路径名称如下：

```
/home/jdoe/dev/crypto/ismtool/token.srv
```

在 csh 中，如下所示设置环境变量：

```
setenv CRYPTO_PATH /home/jdoe/dev/crypto
```

在 Windows 上， token.srv 或许位于以下位置：

```
C:\jdoe\dev\crypto\ismtool\token.srv
```

可以如下所示设置环境变量：

```
set CRYPTO_PATH=C:\jdoe\dev\crypto
```

ISMTOOLBINPATH

此环境变量是用冒号分隔的目录名称列表， ISMTool 在此处搜索系统级别工具 （例如 tar 和 cpio）。将使用以下搜索策略：

1. 从环境变量 ISMTOOLBINPATH 搜索路径。
2. 在 /usr/local/ismtool/lib/tools/bin 中搜索编译好的二进制文件 （如果有）。
3. 在用户路径中进行搜索。

ISMTOOLCC

此环境变量是 ISMTool 上载到文件夹期间使用的 Opsware 命令中心核心组件的 HOST[:PORT]。

ISMTOOLCE

此环境变量是 ISMTool 使用的 SA 命令引擎的 HOST[:PORT]。

ISMTOOLDA

此环境变量是 ISMTool 使用的 SA 数据访问引擎的 HOST[:PORT]。

ISMTOOLPASSWORD

此环境变量是一个字符串，在 ISMTool 上载期间指定 SA 密码。

ISMTOOLSITEPATH

此环境变量是 “站点” 目录的路径。

ISMTool 包含创建新 ISM 时引用的某些默认脚本和特性值（例如，安装前缀）。开发人员可以使用站点目录覆盖默认脚本和选定的一组特性值。

defaults.conf 文件

在站点目录中，开发人员可以创建 defaults.conf 文件，其中包含特性值的覆盖值。defaults.conf 中的行采用以下格式：<tag>:<value>。以 # 字符开头的行是注释。以下示例显示了可以在 defaults.conf 中设置的值：

Unix：

```
prefix:    /usr/local
ctlprefix: /var/opt/OPSWism
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
ctluser:   root
ctlgroup:  bin
```

Windows：

```
prefix:    ???
ctlprefix: ???
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
```

templates 子目录

开发人员可以覆盖 /usr/local/ismtool/lib/ismtoollib/templates 目录中的文件，方法是将自己的副本放在 ISMTOOLSITEPATH 内的 templates 子目录中。例如，开发人员可以覆盖 Windows 或 Unix 的默认打包挂接文件。

control 子目录

有时，开发人员需要将一套常用工具安装到 ISM 的 control 目录。 ISMTool 通过将所有文件从 ISMTOOLSITEPATH 的 control 子目录复制到 ISM 的 control 目录来支持这种要求。如果文件已存在于 ISM 的 control 目录中，则不会覆盖。

ISMTOOLSR

此环境变量是 ISMTool 使用的 SA 软件数据库的 HOST[:PORT]。

ISMTOOLUSERNAME

此环境变量是一个字符串，在 ISMTool 上载期间指定 SA 用户名。

ISMUserTool

ISMTool 的 `--upload` 命令会提示您输入 SA 用户名。要使 SA 用户能够执行上载，请运行 ISMUsertool 来分配权限。

要列出具有上载权限的用户：

```
% ismusertool --showUsers
```

要向用户授予用户上载权限：

```
% ismusertool --addUser johndoe
```

要撤销上载权限：

```
% ismusertool --removeUser johndoe
```

ISMUsertool 允许您在一个命令行中指定多个选项。有关详细信息，请指定 `--help` 选项：

```
% ismusertool --help
```

默认情况下， Opsware `admin` 用户具有无法撤销的上载权限。

文件夹在 Server Automation 6.0 版中是新文件夹。要将 ISM 上载到文件夹中，用户必须具有文件夹权限。默认情况下， `admin` 用户没有文件夹权限。在生产环境中， `admin` 不应有文件夹权限，因此您不应使用 `admin` 进行上载。