在 Oracle® Solaris 11.1 中使用映像包管理系统打包和交付软件

文件号码 E36661–02
2012 年 10 月

ORACLE

121210@25097

目录

前言

《在 Oracle Solaris 11.1 中使用映像包管理系统打包和交付软件》介绍了如何使用 Oracle Solaris 映像包管理系统 (IPS) 功能为 Oracle Solaris 11 操作系统 (operating system, OS) 创建软件包。

目标读者

本手册的目标读者是要创建可使用 IPS 在 Oracle Solaris 11 OS 上安装和维护的软件包的软件开发者，以及希望更好地了解 IPS 以及如何使用 IPS 打包 Oracle Solaris OS 的开发者和系统管理员。讨论了底层的 IPS 设计理念，以便读者更好地了解和使用 IPS 的高级功能。

本书的结构

- 第 1 章，IPS 设计目标、概念和术语概述了 IPS 的基本设计理念及其作为软件模式的表达形式。
- 第 2 章，使用 IPS 打包软件使您可以开始构建自己的软件包。
- 第 3 章，安装、删除和更新软件包介绍了在安装、更新和删除已安装在映像中的软件时 IPS 客户机的内部工作原理。
- 第 4 章，指定软件包相关项介绍了不同类型的 IPS 相关项以及如何使用它们来构造可运转的软件系统。
- 第 5 章，允许变量介绍了如何向最终用户提供不同的安装选项。
- 第 6 章，以编程方式修改软件包清单介绍了如何对软件包清单进行计算机编辑以自动注释和检查清单。
- 第 7 章，在软件包安装过程中自动进行系统更改介绍了如何使用服务管理工具 (Service Management Facility, SMF) 来自动处理作为软件包安装结果发生的任何必需的系统修改。
- 第 8 章，有关软件包更新的高级主题讨论了如何重命名、合并、拆分软件包，移动软件包内容，交付应用程序的多个实现以及在引导环境间共享信息。
- 第 9 章，对 IPS 软件包进行签名介绍了 IPS 软件包签名以及开发者和质量保证组织如何对新软件包或现有已经签名的软件包进行签名。

- 第 10 章，处理非全局区域介绍了 IPS 如何处理区域并讨论了打包时需要考虑非全局区域的情况。
- 第 11 章，修改已发布的软件包介绍了管理员如何按本地条件修改现有软件包。
- 附录 A，对软件包进行分类显示了软件包信息分类方案定义。
- 附录 B，如何使用 IPS 打包 Oracle Solaris OS 介绍了 Oracle 如何使用 IPS 功能打包 Oracle Solaris OS。

相关文档

- 《在 Oracle Solaris 11.1 中管理服务和故障》中的第 1 章 “管理服务（概述）”描述了 Oracle Solaris 服务管理工具 (Service Management Facility, SMF) 功能
- 《复制和创建 Oracle Solaris 11.1 软件包系统信息库》
- 《添加和更新 Oracle Solaris 11.1 软件包》
- 《创建和管理 Oracle Solaris 11.1 引导环境》和 `beadm(1M)` 手册页
- 《安装 Oracle Solaris 11.1 系统》
- 《Oracle Solaris 管理：Oracle Solaris Zones、Oracle Solaris 10 Zones 和资源管理》
- 《Oracle Solaris 11.1 管理：ZFS 文件系统》

获取 Oracle 支持

Oracle 客户可以通过 My Oracle Support 获取电子支持。有关信息，请访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info`，或访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs`（如果您听力受损）。

印刷约定

下表介绍了本书中的印刷约定。

表 P–1 印刷约定

字体或符号	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `machine_name% you have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`machine_name%` **`su`** `Password:`

表 P–1 印刷约定 （续）

字体或符号	含义	示例
aabbcc123	要使用实名或值替换的命令行占位符	删除文件的命令为 `rm` *filename*。
AaBbCc123	保留未译的新词或术语以及要强调的词	这些称为 *Class* 选项。 **注意**：有些强调的项目在联机时以粗体显示。
新词术语强调	新词或术语以及要强调的词	**高速缓存**是存储在本地的副本。 请**勿**保存文件。
《书名》	书名	阅读《用户指南》的第 6 章。

命令中的 shell 提示符示例

下表显示了 Oracle Solaris OS 中包含的缺省 UNIX shell 系统提示符和超级用户提示符。请注意，在命令示例中显示的缺省系统提示符可能会有所不同，具体取决于 Oracle Solaris 发行版。

表 P–2 shell 提示符

shell	提示符
Bash shell、Korn shell 和 Bourne shell	`$`
Bash shell、Korn shell 和 Bourne shell 超级用户	`#`
C shell	`machine_name%`
C shell 超级用户	`machine_name#`

◆ ◆ ◆ 第 1 章

1 IPS 设计目标、概念和术语

本章概述了 IPS 的基本设计理念及其作为软件模式的表达形式。

IPS 设计目标

设计 IPS 是为了消除早期软件分发、安装和维护机制中长期存在并对 Oracle Solaris 客户、开发者、维护者以及 ISV 造成严重问题的一些问题。

IPS 基本设计目标包括：

最大限度地缩短停机时间。
通过使软件更新能够在计算机正在执行生产工作时进行，最大限度地缩短计划内停机时间。

通过支持快速重新引导至可运转的已知软件配置，最大限度地缩短计划外停机时间。

自动化安装和更新。
尽可能自动化新软件的安装和对现有软件的更新。

降低介质要求。
解决软件大小日益增长以及分发介质空间有限所带来的难题。

验证软件安装是否正确。
确保能够确定是否按照软件包设计者（发布者）定义的步骤正确安装了软件包。此类检查应当不具有可欺骗性。

支持轻松虚拟化。
引入相应机制（尤其是通过使用区域）以在各种级别轻松实现 Oracle Solaris 的虚拟化。

简化升级。
减少为现有系统生成修补程序或升级程序需要做的工作。

支持轻松创建软件包。
使其他软件发布者（ISV 和最终用户自己）可以轻松创建和发布适用于 Oracle Solaris 的软件包。

这些目标引出了以下理念：

按需创建引导环境。
利用 ZFS 快照和克隆功能，根据需要动态创建引导环境。

- 由于 Oracle Solaris 11 要求将 ZFS 作为根文件系统，所以区域文件系统也需要位于 ZFS 上。
- 用户可以根据需要创建任意数量的引导环境。
- IPS 可以根据需要（在修改正在运行的系统之前出于备份目的或者为安装新版本的 OS）自动创建引导环境。

统一安装、修补和更新。
消除用于安装、修补和更新的重复机制与代码。

此理念将导致 Oracle Solaris 的维护方式发生一些重大改变，包括以下重要示例：

- 所有 OS 软件更新和修补都直接通过 IPS 来执行。
- 任何时候只要安装了新软件包，该软件包就已经是恰好合适的版本。

最大限度地降低不正确安装的机率。
软件包安装的无欺骗性验证要求可带来以下结果：

- 如果软件包需要支持以多种方式进行安装，则开发者必须指定这些方式，以便验证过程可以考虑此项。
- 脚本编写过程本身是不可验证的，因为包管理系统无法确定脚本编写者的意图。此问题与稍后讨论的其他问题一起决定了在打包操作期间禁止脚本编写。
- 软件包不能包含用来编辑其清单的任何机制，因为如果有这样的机制，验证将无法进行。
- 如果管理员希望通过不符合原始发布者的定义的方式安装软件包，则包管理系统应当能够让管理员轻松地重新发布要更改的软件包，以便明确更改范围，使更改在升级后不会丢失，并且可以采用与原始软件包相同的方式进行验证。

提供软件系统信息库。
为了不受空间大小的限制，引入了一个软件系统信息库模型，可通过几种不同的方法访问此模型。可以将不同的系统信息库源组合到一起以提供一整套软件包，并且这些系统信息库可以作为单个文件进行分发。这样，将不需要像以前那样要求在单个介质上包含所有可用软件。为支持断开连接的操作或有防火墙的操作，提供了用于复制和合并系统信息库的工具。

在软件包中包含元数据。
支持多个（可能存在竞争关系的）软件发布者这一需求将导致以下决策：将所有打包元数据存储在软件包本身中，不存在用于存储诸如所有软件包及其相关项之类信息的任何主数据库。出于性能考虑，系统信息库中包含了来自软件发布者的可用软件包的目录，但也可以通过软件包中包含的数据重新生成该目录。

软件自组装

根据上述目标和理念，IPS 引入了软件自组装的一般概念：系统上安装的软件的任何集合都应该能够在该系统引导时、打包操作完成时或软件运行时将自己构建到工作配置中。

有了软件自组装功能，IPS 中消除了在安装时编写脚本的需求。软件将负责其自己的配置，而不是依赖于包管理系统来代表软件执行该配置。软件自组装还能使包管理系统在备用映像（如当前未引导的引导环境或处于脱机状态的区域根目录）上安全运行。此外，由于自组装仅在正在运行的映像上执行，所以软件包开发者不需要应对跨版本或跨体系结构的运行时上下文。

在引导之前，必须做一些操作系统映像准备工作，IPS 以透明方式管理此过程。映像准备工作包括更新引导块、准备引导归档文件 (ramdisk) 以及在某些体系结构中管理引导选择菜单。

软件自组装工具

以下 IPS 功能和特性有助于实施软件自组装。

原子软件对象

在 IPS 中，**操作**是用于交付软件的原子单位。每个操作交付一个软件对象。该软件对象可以是文件系统对象，例如文件、目录或链接，也可以是更为复杂的软件结构，例如用户、组或驱动程序。在 SVR4 包管理系统中，这些更为复杂的操作类型通过使用类操作脚本进行处理。在 IPS 中，不需要编写脚本。

操作组合到一起成为软件包，并可以在活动映像和脱机映像中安装、更新和删除。

第 18 页中的 “软件包内容：操作”中对操作进行了更详细的论述。

配置组合

由于配置文件在打包操作期间需要开销巨大的脚本编写来更新每个配置文件，IPS 提倡交付配置文件的片段，而不是维护复杂的配置文件。打包后的应用程序在读取其配置时可以直接访问这些片段，也可以将这些片段组合成一个完整的配置文件，然后读取该文件。

Oracle Solaris 11 用户属性数据库就是一个典型的片段式配置文件。`/etc/user_attr` 配置文件用来为系统上的角色和用户配置扩展属性。在 Oracle Solaris 11 中，`/etc/user_attr` 文件仅用于本地更改。完整配置是从交付到 `/etc/user_attr.d` 目录中的各个文件中读取的。多个软件包交付了完整配置的各个片段。安装、删除或更新片段时不需要编写脚本。

这种用来组合配置文件的方法要求在编写软件时始终牢记组合这一理念，但这并不是始终可行。

用来支持组合的另一种方式是使服务将配置文件视为可变的，然后在安装、删除或更新配置的片段时重新组装该配置文件。通常，此组装由 SMF 服务执行。在接下来的章节中将进一步讨论 SMF 服务执行的组装。

执行器和 SMF 服务

执行器是一个标记，应用于包管理系统提供的所有操作，在安装、删除或更新相应操作时该标记会导致系统发生更改。这些更改通常以 SMF 服务的方式实现。

SMF 服务可以直接配置软件，也可以使用 SMF 清单中提供的数据或来自系统上所安装文件的数据来构造配置文件。

SMF 具有丰富的语法来表达相关项。每个服务只有在其所有相关项都得到满足时才能运行。

任何服务都可以将其自身作为一个相关项添加到 `svc:/milestone/self-assembly-complete:default` SMF 里程碑中。当正在引导的操作系统到达该里程碑后，所有自组装操作也应当完成了。

一种特殊类型的区域称为**不变区域**，该区域可以配置为对其文件系统的各部分具有受限制的写入权限。请参见 `zonecfg(1M)` 手册页中有关 `file-mac-profile` 的论述。要在此类型的区域中完成自组装，请引导区域进行读/写。在到达 `self-assembly-complete` SMF 里程碑后，区域将自动引导至所需的 `file-mac-profile` 设置。

Oracle Solaris 中的软件自组装的示例

以下示例介绍了作为 Oracle Solaris 的一部分交付的软件包。

Apache Web Server

Oracle Solaris 软件包中的 Apache Web Server 是一个典型的自组装示例：`pkg:/web/server/apache-22`。该软件包附带了一个缺省的 `httpd.conf` 文件，该文件中包含一个引用 `/etc/apache2/2.2/conf.d` 文件的 `Include` 指令。

```
Include /etc/apache2/2.2/conf.d/*.conf
```

其他软件包可以将新的 `.conf` 文件交付到该目录，并且每当安装、更新或删除交付此新 `.conf` 文件的软件包时，都将使用 `refresh_fmri` 执行器自动刷新 Apache 实例。刷新 Apache 实例会导致 Web 服务器重新生成其配置。

```
file etc/apache2/2.2/conf.d/custom.conf path=etc/apache2/2.2/conf.d/custom.conf \
    owner=root group=bin mode=0644 refresh_fmri=svc:/network/http:apache22
```

有关如何使用 `refresh_fmri` 执行器的信息，请参见第 37 页中的 “添加需要的任何侧面或执行器”和第 7 章，在软件包安装过程中自动进行系统更改。

交付配置片段的多个软件包

Oracle Solaris OS 中的另一个自组装示例是多个软件包将内容交付到 `/etc/security/exec_attr.d/` 目录中。

在早期 Oracle Solaris 发行版中，某个 SMF 服务会将在 `exec_attr.d` 中交付的文件合并到一个数据库 `/etc/security/exec_attr` 中。在 Oracle Solaris 11 OS 中，`libsecdb` 直接读取 `exec_attr.d` 中的片段，无需通过单独的服务来执行此合并。

其他包含 `/etc/security` 中配置的片段的目录会以类似方式进行处理。

IPS 软件包生命周期

本节简要介绍了 IPS 软件包生命周期中的每种状态。为获得最佳结果，软件包开发者和系统管理员都应该了解软件包生命周期的各个阶段。

创建 任何人都可以创建软件包。IPS 不会将任何特定的软件构建系统或目录分层结构强加给软件包设计者。有关软件包创建的详细信息，请参见第 2 章，使用 IPS 打包软件。有关软件包创建的各个方面将在本指南的剩余章节中进行讨论。

发布 软件包将发布到 IPS 系统信息库，可以是 HTTP 位置，也可以是文件系统。发布的软件包还可以转换为 `.p5p` 软件包归档文件。要访问 IPS 系统信息库中的软件，可以使用 `pkg set-publisher` 命令将该系统信息库添加到系统中，也可以通过使用带 `-g` 选项的 `pkg` 命令将系统信息库作为临时源进行访问。第 2 章，使用 IPS 打包软件中显示了一些软件包发布示例。

安装 可以将以下位置的软件包安装到系统上：通过 `http://`、`https://` 或 `file://` 等 URL 访问的 IPS 系统信息库，或者 `.p5p` 软件包归档文件。第 3 章，安装、删除和更新软件包中对软件包安装过程进行了更详细的说明。

更新 无论是发布到 IPS 系统信息库的软件包，还是作为新的 `.p5p` 软件包归档文件交付的软件包，都可能会有更新的版本可用。已安装的软件包随后可以更新到最新版本，可以分别进行更新，也可以作为整个系统更新的一部分进行更新。

请注意，IPS 不使用 SVR4 包管理系统使用的“修补”概念。对 IPS 打包的软件所做的所有更改都通过更新的软件包进行交付。

软件包更新的执行方式大致与软件包安装相同，但包管理系统已进行了优化，它仅安装更新的软件包所交付的更改部分。第 3 章，安装、删除和更新软件包中对软件包更新过程进行了更详细的说明。

重命名 在软件包的生命周期中，您可能希望重命名软件包。出于组织结构原因或为了重构软件包，可能要重命名软件包。软件包重构的示例包括将多个软件包合并到单个软件包中，或将单个软件包拆分成多个较小的软件包。

IPS 可以从容地处理在软件包之间移动的内容。IPS 还允许旧软件包名称继续存在于系统上，在用户要求安装已重命名的软件包时会自动安装新的软件包。第 10 章中会更详细地介绍软件包重命名。

过时 最终，软件包可能会走到其生命周期的尽头。软件包发布者可能会决定不再支持某个软件包并且不再对该软件包进行更新。IPS 允许发布者将此类软件包标记为已过时。

过时的软件包不能再用作其他软件包中的大多数相关项的目标，并且升级到过时版本的所有软件包都将自动从系统中删除。第 69 页中的 “重命名、合并和拆分软件包”中对软件包过时进行了更详细的说明。

删除 最后，如果任何其他软件包都没有依赖于某个软件包的相关项，则可以将该软件包从系统中删除。第 3 章，安装、删除和更新软件包中对软件包删除过程进行了更详细的说明。

IPS 术语和组件

本节定义了 IPS 术语并介绍了 IPS 组件。

可安装的映像

IPS 设计用于安装映像中的软件包。映像是一个目录树，可以根据需要挂载在各种位置。映像属于以下三种类型之一：

完整 在完整映像中，所有相关项都在映像自身内解析，IPS 以一致的方式维护这些相关项。

区域 非全局区域映像与某个完整映像（父全局区域映像）相链接，但本身不提供完整系统。在区域映像中，IPS 根据软件包中相关项的定义来维护非全局区域与其全局区域的一致性。

用户 用户映像只包含可重定位的软件包。

通常，映像是由安装程序创建或克隆的，例如 `beadm(1M)` 或 `zonecfg(1M)`，而不是通过 `pkg image-create` 创建的。

软件包标识符：FMRI

每个 IPS 软件包都通过一个故障管理资源标识符 (fault management resource identifier, FMRI) 进行标识，该标识符由发布者、名称、版本以及方案 `pkg` 组成。在以下软件包 FMRI 示例中，`solaris` 是发布者，`system/library` 是软件包名称，`0.5.11,5.11-0.175.0.0.0.2.1:20111019T082311Z` 是版本：

```
pkg://solaris/system/library@0.5.11,5.11-0.175.1.0.0.2.1:20120919T082311Z
```

如果缩写形式的 FMRI 仍然是唯一的，则可以采用缩写形式指定 FMRI。方案、发布者和版本可以省略。可以在软件包名称中省略前导组件。

- 当 FMRI 以 `pkg://` 或 `//` 开头时，`//` 后面的第一个单词必须是发布者名称，并且不能省略软件包名称中的任何一个组件。当未省略软件包名称中的任何组件时，软件包名称被视为完整的或**有根的**。
- 当 FMRI 以 `pkg:/` 或 `/` 开头时，斜杠后面的首个单词是软件包名称，并且不能省略软件包名称中的任何组件。不能提供发布者名称。
- 当省略版本时，通常会将软件包解析为可以安装的最新软件包版本。

软件包发布者

发布者是指开发和构造软件包的实体。发布者名称（或前缀）唯一地标识此来源。发布者名称可以包含大写和小写字母、数字、连字符和句点，与有效主机名所包含的字符相同。对于发布者名称，Internet 域名或注册商标是很好的选择，因为它们提供了天然的名称空间划分。

在确定打包解决方案时，软件包客户机会将给定发布者的所有指定来源的软件包合并在一起。

软件包名称

软件包名称是由任意数量的组件按层次结构组成的名称，其中组件之间由正斜杠 (/) 字符分隔。软件包名称组件必须以字母或数字开头，并且可以包含下划线 (_)、连字符 (-)、句点 (.) 和加号 (+)。软件包名称组件区分大小写。

软件包名称构成了跨发布者的单个名称空间。从外部相关项和接口的角度来看，可以认为名称和版本相同但发布者不同的软件包是可以互换的。

如果使用的软件包名称是唯一的，则可以省略软件包名称的前导组件。例如，`/driver/network/ethernet/e1000g` 可以缩写为 `network/ethernet/e1000g`、`ethernet/e1000g`，甚至只有 `e1000g`。当未省略软件包名称中的任何组件时，软件包名称被视为完整的或**有根的**。如果包管理客户机指出软件包名称具有多义性，请指定包含更多组件的软件包名称，或指定完整的有根名称。为软件包选择名称时，应尽可能降低多义性。

如果 FMRI 包含发布者名称，则必须指定完整的有根软件包名称。

脚本应通过软件包的完整有根名称来引用软件包。

还可以使用星号 (*) 来指定 FMRI 以匹配软件包名称中的任一部分。因此，`/driver/*/e1000g` 和 `/dri*00g` 均可以展开为 `/driver/network/ethernet/e1000g`。

软件包版本

软件包版本包含四个由标点符号分隔的整数序列。前三个序列中的元素由圆点分隔，各序列可具有任意长度。禁止在版本元素中使用前导零，以便能够按软件包版本进行明确排序。例如，01.1 和 1.01 是无效的版本元素。

在以下软件包版本示例中，第一个序列为 `0.5.11`，第二个序列为 `5.11`，第三个序列为 `0.175.1.0.0.2.1`，第四个序列为 `20120919T082311Z`。

```
0.5.11,5.11-0.175.1.0.0.2.1:20120919T082311Z
```

组件版本　第一个序列是组件版本。对于作为 Oracle Solaris 的一部分而开发的组件，此序列代表最后一次更改此软件包时的发行点。对于具有自己的开发生命周期的组件，此序列是一个由小圆点分隔的发行编号，例如 2.4.10。

内部版本　第二个序列是内部版本。如果提供此序列，前面必须有一个逗号。Oracle Solaris 使用此序列指示编译软件包时所针对的 OS 的发行版。

分支版本　第三个序列是分支版本，提供特定于供应商的信息。如果提供此序列，前面必须有一个连字符。此序列可以包含内部版本号或提供一些其他信息。该值可以在打包元数据发生更改时增大，独立于组件。有关 Oracle Solaris 中如何使用分支版本字段的说明，请参见第 95 页中的 “Oracle Solaris 软件包版本控制”。

时间戳　第四个序列是时间戳。如果提供此序列，前面必须有一个冒号。此序列表示在 GMT 时区中软件包的发布日期和时间。此序列在发布软件包时自动更新。

软件包版本按从左到右的优先级进行排序：紧跟在 `@` 之后的数字是版本空间中最重要的部分。时间戳是版本空间中最不重要的部分。

可使用 `pkg.human-version` 属性来保存人工可读的版本字符串，但还必须存在上述的版本控制方案。人工可读的版本字符串仅用于显示，如第 22 页中的 “设置操作”中所述。

通过允许任意版本长度，IPS 可以提供用来支持软件的各种不同模型。例如，软件包设计者可以使用内部版本或分支版本并将版本控制方案的一部分指定给安全更新，一部分用于付费和免费的支持更新，还有一部分用于次要错误修复，或用于任何需要的信息。

版本还可以是 `latest` 标记，该标记指定已知的最新版本。

附录 B，如何使用 IPS 打包 Oracle Solaris OS 介绍了 Oracle Solaris 如何实现版本控制。

软件包内容：操作

操作定义了包含软件包的软件；还定义了创建此软件组件所需的数据。软件包内容在软件包清单文件中表示为一组操作。

软件包清单主要是使用程序创建的。软件开发者提供很少信息，并且清单是使用软件包开发工具完成的，如第 2 章，使用 IPS 打包软件中所述。

操作在软件包清单文件中表示为以下形式：

action_name attribute1=value1 attribute2=value2 ...

在以下示例操作中，`dir` 表示该操作指定一个目录。以 *name*= *value* 形式表示的特性描述了该目录的属性：

```
dir path=a/b/c group=sys mode=0755 owner=root
```

以下示例显示了一个具有关联数据的操作。在此 `file` 操作中，第二个字段（没有 *name*= 前缀）称为有效负荷:

```
file 11dfc625cf4b266aaa9a77a73c23f5525220a0ef path=etc/release owner=root \
    group=sys mode=0444 chash=099953b6a315dc44f33bca742619c636cdac3ed6 \
    pkg.csize=139 pkg.size=189 variant.arch=i386
```

在此示例中，有效负荷为文件的 SHA-1 散列。此有效负荷还可以显示为名称为 `hash` 的一般属性，如以下示例中所示。如果同一操作中同时存在这两种形式，则它们必须具有完全相同的值。

```
file hash=11dfc625cf4b266aaa9a77a73c23f5525220a0ef path=etc/release owner=root \
    group=sys mode=0444 chash=099953b6a315dc44f33bca742619c636cdac3ed6 \
    pkg.csize=139 pkg.size=189 variant.arch=i386
```

操作元数据是可以自由扩展的。可以根据需要向操作添加其他属性。属性名称不能包括空格、引号或等号 (=)。属性值中可以包含所有这些符号，但是包含空格的值必须括在单引号或双引号中。位于括在双引号中的字符串内的单引号不需要进行转义，位于括在单引号中的字符串内的双引号也不需要进行转义。可在引号前使用反斜杠 (\) 来避免终止带引号的字符串。反斜杠可使用反斜杠进行转义。定制属性名称应使用唯一的前缀以防止出现意外的名称空间重叠。请参见第 17 页中的 “软件包发布者”中有关发布者名称的论述。

可以存在同名的多个属性，这些属性将被视为无序列表。

大多数操作都有一个关键属性。**关键属性**是指将此操作与映像中的其他所有操作区分开来的属性。对于文件系统对象，关键属性是该对象的路径。

以下各节介绍了每种 IPS 操作类型以及定义这些操作的属性。操作类型在 pkg(5) 手册页中进行了详细介绍，但在此处再次介绍以供参考。每节中都包含一个示例操作，该操作与其在创建软件包期间在软件包清单中显示的一样。其他属性在发布期间可以自动添加到操作中。

文件操作

file 操作是到目前为止最常见的操作。file 操作表示普通文件。file 操作引用有效负荷，具有以下四个标准属性：

path　安装文件的文件系统路径。这是 file 操作的关键属性。path 属性的值是相对于映像根目录的路径。不要包括前导 /。

mode　文件的访问权限。mode 属性的值是采用数字形式的简单权限，而非 ACL。

owner　拥有文件的用户的名称。

group　拥有文件的组的名称。

有效负荷通常指定为位置属性：有效负荷是操作名称后面的第一个单词，它没有属性名称。在已发布的清单中，有效负荷的值是文件内容的 SHA-1 散列。如果在尚未发布的清单中存在有效负荷，则该值表示可以找到有效负荷的路径，如 pkgsend(1) 手册页中所述。如果有效负荷的值包含一个等号 (=)、双引号 (") 或空格字符，则必须使用已命名的 hash 属性而非位置属性。在同一操作中可以同时使用位置属性和 hash 属性，但散列必须完全相同。

file 操作还可以包含以下属性：

preserve　指定在升级时不应覆盖文件的内容（如果确定自文件安装或上次升级后其内容已发生了更改）。在初始安装时，如果找到现有文件，则挽救该现有文件（存储在 /var/pkg/lost+found 中）。

preserve 属性可以使用以下值之一：

renameold　使用扩展名 .old 将现有文件重命名，并将新文件放置到其所在位置中。

renamenew　现有文件保持不变，使用扩展名 .new 安装新文件。

legacy　此文件不是在初始软件包安装时安装的。在升级时，会使用扩展名 .legacy 重命名任何现有文件，并在随后将新文件放入相应位置。

true　现有文件保持不变，也不安装新文件。

overlay　指定操作是允许其他软件包在同一位置交付文件，还是用交付的文件覆盖其他文件。此功能设计用于不参与任何自组装（例如 /etc/motd）且可安全覆盖的配置文件。

如果未指定 overlay，多个软件包将无法向同一位置交付文件。

overlay 属性可以使用以下值之一：

allow　允许另一个软件包将文件交付到同一位置。除非也设置了 preserve 属性，否则此值没有效果。

`true` 该操作交付的文件将覆盖已指定了 `allow` 的任何其他操作。

基于覆盖文件的 `preserve` 属性值保留对已安装文件所做的更改。在删除时，如果仍要安装将被覆盖的操作，则将保留文件的内容，无论是否指定了 `preserve` 属性。只能一个操作覆盖另一个操作，且 `mode`、`owner` 和 `group` 属性必须匹配。

`original_name` 此属性用于处理可编辑文件在软件包之间、在位置之间或在这两者之间的移动操作。此属性的值采用的格式为源软件包的名称后跟一个冒号和文件的原始路径。所删除的任何文件将使用其软件包和路径或 `original_name` 属性的值（如果指定）进行记录。所安装的已设置 `original_name` 属性的任何可编辑文件将使用具有该名称的文件（如果它在同一打包操作中被删除）。

一旦设置此属性后，就不要更改其值，即使重复重命名软件包或文件也是如此。保持同一个值将允许从所有先前的版本进行升级。

`release-note` 该属性用于指明此文件包含发行说明文本。该属性的值为软件包 FMRI。如果 FMRI 指定的软件包名称存在于原始映像中，指定的版本比原始映像中的软件包版本更高，则此文件将成为发行说明的一部分。特殊 FMRI `feature/pkg/self` 是指包含软件包。如果 `feature/pkg/self` 的版本为 0，则此文件仅在初次安装时是发行说明的一部分。

`revert-tag` 此属性用于标记应恢复为一个组的可编辑文件。可以指定多个 `revert-tag` 值。在指定了任何这些标记的情况下调用 `pkg revert` 命令时，文件将恢复为其清单定义的状态。有关 `revert` 子命令的信息，请参见 `pkg(1)` 手册页。

特定类型的文件还可能具有其他属性。对于 ELF 文件，可识别下列属性：

`elfarch` ELF 文件的体系结构。此值是 `uname -p` 在构建文件的体系结构上的输出。

`elfbits` 此值为 32 或 64。

`elfhash` 此值是文件中在装入二进制文件时映射到内存中的 ELF 部分的散列值。在确定两个二进制文件的可执行行为是否将不同时，仅需要考虑这些部分。

下面是 `file` 操作的示例：

```
file path=usr/bin/pkg owner=root group=bin mode=0755
```

目录操作

`dir` 操作类似于 `file` 操作，也表示文件系统对象，不同之处在于它表示目录而非普通文件。`dir` 操作具有与 `file` 操作相同的四个标准属性（`path`、`owner`、`group` 和 `mode`），其中 `path` 是关键属性。

在 IPS 中对目录进行引用计数。当显式或隐式引用某目录的最新软件包不再引用该目录时，将删除该目录。如果该目录包含未打包的文件系统对象，则会将这些项移动到 /var/pkg/lost+found 中。

使用以下属性可将未打包的内容移动到一个新目录中：

salvage-from　指定所挽救项的目录。具有此属性的目录在创建时可继承所挽救目录的内容（如果存在）。有关示例，请参见第 72 页中的 “删除或重命名目录时移动未打包的内容”。

在安装期间，pkg(1) 会检查系统上给定目录操作的所有实例是否具有相同的 owner、group 和 mode 属性值。如果在系统上或在其他要在同一操作中安装的软件包中发现冲突值，则不会安装 dir 操作。

下面是 dir 操作的示例：

```
dir path=usr/share/lib owner=root group=sys mode=0755
```

链接操作

link 操作表示符号链接。link 操作具有以下标准属性：

path　安装符号链接的文件系统路径。这是 link 操作的关键属性。

target　符号链接的目标。链接将解析到的文件系统对象。

link 操作还包括允许同时在系统上安装给定软件包部分的多个版本或多个实现的属性。此类链接是中介链接，允许管理员根据需要轻松切换哪些链接指向哪个版本或实现。这些中介链接在第 72 页中的 “交付应用程序的多个实现”中进行了论述。

下面是 link 操作的示例：

```
link path=usr/lib/libpython2.6.so target=libpython2.6.so.1.0
```

硬链接操作

hardlink 操作表示硬链接。它具有与 link 操作相同的属性，path 也是其关键属性。

下面是 hardlink 操作的示例：

```
hardlink path=opt/myapplication/hardlink target=foo
```

设置操作

set 操作表示软件包级别的属性或元数据，例如软件包描述。

可以识别下列属性：

name　　属性的名称。

value　　提供给属性的值。

set 操作可以提供软件包设计者选择的任何元数据。以下属性名称对包管理系统具有特定意义：

pkg.fmri	包含方软件包的名称和版本。
info.classification	pkg(5) 客户机可以使用一个或多个标记对软件包进行分类。该值应包含一个方案（例如 org.opensolaris.category.2008 或 org.acm.class.1998）和实际分类（例如 Applications/Games），以冒号 (:) 分隔。packagemanager(1) GUI 使用的方案。附录 A，对软件包进行分类中提供了一组 info.classification 值。
pkg.summary	描述的简短概要。此值显示在 pkg list -s 输出的每行末尾，还显示在 pkg info 输出的某一行中。此值的长度不应超过 60 个字符。此值应对软件包进行描述，但不得重复软件包的名称或版本。
pkg.description	软件包的内容和功能的详细描述，长度通常约为一个段落。此值应描述为何用户可能需要安装此软件包。
pkg.obsolete	如果为 true，则将软件包标记为过时。过时的软件包除了 set 操作外不能具有任何其他操作，且不得标记为已重命名。第 71 页中的 “使软件包过时”中介绍了软件包过时。
pkg.renamed	如果为 true，则软件包已被重命名。软件包还必须包括一个或多个 depend 操作，且这些操作指向此软件包已重命名到的软件包版本。软件包不能同时标记为已重命名和过时，但在其他情况下可以具有任意多个 set 操作。第 69 页中的 “重命名、合并和拆分软件包”中介绍了软件包重命名过程。
pkg.human-version	IPS 使用的版本方案很严格，不允许 pkg.fmri 版本字段中出现字母或单词。对于给定的软件包，如果存在人们可读的常用版本，则可在此处设置该版本。此值将通过 IPS 工具进行显示。此值不用作版本比较的基准，并且不能用来替换 pkg.fmri 版本。

附录 B，如何使用 IPS 打包 Oracle Solaris OS 中介绍了一些其他信息性属性以及由 Oracle Solaris 使用的一些属性。

下面是 set 操作的示例：

```
set name=pkg.summary value="Image Packaging System"
```

驱动程序操作

driver 操作表示设备驱动程序。driver 操作不引用有效负荷。驱动程序文件自身必须作为 file 操作进行安装。可以识别下列属性。有关这些属性值的更多信息，请参见 add_drv(1M)。

name	驱动程序的名称。这通常是（但并不总是）二进制驱动程序文件的文件名。此属性是驱动程序操作的关键属性。
alias	驱动程序的别名。给定的驱动程序可以具有多个 alias 属性。无需任何特殊的引号规则。
class	一个驱动程序类。给定的驱动程序可以具有多个 class 属性。
perms	驱动程序的设备节点的文件系统权限。
clone_perms	此驱动程序的克隆驱动程序的次要节点的文件系统权限。
policy	设备的其他安全策略。给定的驱动程序可以具有多个 policy 属性，但次要设备规范不可以存在于多个属性中。
privs	驱动程序使用的特权。给定的驱动程序可以具有多个 privs 属性。
devlink	/etc/devlink.tab 中的一个项。该值定义了进入文件的确切行，带有由 \t 表示的制表符。有关更多信息，请参见devlinks(1M) 手册页。给定的驱动程序可以具有多个 devlink 属性。

下面是 driver 操作的示例：

```
driver name=vgatext \
    alias=pciclass,000100 \
    alias=pciclass,030000 \
    alias=pciclass,030001 \
    alias=pnpPNP,900 variant.arch=i386 variant.opensolaris.zone=global
```

依赖操作

depend 操作表示软件包间的相关项。一个软件包可以依赖于另一个软件包，因为第一个软件包需要第二个软件包中的功能才能运行自身包含的功能或者甚至进行安装。第 4 章，指定软件包相关项中介绍了相关项。

可以识别下列属性：

fmri	表示相关项的目标的 FMRI。此属性是 depend 操作的关键属性。FMRI 值不得包括发布者。将假定软件包名称是完整的（即有根的），即使此名称没有以正斜杠 (/) 开头。require-any 类型的相关项可具有多个 fmri 属性。fmri 值中的版本是可选项，虽然对于某些类型的相关项来说，不带版本的 FMRI 没有任何意义。 FMRI 值不能使用星号 (*)，也不能将 latest 标记用于版本。

`type` 相关项的类型。

- `require` 目标软件包是必需的，而且其版本必须为 `fmri` 属性中所指定的版本或更高版本。如果未指定版本，则任何版本都满足相关项。如果不能满足其任一 `require` 相关项，则无法安装软件包。
- `optional` 相关项目标（如果存在）必须处于指定的版本级别或更高级别。
- `exclude` 如果相关项目标处于指定的版本级别或更高级别，则无法安装包含软件包。如果未指定版本，则目标软件包无法与指定相关项的软件包同时安装。
- `incorporate` 该相关项是可选项，但是目标软件包的版本会受到约束。有关约束和冻结的说明，请参见第 4 章，指定软件包相关项。
- `require-any` 多个 `fmri` 属性指定的多个目标软件包中的任一个都可满足相关项（遵循与 `require` 相关项类型相同的规则）。
- `conditional` 仅当系统上存在 `predicate` 属性定义的软件包时，才需要相关项目标。
- `origin` 在安装此软件包之前，相关项目标（如果存在）在要修改的映像上必须具有指定值或更大值。如果 `root-image` 属性的值为 `true`，则目标必须存在于根目录为 `/` 的映像上，才能安装此软件包。
- `group` 除非软件包出现在映像避免列表上，否则该相关项目标是必需的。请注意，过时软件包会无提示地满足 `group` 相关项。有关映像避免列表的信息，请参见 `pkg(1)` 手册页中的 `avoid` 子命令。
- `parent` 如果映像不是子映像（例如区域），则会忽略相关项。如果映像是子映像，则相关项目标必须存在于父映像中。与 `parent` 相关项匹配的版本与用于 `incorporate` 相关项的版本相同。

`predicate` 表示条件性相关项的谓词的 FMRI。

`root-image` 仅对 `origin` 相关项有影响，如上所述。

下面是 `depend` 操作的示例：

```
depend fmri=crypto/ca-certificates type=require
```

许可证操作

license 操作表示许可证或其他与软件包内容相关联的信息文件。软件包可以通过 license 操作将许可证、免责声明或其他指南交付到软件包安装程序。

license 操作的有效负荷将交付到与软件包相关的映像元数据目录中，且应仅包含用户可读的文本数据。license 操作有效负荷不应包含 HTML 或任何其他形式的标记。通过各个属性，license 操作可以向客户机指示必须显示或接受相关的有效负荷。显示或接受的方法由客户机决定。

可以识别下列属性：

license 为许可证提供有意义的描述，以帮助用户在无需阅读许可证文本本身的情况下确定内容。此属性是 license 操作的关键属性。

其中一些示例值包括：

- ABC Co. Copyright Notice
- ABC Co. Custom License
- Common Development and Distribution License 1.0 (CDDL)
- GNU General Public License 2.0 (GPL)
- GNU General Public License 2.0 (GPL) Only
- MIT License
- Mozilla Public License 1.1 (MPL)
- Simplified BSD License

建议尽可能在描述中包括许可证的版本，如上所示。license 值在软件包内必须唯一。

must-accept 如果为 true，则用户必须先接受此许可证，才能安装或更新相关软件包。省略此属性等效于 false。接受的方法（例如，交互式或基于配置）由客户端决定。

must-display 如果为 true，则在执行打包操作期间客户机必须显示 license 操作的有效负荷。省略此属性等效于 false。此属性不应用于版权声明，仅用于实际许可证或执行操作期间必须显示的其他材料。显示的方法由客户端决定。

下面是 license 操作的示例：

```
license license="Apache v2.0"
```

传统操作

legacy 操作表示由传统 SVR4 包管理系统使用的软件包数据。与 legacy 操作相关联的属性将添加到传统 SVR4 包管理系统的数据库中，以便查询这些数据库的工具可以像实际安装了传统软件包一样工作。特别是，指定 legacy 操作应该会导致由 pkg 属性指定的软件包满足 SVR4 相关项。

可以识别下列属性。有关关联的参数的描述，请参见 pkginfo(4) 手册页。

category	CATEGORY 参数的值。缺省值为 system。
desc	DESC 参数的值。
hotline	HOTLINE 参数的值。
name	NAME 参数的值。缺省值为 none provided。
pkg	要安装的软件包的缩写。缺省值为软件包的 FMRI 中的名称。此属性是 legacy 操作的关键属性。
vendor	VENDOR 参数的值。
version	VERSION 参数的值。缺省值为软件包的 FMRI 中的版本。

下面是 legacy 操作的示例：

```
legacy pkg=SUNWcsu arch=i386 category=system \
    desc="core software for a specific instruction-set architecture" \
    hotline="Please contact your local service provider" \
    name="Core Solaris, (Usr)" vendor="Oracle Corporation" \
    version=11.11,REV=2009.11.11 variant.arch=i386
```

签名操作

在 IPS 中，签名操作用作软件包签名支持功能的一部分。第 9 章，对 IPS 软件包进行签名中详细介绍了签名操作。

用户操作

user 操作按照 /etc/passwd、/etc/shadow、/etc/group 和 /etc/ftpd/ftpusers 文件中指定的方式定义 UNIX 用户。来自 user 操作的信息将添加到相应文件中。

可以识别下列属性：

username	用户的唯一名称。
password	用户的加密口令。缺省值为 *LK*。
uid	用户的唯一数字 ID。缺省值为 100 之下的第一个自由值。
group	用户的主组名称。在 /etc/group 中必须能够找到此名称。
gcos-field	用户的真实姓名，如 /etc/passwd 中的 GECOS 字段所定义。缺省值为 username 属性的值。
home-dir	用户的起始目录。缺省值为 /。
login-shell	用户的缺省 shell。缺省值为空。
group-list	用户所属的辅助组。请参见 group(4) 手册页。

ftpuser	可设置为 true 或 false。缺省值 true 指示允许用户通过 FTP 登录。请参见 ftpusers(4) 手册页。
lastchg	1970 年 1 月 1 日至上次修改口令的日期之间的天数。缺省值为空。
min	所需的相邻两次更改口令之间的最小天数。必须将此字段设置为 0 或更大值才能启用口令有效期。缺省值为空。
max	口令的最大有效天数。缺省值为空。请参见 shadow(4) 手册页。
warn	用户在**口令**到期之前多少天收到警告。
inactive	允许该用户不活动的天数。按每台计算机对此进行计数。可从计算机的 lastlog 文件获取有关上次登录的信息。
expire	表示为自 UNIX 纪元（1970 年 1 月 1 日）后的天数的绝对日期。达到此数字时，将无法再进行登录。例如，到期值为 13514 指定登录将在 2007 年 1 月 1 日失效。
flag	设置为空。

下面是 user 操作的示例：

```
user gcos-field="pkg(5) server UID" group=pkg5srv uid=97 username=pkg5srv
```

组操作

group 操作按照 group(4) 文件中指定的方式定义 UNIX 组。不支持组口令。使用 group 操作定义的组最初不具有用户列表。可以使用 user 操作添加用户。

可以识别下列属性：

groupname	组名的值。
gid	组的唯一数字 ID。缺省值为小于 100 的第一个自由组。

下面是 group 操作的示例：

```
group groupname=pkg5srv gid=97
```

软件包系统信息库

软件包系统信息库包含来自一个或多个发布者的软件包。可以将系统信息库配置为通过各种不同的方式进行访问：HTTP、HTTPS、文件（在本地存储设备上或通过 NFS 或 SMB）以及作为自包含软件包归档文件（通常扩展名为 .p5p）。

使用软件包归档文件可以方便地分发 IPS 软件包，第 40 页中的 “作为软件包归档文件发布”中将进一步讨论软件包归档文件。

通过 HTTP 或 HTTPS 访问的系统信息库具有一个与之关联的服务器进程 `pkg.depotd`。有关更多信息，请参见`pkg.depotd(1M)` 手册页。有关示例，请参见《复制和创建 Oracle Solaris 11.1 软件包系统信息库》中的“使用 HTTP 接口检索软件包”。

对于文件系统信息库，系统信息库软件将作为访问方客户机的一部分运行。系统信息库使用 `pkgrepo` 和 `pkgrecv` 命令创建，如《复制和创建 Oracle Solaris 11.1 软件包系统信息库》中所示。

◆ ◆ ◆ 第 2 章

2 使用 IPS 打包软件

通过本章，您可以开始构建自己的软件包，包括：

- 设计、创建和发布新软件包
- 将 SVR4 软件包转换为 IPS 软件包

设计软件包

本节中所述的针对良好软件包开发的许多条件需要您进行权衡。通常很难同时满足所有要求。以下条件是按重要程度介绍的。但是，该序列可灵活变化，具体取决于您的情况。尽管每一个条件都很重要，但是您需要决定是否优化这些要求以便生成一组好的软件包。

选择软件包名称。
Oracle Solaris 使用用于 IPS 软件包的分层命名策略。请尽可能地将软件包名称设计为适合同一个方案。尽量保持软件包名称的最后部分是唯一的，以便用户可以将在命令中指定短的软件包名称，如 `pkg install`。

优化客户机/服务器配置。
布局软件包时，请考虑各种软件使用模式（客户机和服务器）。良好的软件包设计应区分受影响的文件以优化每种配置类型的安装。例如，对于网络协议实现，软件包用户应具备在不必安装服务器的情况下安装客户机的能力。如果客户机和服务器共享一些实现组件，请创建一个包含共享内容的基本软件包。

按功能边界打包。
软件包应该是自包含的，并且可通过一组功能清楚地识别。例如，一个包含 ZFS 的软件包应该包含所有 ZFS 实用程序，并仅限于 ZFS 二进制文件。

软件包应该从用户的角度组织成功能单元。

按许可证或特许权边界打包。
将因合同协议导致需要特许权使用费以及包含不同的软件许可证条款的代码放入专用软件包或软件包组。除非必要，不要将代码分散到更多软件包中。

避免或管理软件包之间的重叠。
重叠的软件包无法同时安装。重叠的软件包的一个示例是那些将不同的内容交付到同一文件系统位置的软件包。由于该错误只有在用户尝试安装软件包时才可能出现，所以重叠的软件包会导致很糟糕的用户体验。`pkglint(1)` 工具有助于在软件包设计期间检测此错误。

如果软件包内容必须不同，请声明 `exclude` 相关项，以便 IPS 不允许同时安装这些软件包。

正确设置软件包大小。
软件包代表一个单独的软件单元，可以安装也可以不安装。（请参见第 58 页中的“可选软件组件”中有关侧面的说明，以了解软件包如何能够提供可选软件组件。）总是同时安装的软件包应该组合在一起。由于 IPS 只下载更新中的已更改文件，所以如果更改有限，即使大型软件包也能快速更新。

创建并发布软件包

由于大量的自动化，使用 IPS 对软件包打包通常很简单。自动化可以避免重复执行单调的操作，这种重复似乎是大部分打包错误产生的主要原因。

在 IPS 中发布包含以下步骤：

1. 生成软件包清单。
2. 将必要的元数据添加到生成的清单中。
3. 评估相关项。
4. 添加需要的任何侧面或执行器。
5. 验证软件包。
6. 发布软件包。
7. 测试软件包。

生成软件包清单

最简单的入门方法是将组件文件组织成安装系统上所需的相同目录结构。

有两种方法可用于实现此目的：

- 如果您要打包的软件已经存在于 tarball 中，则将该 tarball 解压缩到子目录中。对于使用 `autoconf` 实用程序的许多开源软件包，将 `DESTDIR` 环境变量设置为指向所需的原型区域可以实现此目的。`pkg:/developer/build/autoconf` 软件包中提供了 `autoconf` 实用程序。
- 在 Makefile 中使用 `install` 目标。

假定您的软件包含一个二进制文件、一个库和一个手册页，并且要在 `/opt` 下名为 `mysoftware` 的目录中安装该软件。在构建区域中创建一个目录，在此目录下，您的软件以以上布局显示。在以下示例中，此目录名为 `proto`：

```
proto/opt/mysoftware/lib/mylib.so.1
proto/opt/mysoftware/bin/mycmd
proto/opt/mysoftware/man/man1/mycmd.1
```

使用 pkgsend generate 命令为 proto 区域生成清单。通过 pkgfmt 可对输出软件包清单进行排序，以便该清单可读性更好。有关更多信息，请参见 pkgsend(1) 和 pkgfmt(1) 手册页。

在以下示例中，proto 目录位于当前工作目录中：

```
$ pkgsend generate proto | pkgfmt > mypkg.p5m.1
```

输出的 mypkg.p5m.1 文件包含以下行：

```
dir  path=opt owner=root group=bin mode=0755
dir  path=opt/mysoftware owner=root group=bin mode=0755
dir  path=opt/mysoftware/bin owner=root group=bin mode=0755
file opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd owner=root \
    group=bin mode=0644
dir  path=opt/mysoftware/lib owner=root group=bin mode=0755
file opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    owner=root group=bin mode=0644
dir  path=opt/mysoftware/man owner=root group=bin mode=0755
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644
```

要打包的文件路径在 file 操作中出现了两次：

- 单词 file 之后的第一个单词说明了 proto 区域中文件的位置。
- path= 属性中的路径指定了要安装文件的位置。

使用该双项，您可以在不修改 proto 区域的情况下修改安装位置。假如您要对设计用于安装在其他操作系统上的软件重新打包，此功能可以节省大量时间。

请注意，pkgsend generate 对目录所有者和组都应用了缺省值。对于 /opt，缺省值是不正确的。需要删除该目录，因为它是由已经存在于系统上的其他软件包提供的，并且如果 /opt 的属性和那些已经存在于系统上的属性冲突，则 pkg(1) 将不会安装该软件包。下面第 34 页中的 “将必要的元数据添加到生成的清单中”显示了删除不需要的目录的编程方式。

如果文件名中包含等号 (=)、双引号 (") 或空格字符，则 pkgsend 在清单中生成 hash 属性，如下例所示：

```
$ mkdir -p proto/opt
$ touch proto/opt/my\ file1
$ touch proto/opt/"my file2"
$ touch proto/opt/my=file3
$ touch proto/opt/’my"file4’
$ pkgsend generate proto
dir group=bin mode=0755 owner=root path=opt
file group=bin hash=opt/my=file3 mode=0644 owner=root path=opt/my=file3
```

```
file group=bin hash="opt/my file2" mode=0644 owner=root path="opt/my file2"
file group=bin hash='opt/my"file4' mode=0644 owner=root path='opt/my"file4'
file group=bin hash="opt/my file1" mode=0644 owner=root path="opt/my file1"
```

发布软件包（请参见第 39 页中的 “发布软件包”）时，hash 属性的值会成为文件内容的 SHA-1 散列，如第 20 页中的 “文件操作”中所述。

将必要的元数据添加到生成的清单中

软件包应该定义以下元数据。有关这些值以及如何设置这些值的更多信息，请参见第 22 页中的 “设置操作”。

pkg.fmri
: 第 16 页中的 “软件包标识符：FMRI”中所介绍的软件包的名称和版本。有关 Oracle Solaris 版本控制的介绍可以在第 95 页中的 “Oracle Solaris 软件包版本控制”中找到。

pkg.description
: 软件包内容的描述。

pkg.summary
: 单行描述概要。

variant.arch
: 该软件包适用的每种体系结构。如果整个软件包可以安装在任何体系结构中，则可以省略 variant.arch。第 5 章，允许变量中讨论了生成包含用于不同体系结构的不同组件的软件包。

info.classification
: packagemanager(1) GUI 使用的分组方案。支持的值显示在附录 A，对软件包进行分类中。本节中的示例随机指定了一个等级。

此示例还将 link 操作添加到 /usr/share/man/index.d 中，指向 mysoftware 下的 man 目录。第 37 页中的 “添加需要的任何侧面或执行器”中将进一步介绍该链接。

请使用 pkgmogrify(1) 编辑生成的清单，而不要直接修改清单。有关使用 pkgmogrify 修改软件包清单的完整说明，请参见第 6 章，以编程方式修改软件包清单。

创建以下 pkgmogrify 输入文件以指定要对清单进行的更改。将此文件命名为 mypkg.mog。在此示例中，使用宏定义体系结构，使用正则表达式匹配从清单中删除 /opt 目录。

```
set name=pkg.fmri value=mypkg@1.0,5.11-0
set name=pkg.summary value="This is an example package"
set name=pkg.description value="This is a full description of \
all the interesting attributes of this example package."
set name=variant.arch value=$(ARCH)
set name=info.classification \
    value=org.opensolaris.category.2008:Applications/Accessories
link path=usr/share/man/index.d/mysoftware target=opt/mysoftware/man
<transform dir path=opt$->drop>
```

对 mypkg.p5m.1 清单运行 pkgmogrify（mypkg.mog 已更改）：

```
$ pkgmogrify -DARCH='uname -p' mypkg.p5m.1 mypkg.mog | pkgfmt > mypkg.p5m.2
```

输出的 mypkg.p5m.2 文件包含以下内容。path=opt 的 dir 操作已删除，并且 mypkg.mog 中的元数据和链接内容已添加到原始的 mypkg.p5m.1 内容中。

```
set name=pkg.fmri value=mypkg@1.0,5.11-0
set name=pkg.summary value="This is an example package"
set name=pkg.description \
    value="This is a full description of all the interesting attributes of this example package."
set name=info.classification \
    value=org.opensolaris.category.2008:Applications/Accessories
set name=variant.arch value=i386
dir  path=opt/mysoftware owner=root group=bin mode=0755
dir  path=opt/mysoftware/bin owner=root group=bin mode=0755
file opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd owner=root \
    group=bin mode=0644
dir  path=opt/mysoftware/lib owner=root group=bin mode=0755
file opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    owner=root group=bin mode=0644
dir  path=opt/mysoftware/man owner=root group=bin mode=0755
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644
link path=usr/share/man/index.d/mysoftware target=opt/mysoftware/man
```

评估相关项

使用 pkgdepend(1) 命令可自动生成软件包的相关项。生成的 depend 操作在第 24 页中的“依赖操作”中进行了定义，并在第 4 章，指定软件包相关项中有进一步的讨论。

相关项生成由两个单独的步骤组成：

1. 相关项生成。确定软件依赖的文件。使用 pkgdepend generate 命令。
2. 相关项解析。确定包含软件所依赖的那些文件的软件包。使用 pkgdepend resolve 命令。

生成软件包相关项

在以下命令中，-m 选项导致 pkgdepend 在输出中包含整个清单。-d 选项将 proto 目录传递给命令。

```
$ pkgdepend generate -md proto mypkg.p5m.2 | pkgfmt > mypkg.p5m.3
```

输出的 mypkg.p5m.3 文件包含以下内容。pkgdepend 实用程序通过 mylib.so.1 和 mycmd 添加了有关依赖于 libc.so.1 的相关项的表示法。无提示地省略了 mycmd 和 mylib.so.1 之间的内部相关项。

```
set name=pkg.fmri value=mypkg@1.0,5.11-0
set name=pkg.summary value="This is an example package"
set name=pkg.description \
```

```
    value="This is a full description of all the interesting attributes of this example package."
set name=info.classification \
    value=org.opensolaris.category.2008:Applications/Accessories
set name=variant.arch value=i386
dir  path=opt/mysoftware owner=root group=bin mode=0755
dir  path=opt/mysoftware/bin owner=root group=bin mode=0755
file opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd owner=root \
    group=bin mode=0644
dir  path=opt/mysoftware/lib owner=root group=bin mode=0755
file opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    owner=root group=bin mode=0644
dir  path=opt/mysoftware/man owner=root group=bin mode=0755
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644
link path=usr/share/man/index.d/mysoftware target=opt/mysoftware/man
depend fmri=__TBD pkg.debug.depend.file=libc.so.1 \
    pkg.debug.depend.reason=opt/mysoftware/bin/mycmd \
    pkg.debug.depend.type=elf type=require pkg.debug.depend.path=lib \
    pkg.debug.depend.path=opt/mysoftware/lib pkg.debug.depend.path=usr/lib
depend fmri=__TBD pkg.debug.depend.file=libc.so.1 \
    pkg.debug.depend.reason=opt/mysoftware/lib/mylib.so.1 \
    pkg.debug.depend.type=elf type=require pkg.debug.depend.path=lib \
    pkg.debug.depend.path=usr/lib
```

解析软件包相关项

为解析相关项，pkgdepend 检查当前安装在用于生成软件的映像中的软件包。缺省情况下， pkgdepend 将输出置于 mypkg.p5m.3.res 中。此步骤需要运行一段时间，因为需要装入有关运行中系统的大量信息。如果希望将此时间分摊到所有的软件包上，可以使用 pkgdepend 实用程序一次性解析多个软件包。一次只对一个软件包运行 pkgdepend 从时间上来看不太高效。

```
$ pkgdepend resolve -m mypkg.p5m.3
```

完成后，输出的 mypkg.p5m.3.res 文件包含以下内容。pkgdepend 实用程序将有关依赖于 libc.so.1 的文件相关项的表示法转换为交付该文件的 pkg:/system/library 中的软件包相关项。

```
set name=pkg.fmri value=mypkg@1.0,5.11-0
set name=pkg.summary value="This is an example package"
set name=pkg.description \
    value="This is a full description of all the interesting attributes of this example package."
set name=info.classification \
    value=org.opensolaris.category.2008:Applications/Accessories
set name=variant.arch value=i386
dir  path=opt/mysoftware owner=root group=bin mode=0755
dir  path=opt/mysoftware/bin owner=root group=bin mode=0755
file opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd owner=root \
    group=bin mode=0644
dir  path=opt/mysoftware/lib owner=root group=bin mode=0755
file opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    owner=root group=bin mode=0644
dir  path=opt/mysoftware/man owner=root group=bin mode=0755
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
```

```
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644
link path=usr/share/man/index.d/mysoftware target=opt/mysoftware/man
depend fmri=pkg:/system/library@0.5.11-0.175.1.0.0.21.0 type=require
```

您应该使用 pkgdepend 生成相关项，而不是手动声明 depend 操作。由于软件包内容会随着时间而更改，手动相关项可能不正确或没有必要。例如，当应用程序所依赖的文件移动到其他软件包时，任何手动声明的依赖于先前软件包的相关项对于该相关项都将是不正确的。

如果 pkgdepend 无法完全确定相关项，手动声明某些相关项可能会是必要的。在这种情况下，应该向清单中添加解释性注释。

添加需要的任何侧面或执行器

第 5 章，允许变量和第 7 章，在软件包安装过程中自动进行系统更改中更详细地介绍了侧面和执行器。**侧面**表示一种操作，该操作不是必要的，可以选择性安装。**执行器**指定在安装、更新或删除关联操作时必须发生的系统更改。

此示例软件包在 opt/mysoftware/man/man1 中交付了一个手册页。本节说明如何添加侧面以指示手册页是可选的。用户可以选择安装软件包中除手册页以外的所有内容。（用户将侧面设置为 false 时，对于使用该侧面对 file 操作进行标记的所有软件包，不会安装任何手册页。）

要将手册页包含在索引中，安装软件包时，必须重新启动 svc:/application/man-index:default SMF 服务。本节说明如何添加 restart_fmri 执行器以执行该任务。man-index 服务在 /usr/share/man/index.d 中查找指向手册页所在目录的符号链接，将每一个链接的目标添加到其扫描的目录列表中。为了将手册页包含在索引中，此示例软件包包含从 /usr/share/man/index.d/mysoftware 到 /opt/mysoftware/man 的链接。包含该链接和执行器是第 13 页中的 “软件自组装”中所讨论的自组装的一个很好示例，用于 Oracle Solaris OS 的整个打包过程。

您可以使用的一组 pkgmogrify 转换位于 /usr/share/pkg/transforms 中。这些转换用于打包 Oracle Solaris OS，并在第 6 章，以编程方式修改软件包清单中有详细介绍。

文件 /usr/share/pkg/transforms/documentation 中包含的转换类似于此示例中设置手册页侧面和重新启动 man-index 服务所需的转换。由于此示例将手册页交付到 /opt，documentation 转换必须按如下所示进行修改。这些修改的转换包括正则表达式 opt/.+/man(/.+)?，该表达式与 opt 下包含 man 子目录的所有路径相匹配。将以下修改的转换保存到 /tmp/doc-transform：

```
<transform dir file link hardlink path=opt/.+/man(/.+)? -> \
    default facet.doc.man true>
<transform file path=opt/.+/man(/.+)? -> \
    add restart_fmri svc:/application/man-index:default>
```

使用以下命令可将这些转换应用到清单：

```
$ pkgmogrify mypkg.p5m.3.res /tmp/doc-transform | pkgfmt > mypkg.p5m.4.res
```

输入 mypkg.p5m.3.res 清单包含以下三个与手册页相关的操作：

```
dir  path=opt/mysoftware/man owner=root group=bin mode=0755
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644
```

应用转换后，输出 mypkg.p5m.4.res 清单包含以下修改的操作：

```
dir  path=opt/mysoftware/man owner=root group=bin mode=0755 facet.doc.man=true
dir  path=opt/mysoftware/man/man1 owner=root group=bin mode=0755 \
    facet.doc.man=true
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    owner=root group=bin mode=0644 \
    restart_fmri=svc:/application/man-index:default facet.doc.man=true
```

出于效率考虑，在评估相关项前，初次添加元数据时可能已经添加了这些转换。

验证软件包

发布前的最后一步是对清单运行 pkglint(1)，以便发现在发布和测试前可以识别的错误。pkglint 可以发现的某些错误在发布时或用户尝试安装软件包时也会发现，但是您当然希望在软件包设计期间尽可能早地识别错误。

pkglint 报告的错误示例包括：

- 交付的文件已经属于其他软件包。
- 有关共享、引用计数操作的元数据（如目录）存在不一致情况。此错误的示例在第 32 页中的 “生成软件包清单”结尾有介绍。

可以使用以下模式之一运行 pkglint：

- 直接针对软件包清单运行。此模式通常足以快速地检查清单的有效性。
- 针对软件包清单运行，同时引用软件包系统信息库。发布到系统信息库前，至少需要使用该模式一次。

  通过引用系统信息库，pkglint 可以执行额外检查以确保软件包和系统信息库中的其他软件包之间可以很好地交互。

使用 pkglint -L 命令可显示 pkglint 所执行检查的完整列表。pkglint(1) 手册页中提供了有关如何启用、禁用和绕过特定检查的详细信息。该手册页还详细介绍了如何扩展 pkglint 以运行额外检查。

以下输出显示了示例清单的问题：

```
$ pkglint mypkg.p5m.4.res
Lint engine setup...
Starting lint run...
```

```
WARNING pkglint.action005.1       obsolete dependency check skipped: unable
to find dependency pkg:/system/library@0.5.11-0.175.1.0.0.21.0 for
pkg:/mypkg@1.0,5.11-0
```

此警告对于此示例可接受。pkglint.action005.1 警告显示 pkglint 无法找到此示例软件包依赖的名为 pkg:/system/library@0.5.11-0.175.1.0.0.21.0 的软件包。该相关项软件包在软件包系统信息库中，但无法找到，因为调用 pkglint 时仅使用清单文件作为参数。

在以下命令中，-r 选项引用了一个包含相关项软件包的系统信息库。-c 选项指定了一个本地目录，用于缓存来自 lint 和引用系统信息库的软件包元数据：

```
$ pkglint -c ./solaris-reference -r http://pkg.oracle.com/solaris/release mypkg.p5m.4.res
```

发布软件包

IPS 提供了三种不同的软件包交付方式：

- 发布到本地基于文件的系统信息库中。
- 发布到远程基于 HTTP 的系统信息库中。
- 转换为 .p5p 软件包归档文件。

一般情况下，对于测试软件包来说，发布到基于文件的系统信息库就足够了。

如果必须将软件包传输到无法访问软件包系统信息库的其他计算机上，则将一个或多个软件包转换为软件包归档文件会很方便。

也可以直接将软件包发布到 HTTP 系统信息库，该系统信息库托管在包含 svc:/application/pkg/server 服务的读/写实例的计算机上，由该计算机运行 pkg.depotd(1M)。

通常不建议发布到 HTTP 系统信息库，因为通过 HTTP 发布时，不对传入软件包进行授权或验证检查。如果无法对文件系统信息库进行 NFS 或 SMB 访问，那么在安全的网络中或者当测试多个计算机上的同一个软件包时，发布到 HTTP 系统信息库会很方便。

通过 HTTP 或 HTTPS 安装软件包是可行的。

发布到本地文件系统信息库

使用 pkgrepo(1) 命令可创建并管理系统信息库。在系统上选择一个位置，创建一个系统信息库，然后为该系统信息库设置缺省发布者：

```
$ pkgrepo create my-repository
$ pkgrepo -s my-repository set publisher/prefix=mypublisher
$ ls my-repository
pkg5.repository
```

使用 pkgsend(1) 命令可发布示例软件包，然后使用 pkgrepo 检查该系统信息库：

```
$ pkgsend -s my-repository publish -d proto mypkg.p5m.4.res
pkg://mypublisher/mypkg@1.0,5.11-0:20120331T034425Z
PUBLISHED
$ pkgrepo -s my-repository info
PUBLISHER   PACKAGES STATUS           UPDATED
mypublisher 1        online           2012-03-31T03:44:25.235964Z
```

如果需要，可以使用 pkg.depotd 通过 HTTP 或 HTTPS 提供文件系统信息库访问。

作为软件包归档文件发布

使用软件包归档文件，您可以在一个文件中分发多组软件包。使用 pkgrecv(1) 命令可根据软件包系统信息库创建软件包归档文件，或根据软件包归档文件创建软件包系统信息库。

软件包系统信息库不可用时，可以轻松地从现有的 Web 站点下载软件包归档文件，复制到 USB key，或刻录到 DVD 中以进行安装。

以下命令根据上节中创建的简单系统信息库创建软件包归档文件：

```
$ pkgrecv -s my-repository -a -d myarchive.p5p mypkg
Retrieving packages for publisher mypublisher ...
Retrieving and evaluating 1 package(s)...
DOWNLOAD                                  PKGS       FILES   XFER (MB)   SPEED
Completed                                  1/1         3/3     0.7/0.7 17.9k/s

ARCHIVE                                               FILES   STORE (MB)
myarchive.p5p                                         14/14      0.7/0.7
```

使用软件包系统信息库和归档文件

使用 pkgrepo 命令可列出系统信息库或归档文件中的最新可用软件包：

```
$ pkgrepo -s my-repository list '*@latest'
PUBLISHER   NAME                                                  O VERSION
mypublisher mypkg                                                   1.0,5.11-0:20120331T034425Z
$ pkgrepo -s myarchive.p5p list '*@latest'
PUBLISHER   NAME                                                  O VERSION
mypublisher mypkg                                                   1.0,5.11-0:20120331T034425Z
```

此输出可用于构建脚本，以便使用给定系统信息库中所有软件包的最新版本创建归档文件。

可以使用 -g 选项向 pkg install 和其他 pkg 操作提供临时系统信息库或软件包归档文件。此类临时系统信息库和归档文件无法用于包含子映像或父映像的系统（例如，包含非全局区域的系统），因为不会使用该发布者信息临时配置系统信息库。非全局区域与全局区域之间为子/父关系。但是，软件包归档文件可以在非全局区域中设置为本地发布者的源。

测试软件包

软件包开发的最后一步是测试已发布的软件包是否已正确打包。

要在不需要 root 特权的情况下测试安装，需要为测试用户指定 Software Installation（软件安装）配置文件。使用 `usermod` 命令的 `-P` 选项可为测试用户指定 Software Installation（软件安装）配置文件。

注 – 如果该映像已安装了子映像（非全局区域），则无法使用带 `-g` 选项的 `pkg install` 命令测试该软件包的安装。您必须在该映像中配置 `my-repository` 系统信息库。

将 `my-repository` 系统信息库中的发布者添加到该映像中的已配置发布者：

```
$ pfexec pkg set-publisher -p my-repository
pkg set-publisher:
  Added publisher(s): mypublisher
```

可以使用 `pkg install -nv` 命令查看安装命令将执行的操作，而不进行任何更改。以下命令实际上安装软件包：

```
$ pfexec pkg install mypkg
           Packages to install:  1
       Create boot environment: No
Create backup boot environment: No
            Services to change:  1

DOWNLOAD                                  PKGS       FILES   XFER (MB)   SPEED
Completed                                  1/1         3/3     0.0/0.0    0B/s

PHASE                                        ITEMS
Installing new actions                       15/15
Updating package state database               Done
Updating image state                          Done
Creating fast lookup database                 Done
Reading search index                          Done
Updating search index                          1/1
```

检查交付到系统上的软件：

```
$ find /opt/mysoftware/
/opt/mysoftware/
/opt/mysoftware/bin
/opt/mysoftware/bin/mycmd
/opt/mysoftware/lib
/opt/mysoftware/lib/mylib.so.1
/opt/mysoftware/man
/opt/mysoftware/man/man-index
/opt/mysoftware/man/man-index/term.doc
/opt/mysoftware/man/man-index/.index-cache
/opt/mysoftware/man/man-index/term.dic
/opt/mysoftware/man/man-index/term.req
/opt/mysoftware/man/man-index/term.pos
/opt/mysoftware/man/man1
/opt/mysoftware/man/man1/mycmd.1
```

执行器重新启动 `man-index` 服务后，除了二进制文件和手册页，系统同时生成了手册页索引。

pkg info 命令显示添加到软件包的元数据：

```
$ pkg info mypkg
          Name: mypkg
       Summary: This is an example package
   Description: This is a full description of all the interesting attributes of
                this example package.
      Category: Applications/Accessories
         State: Installed
     Publisher: mypublisher
       Version: 1.0
 Build Release: 5.11
        Branch: 0
Packaging Date: March 31, 2012 03:44:25 AM
          Size: 0.00 B
          FMRI: pkg://mypublisher/mypkg@1.0,5.11-0:20120331T034425Z
```

查询通过 mypkg 交付的文件时，pkg search 命令将返回命中：

```
$ pkg search -l mycmd.1
INDEX      ACTION VALUE                           PACKAGE
basename   file   opt/mysoftware/man/man1/mycmd.1 pkg:/mypkg@1.0-0
```

将 SVR4 软件包转换为 IPS 软件包

本节显示了将 SVR4 软件包转换为 IPS 软件包的示例，并强调了可能需要特别注意的区域。

要将 SVR4 软件包转换为 IPS 软件包，请遵循本章中上面所介绍的有关在 IPS 中打包任何软件的相同步骤。其中大部分步骤对于 SVR4 到 IPS 软件包的转换都是相同的，因此在本节中不再解释。本节介绍了转换软件包（而非创建新软件包）时的不同步骤。

从 SVR4 软件包生成 IPS 软件包清单

pkgsend generate 命令的 *source* 参数可以是 SVR4 软件包。有关支持的源的完整列表，请参见 pkgsend(1) 手册页。当 *source* 是 SVR4 软件包时，pkgsend generate 将使用该 SVR4 软件包中的 pkgmap(4) 文件，而不是软件包中所交付文件所在的目录。

扫描 prototype 文件时，pkgsend 实用程序同时查找将软件包转换为 IPS 时可能导致问题的项。pkgsend 实用程序将报告那些问题并打印生成的清单。

本节中使用的示例 SVR4 软件包包含以下 pkginfo(4) 文件：

```
VENDOR=My Software Inc.
HOTLINE=Please contact your local service provider
PKG=MSFTmypkg
ARCH=i386
DESC=A sample SVR4 package of My Sample Package
```

```
CATEGORY=system
NAME=My Sample Package
BASEDIR=/
VERSION=11.11,REV=2011.10.17.14.08
CLASSES=none manpage
PSTAMP=linn20111017132525
MSFT_DATA=Some extra package metadata
```

本节中使用的示例 SVR4 软件包包含以下相应的 prototype(4) 文件：

```
i pkginfo
i copyright
i postinstall
d none opt 0755 root bin
d none opt/mysoftware 0755 root bin
d none opt/mysoftware/lib 0755 root bin
f none opt/mysoftware/lib/mylib.so.1 0644 root bin
d none opt/mysoftware/bin 0755 root bin
f none opt/mysoftware/bin/mycmd 0755 root bin
d none opt/mysoftware/man 0755 root bin
d none opt/mysoftware/man/man1 0755 root bin
f none opt/mysoftware/man/man1/mycmd.1 0644 root bin
```

对使用这些文件创建的 SVR4 软件包运行 pkgsend generate 命令可以生成以下 IPS 清单：

```
$ pkgsend generate ./MSFTmypkg | pkgfmt
pkgsend generate: ERROR: script present in MSFTmypkg: postinstall

set name=pkg.summary value="My Sample Package"
set name=pkg.description value="A sample SVR4 package of My Sample Package"
set name=pkg.send.convert.msft-data value="Some extra package metadata"
dir path=opt owner=root group=bin mode=0755
dir path=opt/mysoftware owner=root group=bin mode=0755
dir path=opt/mysoftware/bin owner=root group=bin mode=0755
file reloc/opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd owner=root \
    group=bin mode=0755
dir path=opt/mysoftware/lib owner=root group=bin mode=0755
file reloc/opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    owner=root group=bin mode=0644
dir path=opt/mysoftware/man owner=root group=bin mode=0755
dir path=opt/mysoftware/man/man1 owner=root group=bin mode=0755
file reloc/opt/mysoftware/man/man1/mycmd.1 \
    path=opt/mysoftware/man/man1/mycmd.1 owner=root group=bin mode=0644
legacy pkg=MSFTmypkg arch=i386 category=system \
    desc="A sample SVR4 package of My Sample Package" \
    hotline="Please contact your local service provider" \
    name="My Sample Package" vendor="My Software Inc." \
    version=11.11,REV=2011.10.17.14.08
license install/copyright license=MSFTmypkg.copyright
```

请注意有关 pkgsend generate 输出的以下几点：

- pkg.summary 和 pkg.description 属性是根据 pkginfo 文件中的数据自动创建的。
- set 操作是根据 pkginfo 文件中的额外参数生成的。set 操作在 pkg.send.convert.* 名称空间下设置。使用 pkgmogrify(1) 转换可将此类属性转换为更合适的属性名称。

- legacy 操作是根据 pkginfo 文件中的数据生成的。
- 生成了 license 操作，指向 SVR4 软件包中所使用的版权文件。
- 发出了有关无法转换的脚本操作的错误消息。

以下检查显示了 pkgsend generate 的错误消息和非零返回代码：

```
$ pkgsend generate MSFTmypkg > /dev/null
pkgsend generate: ERROR: script present in MSFTmypkg: postinstall
$ echo $?
1
```

SVR4 软件包使用的 postinstall 脚本无法直接转换为 IPS 等效项。必须手动检查该脚本。

软件包中的 postinstall 脚本包含以下内容：

```
#!/usr/bin/sh
catman -M /opt/mysoftware/man
```

可以使用指向现有 SMF 服务 svc:/application/man-index:default 的 restart_fmri 执行器归档与该脚本相同的结果，如第 37 页中的 “添加需要的任何侧面或执行器”中所述。有关执行器的详尽说明，请参见第 7 章，在软件包安装过程中自动进行系统更改。

pkgsend generate 命令还检查是否存在类操作脚本，并生成错误消息来指示应该检查哪些脚本。

在由 SVR4 软件包到 IPS 软件包的所有转换中，需要的功能可以通过使用现有操作类型或 SMF 服务来实现。有关可用操作类型的详细信息，请参见第 18 页中的 “软件包内容：操作”。有关 SMF 和软件包操作的信息，请参见第 7 章，在软件包安装过程中自动进行系统更改。

添加软件包元数据和解析相关项的方式与第 32 页中的 “创建并发布软件包”中所述相同，因此在本节中不再讨论。对于转换的软件包，软件包创建过程中下一个可能显示其特有问题的步骤是验证步骤。

验证已转换的软件包

转换 SVR4 软件包时的一个常见错误源自通过 SVR4 软件包交付的目录与通过 IPS 软件包交付的相同目录中存在不匹配的属性。

在此示例中的 SVR4 软件包中，样例清单中 /opt 的目录操作的属性与系统软件包为该目录定义的属性不同。

第 21 页中的 “目录操作”一节说明所有的引用计数操作必须使用相同的属性。当尝试安装目前生成的 mypkg 版本时，出现以下错误：

```
# pkg install mypkg
Creating Plan /
pkg install: The requested change to the system attempts to install multiple actions
for dir 'opt' with conflicting attributes:

    1 package delivers 'dir group=bin mode=0755 owner=root path=opt':
        pkg://mypublisher/mypkg@1.0,5.11-0:20111017T020042Z
    3 packages deliver 'dir group=sys mode=0755 owner=root path=opt':
        pkg://solaris/developer/build/onbld@0.5.11,5.11-0.175.0.0.0.1.0:20111012T010101Z
        pkg://solaris/system/core-os@0.5.11,5.11-0.175.0.0.0.1.0:20111012T023456Z

These packages may not be installed together. Any non-conflicting set may
be, or the packages must be corrected before they can be installed.
```

要在发布软件包之前（而不是安装时）捕捉到错误，可以对引用系统信息库使用 pkglint(1) 命令，如以下示例所示：

```
$ pkglint -c ./cache -r file:///scratch/solaris-repo ./mypkg.mf.res
Lint engine setup...

PHASE                                          ITEMS
4                                          4292/4292
Starting lint run...

ERROR pkglint.dupaction007        path opt is reference-counted but has different attributes across 5
duplicates: group: bin -> mypkg group: sys -> developer/build/onbld system/core-os system/ldoms/ldomsmanager
```

请注意有关 `path opt` 在不同软件包中包含不同属性的错误消息。

`pkglint` 报告的额外 `ldomsmanager` 软件包位于引用软件包系统信息库中，但没有安装在测试系统中。先前由 `pkg install` 报告的错误中没有列出 `ldomsmanager` 软件包，因为该软件包没有安装。

其他软件包转换注意事项

虽然可以在 Oracle Solaris 11 系统上直接安装 SVR4 软件包，但是您应该创建相应的 IPS 软件包。安装 SVR4 软件包是临时的解决方案。

除第 26 页中的 “传统操作”中所介绍的 `legacy` 操作以外，这两个包管理系统间不存在任何链接，并且 SVR4 和 IPS 软件包不会相互引用软件包元数据。

IPS 具有可以确定打包内容是否已正确安装的命令，如 `pkg verify`。但是，如果其他包管理系统合法安装了软件包或运行了可修改 IPS 软件包所安装的目录或文件的安装脚本，则会导致错误。

IPS 的 `pkg fix` 和 `pkg revert` 命令可以覆盖通过 SVR4 软件包和 IPS 软件包交付的文件，可能会导致已打包的应用程序出现故障。

`pkg install` 等命令通常用于检查重复操作和引用计数操作的常见属性，当来自其他包管理系统的文件发生冲突时，该命令可能无法检测到潜在的错误。

考虑到这些潜在错误以及 IPS 中的完整软件包开发工具链，建议为 Oracle Solaris 11 开发 IPS 软件包而不是 SVR4 软件包。

◆ ◆ ◆ 第 3 章

3 安装、删除和更新软件包

本章介绍了在安装、更新和删除已安装在映像中的软件时 IPS 客户机的内部工作原理。

了解 pkg(1) 如何执行这些操作对于了解各种可能会出现的错误以及更快地解决软件包相关项问题很重要。

软件包更改是如何执行的

在调用 pkg 来修改计算机上安装的软件时，会执行以下步骤：

- 检查输入中的错误
- 确定系统结束状态
- 运行基本检查
- 运行解析器
- 优化解析器结果
- 评估操作
- 下载内容
- 执行操作
- 处理执行器

在全局区域中执行这些步骤时，pkg 还可以对系统上的任何非全局区域进行操作。例如，pkg 可以确保全局区域和非全局区域间的相关项是正确的，并且可以根据需要为非全局区域下载内容和执行操作。第 10 章，处理非全局区域详细讨论了区域。

检查输入中的错误

将根据在命令行上提供的选项执行基本错误检查。

确定系统结束状态

构造了对期望的系统结束状态的描述。对于更新映像中的所有软件包的情况，期望的结束状态可能为“当前已安装了所有软件包或其更新版本”之类的内容。对于删除软件包的情况，期望的结束状态可能为“除此软件包外，当前已安装了所有软件包”之类的内容。

IPS 尝试确定用户希望该结束状态是什么样的。在某些情况下，IPS 可能会确定一个不合用户期望的结束状态，尽管该结束状态确实与用户的要求相匹配。

在进行故障排除时，越具体越好。以下命令不够具体：

```
# pkg update
```

如果该命令失败，并显示了“该映像无更新可用”消息，则您可能希望尝试更具体的命令，如以下命令：

```
# pkg update "*@latest"
```

此命令更准确地定义了结束状态，并且可以生成更有指导性的错误消息。

运行基本检查

检查期望的系统结束状态以确保解决方案可行。在此基本检查期间，pkg 检查是否存在所有相关项的一个可靠版本，且所需的软件包互不排斥。

如果存在明显错误，则 pkg 将输出相应的错误消息并退出。

运行解析器

pkg(5) 使用一个计算引擎根据映像中的约束和用于安装的任何新软件包引入的约束来确定可以安装、更新或删除的软件包，解析器构成了此计算引擎的核心。

此问题是**布尔型可满足性问题**的一个示例，可以通过 SAT 解析器解决。

针对所有软件包的各种可能选择均指定了布尔变量，并且在那些软件包、任何所需的软件包等之间的所有相关项均以合取范式强制转换为布尔表达式。

生成的表达式集将传递到 MiniSAT。如果 MiniSAT 找不出任何解决方案，则错误处理代码会尝试遍历已安装的软件包集和已尝试的操作，并输出每个可能的选择被排除的原因。

如果当前已安装的软件包集符合要求，而不是其他软件包集符合要求，则 pkg 会报告无需执行任何操作。

如前所述，错误消息生成和特征是通过 pkg 的输入确定的。在发出到 pkg 的命令中提供尽可能具体的信息可以生成最有用的错误消息。

如果 MiniSAT 找到了一个可能的解决方案，优化阶段将开始。

优化解析器结果

优化阶段很有必要，因为没有办法来描述某些解决方案比其他解决方案更适合 SAT 解析器。相反，一旦找到一个解决方案，IPS 将向问题添加约束来隔离不太合意的选择，并隔离当前解决方案。然后，IPS 重复调用 MiniSAT 并重复以上操作，直到无法确定更多的解决方案。将采用最后一个成功的解决方案作为最好的解决方案。

确定解决方案的难度和可能的解决方案的数目成比例。关于期望的结果的信息越具体，生成解决方案的速度越快。

找到最能满足所引起问题的软件包 FMRI 的集合后，评估阶段开始。

评估操作

在评估阶段中，IPS 将比较系统上当前安装的软件包的结束状态，同时比较新旧软件包的软件包清单来确定三个列表：

- 要删除的操作。
- 要添加的操作。
- 要更新的操作。

然后，操作列表将通过以下方式进行更新：

- 对目录和链接操作进行引用计数，并执行中介链接处理。
- 对硬链接进行标记以便修复（如果其目标文件已更新）。如此操作是因为以对当前正在执行的进程安全的方式更新硬链接的目标会破坏硬链接。
- 正确处理可编辑文件在软件包间的移动以便不丢失任何用户编辑内容。
- 对操作列表进行排序，以便删除、添加和更新以正确的顺序发生。

反复核对所有当前已安装的软件包，以确保软件包没有冲突。示例冲突包括：将文件交付到相同位置的两个软件包、为同一目录提供了不同的目录属性的两个软件包。

如果存在冲突，则会报告冲突，且 pkg 退出并显示错误消息。

最后，对操作列表进行扫描以确定如果执行了该操作，是否有任何 SMF 服务需要重新启动；该更改是否可以应用于正在运行的系统；引导归档文件是否需要重新生成；是否存在所需的空间量。

下载内容

如果 pkg 在未使用 -n 标志的情况下运行，则处理将前进到下载阶段。

对于需要内容的每个操作，IPS 通过散列下载任何必需的文件并将其高速缓存。如果要检索的内容量很大，则该步骤可能需要花费一些时间。

下载完成后，如果更改是要应用于实时系统（根目录为 / 的映像）的，且需要重新引导，则会克隆正在运行的系统，目标映像也将切换到克隆。

执行操作

执行操作涉及在映像上实际执行特定于每个操作类型的安装或删除方法。

执行操作是从首先执行所有删除操作开始的。如果在要从系统中删除的目录中发现了任何意外内容，则会将该内容放置在 `/var/pkg/lost+found` 中。

然后，继续执行安装和更新操作。请注意，所有操作都是在所有软件包之间混合执行的。因此，单个软件包操作中的所有更改将一次性应用于系统，而不是逐个软件包进行应用。这允许软件包相互依赖，并安全地交换内容。有关文件更新方式的详细信息，请参见第 20 页中的 “文件操作”。

处理执行器

如果更改应用于实时系统，则此时会执行所有暂挂的执行器。这些执行器通常是 SMF 服务重新启动和刷新。一旦这些执行器启动，IPS 会立刻更新本地搜索索引。第 7 章，在软件包安装过程中自动进行系统更改中对执行器进行了详细论述。

更新引导归档文件

如果必要，将更新引导归档文件。

◆ ◆ ◆ 第 4 章

4 指定软件包相关项

相关项定义各个软件包如何相关。本章介绍了不同类型的 IPS 相关项以及如何使用它们来构造可运转的软件系统。

IPS 提供了各种不同的相关项类型，如第 24 页中的 “依赖操作”中所述。本章提供了有关如何使用相关项类型来控制所安装的软件的更详细信息。

相关项类型

在 IPS 中，除非满足所有软件包相关项，否则将无法安装软件包。IPS 允许软件包相互依赖（具有循环相关项）。IPS 还允许软件包同时具有依赖于同一软件包的不同种类的相关项。

本章中的每节都包含了一个示例 `depend` 操作，该操作与其在创建软件包期间在软件包清单中显示的一样。

require 相关项

最基本类型的相关项是 `require` 相关项。这些相关项通常用来表示功能性相关项，例如库或解释器（如 Python 或 Perl）。

如果软件包 `A@1.0` 包含依赖于软件包 `B@2` 的 `require` 相关项，那么如果安装了 `A@1.0`，则还必须安装版本 2 或更高版本的 `B` 软件包。接受更高版本的软件包反映了现有软件包的较新版本中隐含的二进制兼容性期望。

如果在 `depend` 操作中指定的软件包的任何版本都是可接受的，则您可以省略指定的 FMRI 的版本部分。

下面是 `require` 相关项的示例：

```
depend fmri=pkg:/system/library type=require
```

require-any 相关项

如果有多个软件包可以满足功能要求，则使用 require-any 相关项。如果事先没有满足相关项，IPS 将选择其中一个软件包进行安装。

例如，您可以使用 require-any 相关项来确保在系统上至少安装了 Perl 的一个版本。对版本控制的处理方式与对 require 相关项的处理方式相同。

下面是 require-any 相关项的示例：

```
depend type=require-any fmri=pkg:/editor/gnu-emacs/gnu-emacs-gtk \
    fmri=pkg:/editor/gnu-emacs/gnu-emacs-no-x11 \
    fmri=pkg:/editor/gnu-emacs/gnu-emacs-x11
```

optional 相关项

optional 相关项指定，如果安装了给定软件包，则该软件包必须处于给定版本或更高版本。

这种类型的相关项通常用来处理软件包用来传输内容的情况。在这种情况下，软件包的每个传输后版本将包含依赖于其他软件包的传输后版本的 optional 相关项，这样就不可能安装两个软件包的不兼容版本。在 optional 相关项上省略版本会使相关项变得无意义，但允许这样做。

下面是 optional 相关项的示例：

```
depend fmri=pkg:/x11/server/xorg@1.9.99 type=optional
```

conditional 相关项

conditional 相关项具有一个 predicate 属性和一个 fmri 属性。如果在 predicate 属性的值中指定的软件包以指定的版本或更高版本出现在系统上，conditional 相关项将被视为依赖于 fmri 属性中的软件包的 require 相关项。如果在 predicate 属性中指定的软件包未出现在系统上或以较低版本出现，则会忽略 conditional 相关项。

如果系统上存在必备基础软件包，通常会使用 conditional 相关项来安装软件包的可选扩展。

例如，同时具有 X11 和终端版本的编辑器软件包可能会代替单独的软件包中的 X11 版本，并且由于必备 X 客户机库软件包作为 predicate 存在，软件包可能会包括依赖于文本版本中的 X11 版本的 conditional 相关项。

下面是 conditional 相关项的示例：

```
depend type=conditional fmri=library/python-2/pycurl-26 \
    predicate=runtime/python-26
```

group 相关项

group 相关项用来构造软件包的组。

group 相关项将忽略指定的版本。指定的软件包的任何版本都可以满足此相关项。

指定的软件包是必需的，除非该软件包已经是以下操作之一的对象：

- 软件包已放置在避免列表中。有关避免列表的信息，请参见 pkg(1) 手册页。
- 已通过 pkg install --reject 拒绝了软件包。
- 已通过 pkg uninstall 卸载了软件包。

使用这三个选项，管理员可以取消选择作为 group 相关项的主体的软件包。如果使用了这三个选项中的任何一个，除非另一相关项随后需要该软件包，否则 IPS 在更新期间将不会重新安装该软件包。如果另一后续操作删除了新的相关项，则会再次卸载该软件包。

有关如何使用这些相关项的一个典型示例是，使构造的软件包包含依赖于系统典型用途所需的软件包的 group 相关项。一些示例可能是 solaris-large-server、solaris-desktop 或 developer-gnu。第 98 页中的 “Oracle Solaris 组软件包”显示了一组提供了 group 相关项的 Oracle Solaris 软件包。

安装组软件包可以确保能够通过后续更新将 OS 更新到较新的版本，将向系统中添加合适的软件包。

下面是 group 相关项的示例：

```
depend fmri=package/pkg type=group
```

origin 相关项

origin 相关项的存在是为了解决需要中间转换的升级问题。其缺省行为是指定在要更新的系统上必须存在的软件包的最低版本（如果已安装）。

例如，通常可能使用数据库软件包版本 5，该版本支持从版本 3 或更高版本进行升级，但不支持从更低版本升级。在这种情况下，当正在运行版本 3 时，版本 5 具有依赖于自身的 origin 相关项。因此，如果要全新安装版本 5，安装将继续进行。但是，如果安装了版本 1 的软件包，将无法将该软件包直接升级到版本 5。在这种情况下，pkg update database-package 将不会选择版本 5，而会选择版本 3 作为可升级到的最新版本。

可以通过将 root-image 属性设置为 true 来修改 origin 相关项的行为。在这种情况下，如果正在运行的系统中（而非要更新的映像中）存在指定的软件包，则该软件包必须使用指定的版本或更高版本。这通常用于操作系统问题（例如依赖于引导块安装程序的相关项）。

下面是 origin 相关项的示例：

```
depend fmri=pkg:/database/mydb@3.0 type=origin
```

incorporate 相关项

incorporate 相关项指定如果安装了给定的软件包，该软件包处于给定的版本以满足给定的版本准确性。例如，如果相关 FMRI 版本为 1.4.3，则低于 1.4.3 的版本或 1.4.4 及更高版本将不满足相关项。版本 1.4.3.7 不满足此示例相关项。

使用 incorporate 相关项的常用方法是：将许多此类相关项放置在同一个软件包中以在软件包版本空间中定义一个兼容的接口。包含这样的 incorporate 相关项集的软件包通常称为 incorporation。Incorporation 通常用来定义一起构建的且不单独进行版本控制的一组软件包。Oracle Solaris 中大量使用了 incorporate 相关项来确保将软件的兼容版本安装在一起。

下面是 incorporate 相关项的示例：

```
depend type=incorporate \
    fmri=pkg:/driver/network/ethernet/e1000g@0.5.11,5.11-0.175.0.0.0.2.1
```

parent 相关项

parent 相关项用于区域或其他子映像。在这种情况下，将仅在子映像中检查该相关项，该相关项指定在父映像或全局区域中必须存在的软件包和版本。指定的版本必须匹配指定的精度级别。

例如，如果 parent 相关项依赖于 A@2.1，则以 2.1 开头的任何 A 版本都将匹配。此相关项通常用来要求软件包在非全局区域与全局区域之间保持同步。作为快捷方式，可以将特殊的软件包名称 feature/package/dependency/self 用作包含此相关项的软件包的确切版本的同义词。

parent 相关项用来将安装在非全局区域中的关键操作系统组件（如 libc.so.1）与安装在全局区域中的内核保持同步。第 10 章，处理非全局区域中还论述了 parent 相关项。

下面是 parent 相关项的示例：

```
depend type=parent fmri=feature/package/dependency/self \
    variant.opensolaris.zone=nonglobal
```

exclude 相关项

如果相关软件包以指定的版本级别或更高级别安装在映像中，则包含 exclude 相关项的软件包将无法安装。

如果在 exclude 相关项的 FMRI 中省略了版本，则被排除的软件包的任何版本都无法与指定了该相关项的软件包同时安装。

exclude 相关项很少使用。这些约束可能会使管理员感到绊手绊脚，应尽量避免使用。

下面是 exclude 相关项的示例：

```
depend fmri=pkg:/x11/server/xorg@1.10.99 type=exclude
```

约束和冻结

通过谨慎地使用上述各种类型的 depend 操作，软件包可以定义可用来升级它们的方式。

约束可安装的软件包版本

通常情况下，您希望将一组软件包安装在一个系统上以便同时对其进行支持和升级：该组中的所有软件包要么全部更新，要么都不更新。这就是使用 incorporate 相关项的原因。

以下三个不完整的软件包清单显示了 foo 和 bar 软件包与 myincorp incorporation 软件包之间的关系。

以下摘录来自 foo 软件包清单：

```
set name=pkg.fmri value=foo@1.0
dir path=foo owner=root group=bin mode=0755
depend fmri=myincorp type=require
```

以下摘录来自 bar 软件包清单：

```
set name=pkg.fmri value=bar@1.0
dir path=bar owner=root group=bin mode=0755
depend fmri=myincorp type=require
```

以下摘录来自 myincorp 软件包清单：

```
set name=pkg.fmri value=myincorp@1.0
depend fmri=foo@1.0 type=incorporate
depend fmri=bar@1.0 type=incorporate
```

foo 和 bar 软件包都具有依赖于 myincorp incorporation 的 require 相关项。myincorp 软件包具有 incorporate 相关项，它们按以下方式对 foo 和 bar 软件包进行约束：

- foo 和 bar 软件包最多可升级到版本 1.0：到相关项中指定的版本号所定义的粒度级别。
- 如果 foo 和 bar 软件包已安装，则它们必须处于版本 1.0 或更高版本。

依赖于版本 1.0 的 incorporate 相关项允许版本 1.0.1 或版本 1.0.2.1，例如，但不允许版本 1.1、版本 2.0 或版本 0.9。当安装了在更高版本指定了 incorporate 相关项的更新的 incorporation 软件包时，将允许 foo 和 bar 软件包更新至这些更高的版本。

由于 foo 和 bar 具有依赖于 myincorp 软件包的 require 相关项，因此，必须始终安装该 incorporation 软件包。

放宽对可安装的软件包版本的约束

在某些情况下，您可能希望放宽 incorporation 约束。

bar 或许可以独立于 foo 运行，但您希望将 foo 保留在由 incorporation 中的 incorporate 相关项定义的版本系列中。

您可以使用侧面来放宽 incorporation 约束，以允许管理员有效地禁用某些 incorporate 相关项。第 5 章，允许变量中对侧面进行了更详细的论述。简而言之，侧面是一些特殊的属性，可以将这些属性应用于软件包中的操作以便作者可以将这些操作标记为可选的。

当以此方式用侧面属性对操作进行了标记时，可以使用 pkg change-facet 命令来启用或禁用包含这些侧面的操作。

根据约定，选择性地安装 incorporate 相关项的侧面被命名为 facet.version-lock.*name*，其中 *name* 是包含该 depend 操作的软件包的名称。

使用以上示例，myincorp 软件包清单可以包含以下行：

```
set name=pkg.fmri value=myincorp@1.0
depend fmri=foo@1.0 type=incorporate
depend fmri=bar@1.0 type=incorporate facet.version-lock.bar=true
```

缺省情况下，此 incorporation 包括了对 bar 软件包的 depend 操作，将 bar 约束到版本 1.0。以下命令可放宽该约束：

```
# pkg change-facet version-lock.bar=false
```

成功执行此命令后，bar 软件包将免受 incorporation 约束并且在需要时可以升级到版本 2.0。

冻结可安装的软件包版本

到目前为止，在软件包设计过程中已通过修改软件包清单应用了有关约束的论述内容。管理员还可以在运行时向系统应用约束。

使用 pkg freeze 命令，管理员可以防止给定软件包从它当前的已安装版本（包括时间戳）或在命令行上指定的版本发生更改。此功能与 incorporate 相关项同样有效。

有关 freeze 命令的更多信息，请参见 pkg(1) 手册页。

要向映像应用更复杂的相关项，请创建并安装包括这些相关项的软件包。

◆ ◆ ◆ 第 5 章

5 允许变量

本章介绍了如何向最终用户提供不同的安装选项。

互斥软件组件

Oracle Solaris 支持多种体系结构，SVR4 包管理系统常发生的一个错误是意外安装了错误体系结构的软件包。针对每种受支持的体系结构维护单独的 IPS 软件系统信息库对 ISV 没有吸引力，而且这样软件用户很容易出错。因此，IPS 支持在多种体系结构上安装单个软件包。

用于实现此功能的机制称作**变量**。变量允许目标映像的属性确定实际安装哪些软件组件。

变量包含两部分：名称和可能值列表。下表显示了在 Oracle Solaris 11 中定义的变量。

变量名	可能值
variant.arch	sparc、i386
variant.opensolaris.zone	global、nonglobal
variant.debug.*	true、false

变量出现在软件包中的以下两处：

- set 操作对变量进行命名并定义应用于此软件包的值。
- 如果任何操作只能针对 set 操作中指定的变量值的子集进行安装，则该操作具有一个标记，指定要根据其安装此操作的变量名称和值。

例如，交付符号链接 /var/ld/64 的软件包可能包含以下定义：

```
set name=variant.arch value=sparc value=i386
dir group=bin mode=0755 owner=root path=var/ld
dir group=bin mode=0755 owner=root path=var/ld/amd64 \
    variant.arch=i386
dir group=bin mode=0755 owner=root path=var/ld/sparcv9 \
    variant.arch=sparc
link path=var/ld/32 target=.
link path=var/ld/64 target=sparcv9 variant.arch=sparc
link path=var/ld/64 target=amd64 variant.arch=i386
```

请注意，同时交付到 SPARC 和 x86 上的组件不接收变量标记，但是只交付到其中一种体系结构的组件将接收相应的标记。操作可包含用于不同变量名称的多个标记。例如，某个软件包可能同时包含用于 SPARC 和 x86 的调试H和非调试二进制文件。

在 Oracle Solaris 中，内核组件通常从安装在区域中的软件包中省略，因为内核组件在非全局区域中没有用。所以，内核组件使用设置为 `global` 的 `opensolaris.zone` 变量进行标记，从而不在非全局区域中安装内核组件。通常在发布期间使用 `pkgmogrify(1)` 规则在清单中进行此操作。针对区域标记 `i386` 和 `sparc` 内部版本中的软件包，然后使用 `pkgmerge(1)` 合并来自 `sparc` 和 `i386` 内部版本的软件包。这比尝试手动构造此类软件包要远远可靠和快得多。

软件包开发者不能定义新变量。然而，开发者可以提供组件的调试版本（用 `variant.debug.*` 变量标记），并且如果发生问题用户可以选择此变量。变量名称空间的 `variant.debug.*` 部分预定义为使用缺省值 `false`。请记住变量是按每个映像设置的，所以确保选择一个合适的名称，此名称可合理解释软件的该部分，而且是唯一的。

变量标记在合并期间应用于因体系结构而不同的操作，包括相关项和 `set` 操作。如果软件包标记为不支持当前映像的某个变量值，则不考虑安装此软件包。

`pkgmerge(1)` 手册页提供了有关合并软件包的一些示例。如果需要，`pkgmerge` 命令可同时合并多个不同变量。

可选软件组件

属于主体的软件的一些部分可能是可选的，一些用户可能不想安装这些部分。示例包括不同语言环境、手册页和其他文档的本地化文件，以及只有开发者或 DTrace 用户需要的头文件。

过去，可选内容通过单独的软件包交付，软件包名称中附加 `-dev` 或 `-devel` 标识符。管理员通过安装这些可选软件包安装可选内容。此解决方案的一个问题是管理员必须通过检查可用软件包列表来发现要安装的可选软件包。

IPS 通过实现称为**侧面**的机制来交付可选软件包内容。侧面与变量相似：每个侧面有一个名称和值，操作可以针对不同侧面名称包含多个标记。在映像中，所有侧面的缺省值为 tru，特定侧面的缺省值可以显式设为 true 或 false。侧面名称空间是按层次结构组织的。`pkg` 客户机隐式将映像的 `facet.*` 设置为 true。映像中特定侧面的值是最长匹配侧面名称的值。

以下示例显示管理员如何能够包括手册页但排除在该映像中安装所有其他文档。手册页和其他文档可以与软件和其他管理员想要安装的内容在同一个软件包中。在软件包清单中，手册页使用 facet.doc.man=true 进行标记。例如，其他文档操作可以使用 facet.doc.pdf=true 或 facet.doc.html=true 进行标记。在映像中，管理员可以使用以下命令来包括手册页但排除所有其他文档：

```
# pkg change-facet facet.doc.*=false
# pkg change-facet facet.doc.man=true
```

同样，软件包清单中的操作可以用语言环境侧面来标记，如 facet.locale.de=true 或 facet.locale.fr=true。下列命令只在该映像中安装德语本地化。

```
# pkg change-facet facet.locale.*=false
# pkg change-facet facet.locale.de=true
```

如果操作包含多个侧面标记，任一侧面标记的值为 true 时，将安装该操作。使用 pkg facet 命令显示在映像中显式设置的侧面。

```
$ pkg facet
FACETS          VALUE
facet.doc.*     False
facet.doc.man   True
facet.locale.*  False
facet.locale.de True
```

使用 pkgmogrify 快速准确地将侧面标记添加至您的软件包清单，使用正则表达式匹配不同类型的文件。这将在第 6 章，以编程方式修改软件包清单中详细描述。

侧面还可以用于管理相关项，打开/关闭相关项取决于是否设置了侧面。有关 facet.version-lock.* 的讨论，请参见第 55 页中的 “约束和冻结”。

下列侧面对软件开发者可能有用：

```
facet.devel             facet.locale.es_BO      facet.locale.lt_LT
facet.doc               facet.locale.es_CL      facet.locale.lv
facet.doc.man           facet.locale.es_CO      facet.locale.lv_LV
facet.doc.pdf           facet.locale.es_CR      facet.locale.mk
facet.doc.info          facet.locale.es_DO      facet.locale.mk_MK
facet.doc.html          facet.locale.es_EC      facet.locale.ml
facet.locale.*          facet.locale.es_ES      facet.locale.ml_IN
facet.locale.af         facet.locale.es_GT      facet.locale.mr
facet.locale.af_ZA      facet.locale.es_HN      facet.locale.mr_IN
facet.locale.ar         facet.locale.es_MX      facet.locale.ms
facet.locale.ar_AE      facet.locale.es_NI      facet.locale.ms_MY
facet.locale.ar_BH      facet.locale.es_PA      facet.locale.mt
facet.locale.ar_DZ      facet.locale.es_PE      facet.locale.mt_MT
facet.locale.ar_EG      facet.locale.es_PR      facet.locale.nb
facet.locale.ar_IQ      facet.locale.es_PY      facet.locale.nb_NO
facet.locale.ar_JO      facet.locale.es_SV      facet.locale.nl
facet.locale.ar_KW      facet.locale.es_US      facet.locale.nl_BE
facet.locale.ar_LY      facet.locale.es_UY      facet.locale.nl_NL
facet.locale.ar_MA      facet.locale.es_VE      facet.locale.nn
facet.locale.ar_OM      facet.locale.et         facet.locale.nn_NO
```

```
facet.locale.ar_QA      facet.locale.et_EE      facet.locale.no
facet.locale.ar_SA      facet.locale.eu         facet.locale.or
facet.locale.ar_TN      facet.locale.fi         facet.locale.or_IN
facet.locale.ar_YE      facet.locale.fi_FI      facet.locale.pa
facet.locale.as         facet.locale.fr         facet.locale.pa_IN
facet.locale.as_IN      facet.locale.fr_BE      facet.locale.pl
facet.locale.az         facet.locale.fr_CA      facet.locale.pl_PL
facet.locale.az_AZ      facet.locale.fr_CH      facet.locale.pt
facet.locale.be         facet.locale.fr_FR      facet.locale.pt_BR
facet.locale.be_BY      facet.locale.fr_LU      facet.locale.pt_PT
facet.locale.bg         facet.locale.ga         facet.locale.ro
facet.locale.bg_BG      facet.locale.gl         facet.locale.ro_RO
facet.locale.bn         facet.locale.gu         facet.locale.ru
facet.locale.bn_IN      facet.locale.gu_IN      facet.locale.ru_RU
facet.locale.bs         facet.locale.he         facet.locale.ru_UA
facet.locale.bs_BA      facet.locale.he_IL      facet.locale.rw
facet.locale.ca         facet.locale.hi         facet.locale.sa
facet.locale.ca_ES      facet.locale.hi_IN      facet.locale.sa_IN
facet.locale.cs         facet.locale.hr         facet.locale.sk
facet.locale.cs_CZ      facet.locale.hr_HR      facet.locale.sk_SK
facet.locale.da         facet.locale.hu         facet.locale.sl
facet.locale.da_DK      facet.locale.hu_HU      facet.locale.sl_SI
facet.locale.de         facet.locale.hy         facet.locale.sq
facet.locale.de_AT      facet.locale.hy_AM      facet.locale.sq_AL
facet.locale.de_BE      facet.locale.id         facet.locale.sr
facet.locale.de_CH      facet.locale.id_ID      facet.locale.sr_ME
facet.locale.de_DE      facet.locale.is         facet.locale.sr_RS
facet.locale.de_LI      facet.locale.is_IS      facet.locale.sv
facet.locale.de_LU      facet.locale.it         facet.locale.sv_SE
facet.locale.el         facet.locale.it_CH      facet.locale.ta
facet.locale.el_CY      facet.locale.it_IT      facet.locale.ta_IN
facet.locale.el_GR      facet.locale.ja         facet.locale.te
facet.locale.en         facet.locale.ja_JP      facet.locale.te_IN
facet.locale.en_AU      facet.locale.ka         facet.locale.th
facet.locale.en_BW      facet.locale.ka_GE      facet.locale.th_TH
facet.locale.en_CA      facet.locale.kk         facet.locale.tr
facet.locale.en_GB      facet.locale.kk_KZ      facet.locale.tr_TR
facet.locale.en_HK      facet.locale.kn         facet.locale.uk
facet.locale.en_IE      facet.locale.kn_IN      facet.locale.uk_UA
facet.locale.en_IN      facet.locale.ko         facet.locale.vi
facet.locale.en_MT      facet.locale.ko_KR      facet.locale.vi_VN
facet.locale.en_NZ      facet.locale.ks         facet.locale.zh
facet.locale.en_PH      facet.locale.ks_IN      facet.locale.zh_CN
facet.locale.en_SG      facet.locale.ku         facet.locale.zh_HK
facet.locale.en_US      facet.locale.ku_TR      facet.locale.zh_SG
facet.locale.en_ZW      facet.locale.ky         facet.locale.zh_TW
facet.locale.eo         facet.locale.ky_KG
facet.locale.es_AR      facet.locale.lg
```

◆ ◆ ◆ 第 6 章

以编程方式修改软件包清单

本章介绍了如何对软件包清单进行计算机编辑以自动注释和检查清单。

第 2 章，使用 IPS 打包软件介绍了发布软件包所需的所有技术。本章则提供可帮助您发布大型软件包、发布大量软件包或在一段时间内重新发布软件包的附加信息。

您的软件包可能包含许多需要使用变量/侧面（如第 5 章，允许变量中所述）或服务重新启动（如第 7 章，在软件包安装过程中自动进行系统更改中所述）进行标记的操作。请使用 IPS `pkgmogrify` 实用程序快速、准确并重复地转换软件包清单，而不要通过手动编辑软件包清单或编写脚本或程序来完成此项工作。

`pkgmogrify` 实用程序采用两种类型的规则：transform 和 include。Transform 规则用于修改操作。Include 规则会导致处理其他文件。`pkgmogrify` 实用程序从文件中读取这些规则，并将其应用到指定的软件包清单。

Transform 规则

本节显示了一个 transform 规则示例，并介绍了所有 transform 规则都包含的各个部分。

在 Oracle Solaris 中，交付到名为 `kernel` 的子目录中的文件被视为内核模块并标记为需要重新引导。以下标记适用于其 `path` 属性值包含 `kernel` 的操作：

```
reboot-needed=true
```

为应用此标记，需要在 `pkgmogrify` 规则文件中指定以下规则：

```
<transform file path=.*kernel/.+ -> default reboot-needed true>
```

分隔符	规则括在 < 和 > 中。`->` 左侧的规则部分是选择部分或匹配部分。`->` 右侧的规则部分是操作的执行部分。
`transform`	规则的类型。
`file`	此规则仅应用于 `file` 操作。这称为规则的选择部分。

`path=.*kernel/.+`	仅转换 `path` 属性与正则表达式 `path=.*kernel/.+` 匹配的 `file` 操作。这称为规则的匹配部分。
`default`	将 `default` 后面的属性和值添加到尚未为该属性设置值的任一匹配操作。
`reboot-needed`	要设置的属性。
`true`	要设置的属性的值。

transform 规则的选择或匹配部分会受操作类型和操作属性值的限制。有关这些匹配规则工作原理的详细信息，请参见 `pkgmogrify` 手册页。典型用法是选择交付到文件系统指定区域的操作。例如，在以下规则中，可以使用 *operation* 确保缺省情况下 `usr/bin` 以及在 `usr/bin` 中提供的所有内容都为正确的用户或组。

```
<transform file dir link hardlink path=usr/bin.* -> operation>
```

`pkgmogrify(1)` 手册页介绍了 `pkgmogrify` 可以执行的许多操作，用于添加、删除、设置和编辑操作属性以及添加和删除整个操作。

Include 规则

通过 include 规则，可以在由不同清单重复使用的多个文件和子集之间传播转换。假定您需要交付两个软件包：`A` 和 `B`。这两个软件包都应该将各自的 `source-url` 设置为同一 URL，但只有 `B` 应该将其在 `/etc` 中的文件设置为 `group=sys`。

软件包 `A` 的清单应指定一个 include 规则，用于限制包含 `source-url` 转换的文件。软件包 `B` 的清单也应指定一个 include 规则，用于限制包含文件组设置转换的文件。最终，用于限制包含 `source-url` 转换的文件的 include 规则应添加到软件包 `B` 中或用于设置组的转换所在的文件中。

转换顺序

按在文件中遇到转换的顺序来应用这些转换。排序功能可用于简化转换的匹配部分。

假定交付到 `/foo` 中的所有文件都应该具有缺省组 `sys`，但交付到 `/foo/bar` 中的文件除外，这些文件应具有缺省组 `bin`。

您可以编写复杂的正则表达式，用于匹配以 `/foo` 开始的所有路径（但以 `/foo/bar` 开始的路径除外）。使用转换的排序功能可使此匹配过程更为简单。

对缺省转换进行排序时，始终都是从最具体到最一般的顺序进行排序。否则，将永远不会使用后面的规则。

对于此示例，请使用以下两个规则：

```
<transform file path=foo/bar/.* -> default group bin>
<transform file path=foo/.* -> default group sys>
```

因为您需要查找与仅交付一次的每个软件包匹配的模式，所以使用转换难以添加使用上述匹配的操作。pkgmogrify 工具可创建合成操作以帮助解决此问题。由于 pkgmogrify 处理清单，所以对于每个设置了 pkg.fmri 属性的清单，合成 pkg 操作均由 pkgmogrify 创建。可以通过将 pkg 操作视为实际存在于清单中，来针对此操作进行匹配。

例如，假定您希望将包含 Web 站点 example.com（在该站点中可以找到已交付软件的源代码）的操作添加到每个软件包中。以下转换可实现此目的：

```
<transform pkg -> emit set info.source-url=http://example.com>
```

打包的转换

为方便开发者使用，在打包 Oracle Solaris OS 时使用的一组转换可在 /usr/share/pkg/transforms 中的以下文件中找到：

developer	对交付到 /usr/.*/include 的 *.h 头文件、归档文件和 lint 库、pkg-config(1) 数据文件以及 autoconf(1) 宏设置 facet.devel。
documentation	在文档文件上设置各种 facet.doc.* 侧面。
locale	在特定于语言环境的文件上设置各种 facet.locale.* 侧面。
smf-manifests	添加指向任何已打包 SMF 清单上的 svc:/system/manifest-import:default 的 restart_fmri 执行器，以便系统在安装软件包后导入该清单。

◆ ◆ ◆ 第 7 章

7 在软件包安装过程中自动进行系统更改

本章介绍了如何使用服务管理工具 (Service Management Facility, SMF) 来自动处理作为软件包安装结果发生的任何必需的系统修改。

在软件包操作中指定系统修改

首先确定哪些操作会在安装、更新或删除时导致系统更改。例如，要实现第 13 页中的“软件自组装”中描述的软件自组装概念，需要进行一些系统更改。

需要针对每个软件包操作确定哪个现有的 SMF 服务可提供必要的系统更改。或者，编写一个提供所需功能的新服务，并确保将该服务交付到系统，如第 66 页中的“交付 SMF 服务”中所述。

确定了在安装时会导致系统更改的一组操作后，请在软件包清单中标记这些操作以促使该系统更改发生。导致系统更改的标记的值称作**执行器**。

可将下列执行器标记添加到清单中的任何操作：

`reboot-needed` 该执行器使用值 `true` 或 `false`。此执行器声明：如果软件包系统在实时映像上操作，则必须在新引导环境中更新或删除标记的操作。`be-policy` 映像属性控制新引导环境的创建。有关 `be-policy` 属性的更多信息，请参见 `pkg`(1) 手册页的“映像属性”部分。

SMF 执行器 这些执行器与 SMF 服务相关。

SMF 执行器使用单个服务 FMRI 作为值，可能包含通配字符以与多个 FMRI 匹配。如果同一服务 FMRI 被多个操作标记（可能跨多个正在操作的软件包），IPS 只触发该执行器一次。

下面的 SMF 执行器列表介绍了对服务 FMRI（每个指定的执行器的值）的影响。

`disable_fmri` 执行软件包操作前禁用指定的服务。

`refresh_fmri`	完成软件包操作后刷新指定的服务。
`restart_fmri`	完成软件包操作后重新启动指定的服务。
`suspend_fmri`	执行软件包操作前临时暂停指定的服务，并在完成软件包操作后启用该服务。

交付 SMF 服务

要交付新 SMF 服务，请创建一个交付 SMF 清单文件和方法脚本的软件包。

本节首先讨论交付任何新 SMF 服务的一般情况，然后讨论交付只运行一次的服务的特定情况。最后，本节提供了有关这些服务软件包的自组装的一些提示。

交付新的 SMF 服务

交付新 SMF 服务的软件包通常需要更改系统。

在 SVR4 包管理中，安装后脚本运行 SMF 命令来重新启动 `svc:/system/manifest-import:default` 服务。

在 IPS 中，将清单文件交付到 `lib/svc/manifest` 或 `var/svc/manifest` 的操作应该使用以下执行器进行标记：

```
restart_fmri=svc:/system/manifest-import:default
```

此执行器确保当添加、更新或删除清单时，`manifest-import` 服务将重新启动，从而添加、更新或删除通过该 SMF 清单交付的服务。

如果软件包添加到实时系统，一旦在打包操作期间将所有软件包都添加到了系统就会执行此操作。如果软件包添加到备用引导环境，则在引导环境的第一次引导期间执行该操作。

交付运行一次的服务

需要执行新软件环境的一次配置的软件包应交付 SMF 服务以执行此配置。

交付应用程序的软件包应包含以下操作：

```
file path=opt/myapplication/bin/run-once.sh owner=root group=sys mode=0755
file path=lib/svc/manifest/application/myapplication-run-once.xml owner=root group=sys \
mode=0644 restart_fmri=svc:/system/manifest-import:default
```

服务的 SMF 方法脚本可以包含进一步配置应用程序或修改系统使应用程序有效运行所需的任何内容。在此示例中，方法脚本写入简单的日志消息：

```
#!/usr/bin/sh
. /lib/svc/share/smf_include.sh
assembled=$(/usr/bin/svcprop -p config/assembled $SMF_FMRI)
if [ "$assembled" == "true" ] ; then
    exit $SMF_EXIT_OK
fi
svccfg -s $SMF_FMRI setprop config/assembled = true
svccfg -s $SMF_FMRI refresh
echo "This is output from our run-once method script"
```

通常，如果应用程序尚未配置，SMF 服务应仅执行工作。此示例方法脚本检查 `config/assembled`。备用方法是将服务和应用程序分开打包，然后使用方法脚本删除包含该服务的软件包。

当测试方法脚本时，在运行执行器的软件包安装前或安装后运行 `pkg verify`。比较每个运行的输出来确保脚本没有尝试修改任何未标记为可编辑的文件。

下面显示了此示例的 SMF 服务清单：

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<service_bundle type='manifest' name='MyApplication:run-once'>
<service
    name='application/myapplication/run-once'
    type='service'
    version='1'>
    <single_instance />
    <dependency
        name='fs-local'
        grouping='require_all'
        restart_on='none'
        type='service'>
            <service_fmri value='svc:/system/filesystem/local:default' />
    </dependency>
    <dependent
        name='myapplication_self-assembly-complete'
        grouping='optional_all'
        restart_on='none'>
        <service_fmri value='svc:/milestone/self-assembly-complete' />
    </dependent>
    <instance enabled='true' name='default'>
        <exec_method
            type='method'
            name='start'
            exec='/opt/myapplication/bin/run-once.sh'
            timeout_seconds='0'/>
        <exec_method
            type='method'
            name='stop'
            exec=':true'
            timeout_seconds='0'/>
        <property_group name='startd' type='framework'>
```

```
            <propval name='duration' type='astring' value='transient' />
        </property_group>
        <property_group name='config' type='application'>
            <propval name='assembled' type='boolean' value='false' />
        </property_group>
    </instance>
</service>
</service_bundle>
```

请注意，SMF 服务的 `startd/duration` 属性设置为 `transient`，所以 `svc.startd`(1M) 不跟踪此服务的进程。还要注意服务将自身添加为 `self-assembly-complete` 系统里程碑的相关项。

在 SMF 方法中支持软件包自组装

本节提供了编写 SMF 方法时支持软件包自组装的一些附加提示。

测试是否需要重新编译配置文件

如果每次运行方法脚本时通过已打包配置文件片段编译配置文件的成本较高，请考虑在方法脚本中使用以下测试。

对已打包配置文件片段的目录运行 `ls -t`，然后使用 `head -1` 来选择最近更改的版本。将此文件的时间戳与通过那些片段编译的未打包配置文件的时间戳比较，确定服务是否需要重新编译配置文件。

限制等待自组装完成的时间

以上所示的示例 SMF 服务清单为 `start` 方法定义了 `timeout_seconds='0'`。这意味着 SMF 将会无限期地等待自组装完成。

为了有助于调试，您可能想要对自组装过程强加一个有限超时，从而使 SMF 在出现问题时将服务下降至维护状态。

◆ ◆ ◆ 第 8 章

8 有关软件包更新的高级主题

本章讨论了如何重命名、合并、拆分软件包，移动软件包内容，交付应用程序的多个实现以及在引导环境间共享信息。

避免软件包内容冲突

出于性能考虑，解析器将仅根据在软件包中指定的相关项信息进行工作。对于大多数更新操作，该信息足以能够指示 IPS 自动安装正确的更新软件包。

其相关项指示可以同时安装但其内容冲突的软件包会导致冲突检查在预安装中失败。如果冲突检查失败，最终用户必须尝试解决问题，可能需要手动指定不同版本的某些软件包。内容冲突的一个示例是两个软件包安装同一个文件。

软件包开发者必须确保因约束相关项而无法安装冲突的软件包。使用 `pkglint` 实用程序可帮助搜索此类冲突。有关 `pkglint` 的更多信息，请参见第 38 页中的 “验证软件包”和 `pkglint(1)` 手册页。

重命名、合并和拆分软件包

软件组件的所需组织会因原始软件包中的错误、产品或其使用随时间的变化或周围软件环境的变化而发生变化。有时，只是软件包的名称需要更改。规划此类更改时，请考虑执行升级的用户，以确保不会出现无法预料的负面影响。

本节按 `pkg update` 注意事项复杂性从低到高的顺序讨论了以下三种类型的软件包重新组织：

1. 重命名单个软件包
2. 合并两个软件包
3. 拆分一个软件包

重命名单个软件包

重命名单个软件包很简单。IPS 提供了一种机制来指明软件包已重命名。

要重命名软件包，请发布现有软件包的新版本，使其包含以下两个操作：

- `set` 操作，使用以下格式：

```
set name=pkg.renamed value=true
```

- 依赖于新软件包的 `require` 相关项。

重命名的软件包无法交付除 `depend` 或 `set` 操作以外的内容。

新软件包必须确保它无法与重命名之前的原始软件包同时安装。如果同一 incorporate 相关项同时涵盖了这两个软件包，则此限制是自动实现的。如果没有自动实现，则新软件包必须在重命名的版本中包含依赖于旧版软件包的 `optional` 相关项。这可以确保解析器不会同时选择这两个软件包，若同时选择会导致冲突检查失败。

安装此已重命名软件包的用户将自动接收新名称的软件包，因为新软件包是旧版本的一个相关项。如果没有任何其他软件包依赖于已重命名软件包，则它将自动被从系统中删除。存在旧版本的软件会导致许多已重命名软件包显示为已安装。当该旧版本的软件被删除时，也会自动删除已重命名软件包。

软件包可以多次重命名而不会出现问题，但是建议不要这样做，因为这样会使用户混淆。

合并两个软件包

合并软件包也非常简单。以下两种情况是合并软件包的示例：

- 一个软件包在重命名的版本中兼并另一个软件包。
- 两个软件包都重命名为同一个新的软件包名称。

一个软件包兼并另一个软件包

假定软件包 `A@2` 必须兼并软件包 `B@3`。要完成此操作，请将软件包 `B` 重命名为软件包 `A@2`。不要忘记在 `A@2` 中包括对 `B@3` 的 `optional` 相关项，除非如上所述，两个软件包已合并且因此一起更新。将 `B` 升级到 `B@3` 的用户现在即安装了 `A`，因为 `A` 已兼并了 `B`。

重命名两个软件包

在这种情况中，将两个软件包都命名为合并后的新软件包的名称，在新软件包中包含依赖于旧软件包的两个 `optional` 相关项（如果没有以其他方式对这两个软件包进行约束）。

拆分一个软件包

拆分软件包时，请按第 70 页中的 “重命名单个软件包”中介绍的过程重命名所得到的每个新软件包。如果所得到的其中一个新软件包未重命名，则该软件包的拆分前和拆分后版本不兼容，并且在最终用户尝试更新软件包时可能会违反相关项逻辑。

重命名原始软件包，并包含依赖于拆分得到的所有新软件包的 `require` 相关项。这可确保包含依赖于原始软件包的相关项的任何软件包都会获得所有新的软件包部分。

拆分的软件包的某些组件可以作为合并部分兼并到现有软件包中。请参见第 70 页中的 “一个软件包兼并另一个软件包”。

使软件包过时

软件包过时是用以清空软件包的内容并将其从系统中删除的机制。过时的软件包不满足 `require` 相关项。如果已安装的软件包包含依赖于已过时软件包的 `require` 相关项，则更新将失败，除非已安装的软件包有未包含 `require` 相关项的较新版本可用。

使软件包过时是通过发布除以下 `set` 操作外未包含任何内容的新版本来实现的：

```
set name=pkg.obsolete value=true
```

可通过发布较新的版本使软件包不过时。在所安装的软件包已过时时进行更新的用户将丢失该软件包。在软件包过时之前就已进行更新并且在发布软件包的较新版本后才再次进行更新的用户将更新到该较新版本。

保留迁移的可编辑文件

更新软件包时的一个常见问题是可编辑文件的迁移，这包括在两个软件包之间移动文件以及更改文件在所安装文件系统中的位置。

在软件包之间迁移可编辑文件。

如果文件名和文件路径未发生更改，IPS 将尝试迁移在两个软件包之间移动的可编辑文件。在软件包之间移动文件的一个示例是重命名软件包。

在文件系统中迁移可编辑文件。

如果文件路径发生了更改，请确保指定了 `original_name` 属性以保留用户对文件所做的定制。

如果最初交付此文件的软件包中的 `file` 操作未包含属性 `original_name`，请在更新的软件包中添加该属性。将该属性的值设置为源软件包的名称，后跟一个冒号和文件的原始路径，没有前导 `/`。

在可编辑的文件上提供 `original_name` 属性后，不要更改其属性值。此值用作所有正向移动的唯一标识符，以便不管在更新时跳过了多少个版本号都可以正确保留用户的内容。

删除或重命名目录时移动未打包的内容

通常情况下，删除包含未打包内容的目录时会挽救未打包的内容，因为删除后对该目录的最后引用会消失。

如果某个目录更改了名称，IPS 会将此行为视为删除旧目录并创建新目录。在重命名或删除目录时仍在目录中的任何可编辑文件都将被挽救。

如果旧目录具有未打包的内容，如应该移至新目录的日志文件，请在新目录上使用 `salvage-from` 属性。例如，如果 `pkgA` 将目录从 `/opt/olddata/log` 重命名为 `/opt/newdata/log`，则在做出此更改的 `pkgA` 版本中，在 `dir` 操作上包括用于创建 `/opt/newdata/log` 的以下属性：

```
salvage-from=opt/olddata/log
```

任何时间的所有未打包内容都将迁移到新位置。

第 74 页中的 “交付要在引导环境之间共享的目录”中再次论述了 `salvage-from` 属性。

交付应用程序的多个实现

您可能希望为给定的应用程序交付具有如下特征的多个实现：

- 所有实现在映像中都可用。
- 其中一个实现被指定为首选实现。
- 首选实现具有指向其二进制文件（安装到某个常见目录，例如 `/usr/bin`）的符号链接，以方便搜索。
- 管理员可根据需要更改首选实现，无需添加或删除任何软件包。

交付应用程序的多个实现的一个示例为 GCC。Oracle Solaris 提供了多个版本的 GCC，每个版本均位于各自的软件包中，并且 `/usr/bin/gcc` 指向首选版本。

IPS 使用中介链接在单个映像中管理应用程序的多个实现。**中介链接**是指由 `pkg set-mediator` 和 `pkg unset-mediator` 命令控制的符号链接。对于软件包中用于交付应用程序的不同实现的 `link` 操作，可以说它们参与了**仲裁**。`pkg mediator` 命令列出了映像中的仲裁。有关 mediator 命令的信息，请参见 `pkg(1)` 手册页。

可以在 `link` 操作上设置以下属性来控制中介链接的交付方式：

`mediator`
: 指定由给定仲裁组（例如 `python`）中涉及的所有路径名称共享的仲裁名称空间中的条目。

  可基于 `mediator-version` 和 `mediator-implementation` 执行链接仲裁。给定路径名称的所有中介链接必须指定同一 `mediator`。但是，并非所有中介版本和实现都需要在给定路径上提供链接。如果仲裁不提供链接，则会在选定该仲裁时删除链接。

中介与特定版本和/或实现组合起来表示可选择供包管理系统使用的**仲裁**。

`mediator-version`
指定 `mediator` 属性描述的接口的版本（表示为非负整数的点分序列）。如果指定了 `mediator` 而未指定 `mediator-implementation`，则此属性是必需的。本地系统管理员可以显式设置要使用的版本。指定的值通常应当与交付链接的软件包的版本相匹配。例如，`runtime/python-26` 应使用 `mediator-version=2.6`，尽管不是必须这样做。

`mediator-implementation`
指定中介的实现。该属性可以与 `mediator-version` 属性一起指定或者用来代替后者。实现字符串不被视为有序的。如果系统管理员未明确指定，则是 `pkg(5)` 随机选择的一个字符串。

`mediator-implementation` 的值可以是由字母数字字符和空格组成的任意长度的字符串。如果实现本身可版本化或已版本化，则应在字符串结尾处在 `@` 符号后指定版本。版本表示为非负整数的点分序列。如果存在多个版本的实现，则缺省行为是选择最高版本的实现。

如果系统上仅安装了特定路径的实现仲裁链接的一个实例，则会自动选择该实例。如果以后安装了该路径的其他链接，除非应用供应商、站点或本地覆盖或者如果某一链接进行了版本中介，否则不会切换链接。

`mediator-priority`
在解决中介链接中的冲突时，如果可能，`pkg(5)` 会选择 `mediator-version` 值最大的链接。如果这不可能，`pkg(5)` 会基于 `mediator-implementation` 选择链接。`mediator-priority` 属性用于为常规冲突解决方案过程指定覆盖。如果未指定 `mediator-priority` 属性，则会应用缺省中介选择逻辑。

`mediator-priority` 属性可以使用以下值之一：

`vendor` 与未指定 `mediator-priority` 的链接相比，将优先选择该链接。

`site` 与值为 `vendor` 或未指定 `mediator-priority` 的链接相比，将优先选择该链接。

本地系统管理员可以覆盖上面所述的选择逻辑。

样例清单中的以下两个摘录参与了链接 `/usr/bin/myapp` 的仲裁。实现 1 的版本为 5.8.4：

```
set name=pkg.fmri value=pkg://test/myapp-impl-1@1.0,5.11:20120721T035233Z
file path=usr/myapp/5.8.4/bin/myapp group=sys mode=0755 owner=root
link path=usr/bin/myapp target=usr/myapp/5.8.4/bin/myapp mediator=myapp mediator-version=5.8.4
```

实现 2 的版本为 5.12：

```
set name=pkg.fmri value=pkg://test/myapp-impl-2@1.0,5.11:20120721T035239Z
file path=usr/myapp/5.12/bin/myapp group=sys mode=0755 owner=root
link path=usr/bin/myapp target=usr/myapp/5.12/bin/myapp mediator=myapp mediator-version=5.12
```

这两个软件包可以安装在同一映像中：

```
$ pkg list myapp-impl-1 myapp-impl-2
NAME (PUBLISHER)                  VERSION      IFO
myapp-impl-1                      1.0          i--
myapp-impl-2                      1.0          i--
```

使用 pkg mediator 命令可查看正在使用的仲裁：

```
$ pkg mediator
MEDIATOR VER. SRC. VERSION IMPL. SRC. IMPLEMENTATION
myapp    local     5.12    system
$ ls -al usr/bin/myapp
lrwxrwxrwx 1 root sys 23 Jul 21 16:58 usr/bin/myapp -> usr/myapp/5.12/bin/myapp
```

使用 pkg search 命令可查看参与 myapp 仲裁的其他软件包：

```
$ pkg search -ro path,target,mediator,mediator-version,pkg.shortfmri ::mediator:myapp
PATH          TARGET                    MEDIATOR MEDIATOR-VERSION PKG.SHORTFMRI
usr/bin/myapp usr/myapp/5.12/bin/myapp  myapp    5.12             pkg:/myapp-impl-2@1.0
usr/bin/myapp usr/myapp/5.8.4/bin/myapp myapp    5.8.4            pkg:/myapp-impl-1@1.0
```

使用 pkg set-mediator 命令可更改仲裁。以下示例更改作为首选版本的 myapp 版本：

```
# pkg set-mediator -V 5.8.4 myapp
            Packages to update: 2
           Mediators to change: 1
       Create boot environment: No
Create backup boot environment: No

PHASE                                 ITEMS
Removing old actions                    2/2
Updating modified actions               2/2
Updating image state                   Done
Creating fast lookup database          Done
Reading search index                   Done
Updating search index                   2/2
# ls -al usr/bin/myapp
lrwxrwxrwx  1 root    sys    24 Jul 21 17:02 usr/bin/myapp -> usr/myapp/5.8.4/bin/myapp
```

交付要在引导环境之间共享的目录

交付到引导环境 (boot environment, BE) 中的一些文件必须在 BE 之间共享，这样才能使系统在具有多个 BE 的环境中正常运行。一般情况下，IPS 不支持交付在 BE 之间共享的内容。在一个 BE 中更新的此类共享内容可能不满足其他 BE 的要求。本节介绍了如何交付在 BE 之间共享的内容。

IPS 已在 BE 之间共享了以下目录：

```
/var/audit
/var/cores
/var/crash
/var/mail
```

在每个 BE 中，这些目录是指向以下共享目录的符号链接：

```
/var/share/audit
/var/share/cores
/var/share/crash
/var/share/mail
```

这些共享目录均位于 VARSHARE 数据集中，它是挂载在 /var/share 上的一个共享数据集。

如果需要在 BE 之间共享其他数据但这些数据在 IPS 软件包中是作为非共享数据交付的，则管理员可能会将此类数据放在独立数据集或远程文件服务器中。不过，创建每目录数据集意味着要在每个区域中创建许多数据集，这是不理想的。

相反，请使用以下过程创建可交付共享数据集的软件包，或修改软件包以共享先前作为非共享内容交付的内容。IPS 支持将未共享内容的旧版软件包更新到在 BE 之间共享内容的同一软件包的较新版本。

▼ 如何将内容交付到共享目录

以下过程介绍了如何设计必须交付在 BE 之间共享的内容的软件包。

要在 BE 之间共享数据，请将一个共享数据集在引导期间挂载到 BE 中，并使用从 BE 中的相应位置指向该共享数据集的符号链接。在 BE 内，将基本目录结构交付到一个暂存目录。提供一个在引导期间将暂存在 BE 中的内容移动到共享数据集的 SMF 服务，并提供一个执行器来要求重新引导。

1 将共享内容交付到 BE 内的暂存区域中。

a. 提供一个暂存区域。

在您的软件包中，提供一个存储共享内容的暂存区域。例如，您可以提供一个名为 .migrate 的目录。

b. 提供共享结构。

将子目录交付到提供您希望在共享数据集中使用的目录结构的 .migrate 目录中。

c. 交付共享文件。

根据需要，将文件交付到暂存区域中的目录结构。不能共享其他文件系统对象，如链接。

如果交付到暂存区域的内容先前是作为非共享内容交付的，请在新的 `dir` 或 `file` 操作中使用 `salvage-from` 属性。在下例中，以前交付到 `/opt/myapplication/logs` 的内容现在将交付到可在 BE 之间共享的数据集。此共享数据集的暂存区域为 `/opt/.migrate`。

以下操作是以前交付的：

```
dir path=opt/myapplication/logs owner=daemon group=daemon mode=0755
```

以下操作是针对将共享的目录的新操作：

```
dir path=opt/.migrate/myapplication/logs owner=daemon group=daemon \
    mode=0755 reboot-needed=true salvage-from=/opt/myapplication/logs
```

第 72 页中的 “删除或重命名目录时移动未打包的内容”中还论述了 `salvage-from` 属性。

2 提供一个用以将内容移至共享数据集的脚本。

在引导期间，脚本可以作为 SMF 方法脚本的一部分运行，以将文件内容从暂存目录移至共享数据集。该脚本必须执行以下步骤：

a. 创建共享数据集。

SMF 方法脚本中的以下命令将创建挂载在 `/opt/share` 上的数据集 `rpool/OPTSHARE`。该数据集还可以供 `/opt` 中的其他共享内容使用。脚本应使用 `zfs list` 来测试此数据集是否已存在。

```
zfs create -o mountpoint=/opt/share rpool/OPTSHARE
```

b. 创建共享目录结构。

在共享数据集中，重新创建在 BE 的暂存目录下定义的目录结构中尚未存在的任何部分。

c. 移动文件内容。

将文件内容从暂存目录移至共享数据集。

3 交付从 BE 指向共享目录的符号链接。

以下操作将创建一个符号链接，该符号链接从先前打包的目录指向在系统重新引导时将由脚本在 `/opt/share` 中创建的共享目录：

```
link path=opt/myapplication/logs target=../../opt/share/myapplication/logs
```

4 添加一个执行器来要求重新引导。

这些目录项需要一个 `reboot-needed` 执行器才能正确支持第 13 页中的 “软件自组装”中提到的“不变区域”的更新。如果需要自组装，则在以只读方式重新引导之前，不变区域最多能够在读/写模式下引导到 `svc:/milestone/self-assembly-complete:default` 里程碑。有关更多信息，请参见 `zonecfg(1M)` 手册页中的 `file-mac-profile` 属性。

重新引导时，SMF 服务会将任何新的和挽救的目录内容移至共享数据集。`/opt/myapplication` 中的符号链接指向该共享数据集。

◆ ◆ ◆ 第 9 章

9 对 IPS 软件包进行签名

IPS 的一项重要功能是能够验证安装在用户机器上的软件确实与发布者最初指定的一致。验证已安装系统的能力对于用户和支持工程人员都非常重要。

可以为映像或特定发布者设置签名策略。策略包括忽略签名、验证现有签名、要求签名以及在信任链中要求特定通用名称。

本章介绍了 IPS 软件包签名以及开发者和质量保证组织如何对新软件包或现有已经签名的软件包进行签名。

对软件包清单进行签名

可以对 IPS 软件包清单进行签名，签名将成为清单的一部分。

定义签名操作

与其他所有清单内容一样，签名也表示为操作。由于清单包含所有软件包元数据（例如文件权限、所有权和内容散列），用于验证清单自发布后尚未更改的签名操作是系统验证的重要部分。

`signature` 操作形成一个包含已交付的二进制文件的树，从而使对已安装软件进行完整验证成为可能。

除了验证以外，签名也可以用来表示其他组织或第三方的批准。例如，一旦软件包符合产品使用的要求，内部 QA 组织就可以对软件包的清单进行签名。安装过程中可能需要这些批准。

一个清单可以有多个独立签名。可以添加或删除签名，而不会使存在的其他签名失效。此功能便于产品移交，使用签名来表示各个环节的完成。后续步骤可以随时选择删除以前的签名。

`signature` 操作使用以下格式：

```
signature hash_of_certificate algorithm=signature_algorithm \
    value=signature_value \
    chain="hashes_of_certificates_needed_to_validate_primary_certificate" \
    version=pkg_version_of_signature
```

有效负荷和 chain 属性表示保密性增强的电子邮件 (Privacy Enhanced Mail, PEM) 文件的包管理散列，包含可从原始系统信息库检索的 x.509 证书。有效负荷证书是用于验证 value 中的值的证书。value 是清单的消息文本的已签名散列，按照下面的讨论进行准备。

出现的其他证书需要形成从有效负荷证书到信任锚的证书路径。

支持两种类型的签名算法：

RSA　第一种签名算法是 RSA 算法组。RSA 签名算法的一个示例是 rsa-sha256。连字符后的字符串（此示例中的 sha256）指定用来将消息文本更改为 RSA 算法可以使用的单值的散列算法。

仅散列　第二种签名算法是仅计算散列。这种算法主要用于测试和过程验证目的，并将散列呈现为签名值。如果没有有效负荷证书散列，则说明使用了这种签名操作。如果映像配置为检查签名，将验证这种签名操作。然而，如果要求签名，此签名操作不能算作签名。以下示例显示了仅散列签名操作：

```
signature algorithm=hash_algorithm value=hash \
    version=pkg_version_of_signature
```

发布已签名的软件包清单

发布已签名的清单是包含两个步骤的过程。该过程不改变软件包，包括它的时间戳。

1. 将未签名的软件包发布至系统信息库。
2. 更新准备就绪的软件包，使用 pkgsign 命令将签名操作附加到系统信息库的清单中。

该过程允许除发布者之外的其他人添加签名操作而不使原始发布者的签名失效。例如，公司的 QA 部门可能想要对所有内部安装的软件包进行签名，来表示它们已被批准使用，但是不重新发布软件包，因为重新发布将会创建一个新的时间戳，使原始发布者的签名失效。

请注意，发布已签名软件包的唯一方法是使用 pkgsign 命令。如果发布的软件包已经包含签名，则系统将删除该签名并发出警告。pkgsign(1) 手册页包含如何使用 pkgsign 命令的示例。

包含变量的签名操作将被忽略。所以，对两个清单执行 pkgmerge 会使之前应用的所有签名失效。

注 – 对软件包进行签名应该是测试软件包之前软件包开发的最后一步。

对已签名的软件包进行故障排除

pkgsign 工具在对软件包进行签名时不对其输入执行所有可能检查。所以，检查已签名的软件包以确保它们在签名后可以正确安装是非常重要的。

本节显示了尝试安装或更新已签名软件包时出现的错误，并提供了这些错误的说明和问题的解决方案。

已签名软件包可能会因为已签名软件包的特有原因而安装或更新失败。例如，如果软件包的签名验证失败，或者如果信任链无法验证或固定到信任证书，则软件包将安装失败。

当安装签名的软件包时，下列映像和发布者属性将会影响软件包上执行的检查：

映像属性

- signature-policy
- signature-required-names
- trust-anchor-directory

发布者属性

- signature-policy
- signature-required-names

有关这些属性及其值的更多信息，请参见 pkg(1) 手册页。

未发现链证书

当信任链中的证书缺失或发生错误时，将会出现以下错误。

```
pkg install: The certificate which issued this certificate:
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs1_ch1_ta3/emailAddress=cs1_ch1_ta3
could not be found. The issuer is:
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=ch1_ta3/emailAddress=ch1_ta3
The package involved is: pkg://test/example_pkg@1.0,5.11-0:20110919T184152Z
```

在此示例中，当对软件包进行签名时，信任链中有三个证书。信任链源于信任锚（名为 ta3 的证书）。ta3 证书对名为 ch1_ta3 的链证书进行签名，ch1_ta3 对名为 cs1_ch1_ta3 的代码签名证书进行签名。

当 pkg 命令尝试安装软件包时，能够找到代码签名证书 cs1_ch1_ta3，但是找不到链证书 ch1_ta3，所以无法建立信任链。

该问题最常见的原因是未能向 pkgsign 的 -i 选项提供正确的证书。

未发现授权证书

以下错误与前一示例中显示的错误相似，但原因不同。

```
pkg install: The certificate which issued this certificate:
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs1_cs8_ch1_ta3/emailAddress=cs1_cs8_ch1_ta3
could not be found. The issuer is:
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs8_ch1_ta3/emailAddress=cs8_ch1_ta3
The package involved is: pkg://test/example_pkg@1.0,5.11-0:20110919T201101Z
```

在本例中，使用 cs1_cs8_ch1_ta3 证书对软件包进行签名，该证书则由 cs8_ch1_ta3 证书进行签名。

问题是 cs8_ch1_ta3 证书未被授权对其他证书进行签名。具体来说，cs8_ch1_ta3 证书将 basicConstraints 扩展设置为 CA:false 并标记为关键。

当 pkg 命令验证信任链时，没有找到允许对 cs1_cs8_ch1_ta3 证书进行签名的证书。因为信任链不能从叶到根进行验证，所以 pkg 命令阻止安装软件包。

不可信的自签名证书

当信任链以系统不信任的自签名证书结尾时，将会出现以下错误。

```
pkg install: Chain was rooted in an untrusted self-signed certificate.
The package involved is:pkg://test/example_pkg@1.0,5.11-0:20110919T185335Z
```

当您使用 OpenSSL 创建一个证书链以进行测试时，根证书通常是自签名，因为几乎没有理由让一个外部公司验证只用于测试的证书。

在测试情况下，有两种解决方案：

- 第一个解决方案是将作为信任链的根的自签名证书添加到 /etc/certs/CA 并刷新 system/ca-certificates 服务。这反映了客户可能遇到的情况，其中生产软件包由某个证书签名，该证书最终源于操作系统附带的作为信任锚的证书。
- 第二个解决方案是批准发布者的自签名证书，该发布者使用 --approve-ca-cert 选项和 pkg set-publisher 命令提供测试的软件包。

签名值与预期值不匹配

当无法使用操作声明中与用来为软件包签名的密钥配对的证书来验证 signature 操作中的值时，将出现以下错误。

```
pkg install: A signature in pkg://test/example_pkg@1.0,5.11-0:20110919T195801Z
could not be verified for this reason:
The signature value did not match the expected value. Res: 0
The signature's hash is 0ce15c572961b7a0413b8390c90b7cac18ee9010
```

这种错误可能有两种原因：

- 第一个可能的原因是软件包自签名后已进行更改。这是不大可能的，但是如果软件包清单自签名后进行了手动编辑，这也是有可能的。如果不进行手动干预，软件包自签名后不应该会更改，因为 pkgsend 在发布期间可以去除现有 signature 操作，软件包获得新时间戳时旧签名将变为无效。
- 第二个最可能的原因是用来对软件包进行签名的密钥和证书不是匹配的密钥/证书对。如果提供给 pkgsign 的 -c 选项的证书不是使用提供给 pkgsign 的 -k 选项的密钥创建的，尽管会对软件包签名，但是其签名将无法验证。

未知关键扩展

当信任链中的证书使用 pkg 不能理解的关键扩展时，将会出现以下错误。

```
pkg install: The certificate whose subject is
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs2_ch1_ta3/emailAddress=cs2_ch1_ta3
could not be verified because it uses a critical extension that pkg5 cannot
handle yet. Extension name:issuerAltName
Extension value:<EMPTY>
```

在 pkg 了解如何处理该关键扩展之前，唯一的解决方案是重新生成证书，而不包含有问题的关键扩展。

未知扩展值

以下错误与前一错误相似，只是问题不是在于不熟悉的关键扩展而是在于一个值，针对 pkg 理解的扩展，pkg 却无法理解该值。

```
pkg install: The certificate whose subject is
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs5_ch1_ta3/emailAddress=cs5_ch1_ta3
could not be verified because it has an extension with a value that pkg(5)
does not understand.
Extension name:keyUsage
Extension value:Encipher Only
```

在本例中，pkg 理解 keyUsage 扩展，但是不理解 Encipher Only 值。无论有问题的扩展是否关键，错误看起来都一样。

在 pkg 了解有问题的值之前，解决方案是将该值从扩展中删除，或将扩展全部删除。

未经授权使用证书

当证书用于它未经授权的用途时，将会出现以下错误。

```
pkg install: The certificate whose subject is
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=ch1_ta3/emailAddress=ch1_ta3
could not be verified because it has been used inappropriately.
The way it is used means that the value for extension keyUsage must include
'DIGITAL SIGNATURE' but the value was 'Certificate Sign, CRL Sign'.
```

在本例中，ch1_ta3 证书被用来对软件包进行签名。证书的 keyUsage 扩展意味着证书只有在对其他证书和 CRL（Certificate Revocation List，证书吊销列表）进行签名时才是有效的。

非预期的散列值

以下错误表示自上次从发布者检索证书后已对其进行了更改。

```
pkg install: Certificate
/tmp/ips.test.7149/0/image0/var/pkg/publisher/test/certs/0ce15c572961b7a0413b8390c90b7cac18ee9010
has been modified on disk. Its hash value is not what was expected.
```

提供的路径处的证书用于验证正在安装的软件包，但是磁盘上内容的散列与签名操作所预期的散列不匹配。

简单解决方案是删除证书并允许 pkg 重新下载证书。

已吊销的证书

以下错误指示处于要安装的软件包的信任链中的有问题证书已被该证书的颁发者吊销。

```
pkg install: This certificate was revoked:
/C=US/ST=California/L=Menlo Park/O=pkg5/CN=cs1_ch1_ta4/emailAddress=cs1_ch1_ta4
for this reason: None
The package involved is: pkg://test/example_pkg@1.0,5.11-0:20110919T205539Z
```

◆ ◆ ◆ 第 10 章

10 处理非全局区域

开发与区域工作一致的软件包通常工作量不大或不需要额外的工作。本章介绍了 IPS 如何处理区域并讨论了打包时需要考虑非全局区域的情况。

非全局区域的打包注意事项

考虑区域和打包时需要回答以下两个问题：

- 软件包中的内容是否具有可跨越全局区域与非全局区域之间的边界的接口?
- 要在非全局区域中安装软件包的多少内容?

软件包是否跨越全局区域与非全局区域之间的边界？

如果 pkgA 提供内核和用户级功能，且必须相应地更新接口的两端，则只要在非全局区域中更新 pkgA 就必须在已安装 pkgA 的任何其他区域中更新 pkgA。

要确保正确执行此更新，请在 pkgA 中使用 parent 相关项。如果单个软件包提供接口的两端，则依赖于 feature/package/dependency/self 的 parent 相关项可确保全局区域和非全局区域包含相同的软件包版本，以防止接口中出现版本差异。

parent 相关项还可确保如果软件包位于非全局区域，则其也将存在于全局区域中。

如果该接口跨越多个软件包，则包含接口的非全局区域端的软件包还必须包含软件包的 parent 相关项，以提供接口的全局区域端。第 51 页中的 “相关项类型”中也讨论了 parent 相关项。

非全局区域中应安装多少个软件包？

如果在非全局区域中安装软件包时应该安装软件包的所有内容，则无需对软件包执行任何操作即可使其正常运行。不过，对于软件包使用者来说，了解到软件包设计者已对区域安装进行了正确考虑并确定软件包可在区域中运行，可以更让人放心。因此，您应明确指出软件包在全局区域和非全局区域中都正常运行。为此，请将以下操作添加到清单中：

```
set name=variant.opensolaris.zone value=global value=nonglobal
```

如果无法在非全局区域中安装软件包中的任何内容（例如，只交付内核模块或驱动程序的软件包），则软件包应指出其无法在非全局区域进行安装。为此，请将以下操作添加到清单中：

```
set name=variant.opensolaris.zone value=global
```

如果可在非全局区域中安装软件包的一些（但不是所有）内容，则执行以下步骤：

1. 使用以下 set 操作指出该软件包可在全局区域和非全局区域中进行安装：

   ```
   set name=variant.opensolaris.zone value=global value=nonglobal
   ```
2. 标识仅在全局区域或仅在非全局区域中相关的操作。将以下属性指定到仅在全局区域中相关的操作：

   ```
   variant.opensolaris.zone=global
   ```

   将以下属性指定到仅在非全局区域中相关的操作：

   ```
   zone:variant.opensolaris.zone=nonglobal
   ```

如果某个软件包具有 parent 相关项或在全局区域和非全局区域中存在差异，则进行测试以确保软件包在全局区域和非全局区域中按照预期方式运行。

如果软件包具有依赖于自身的 parent 相关项，则全局区域必须配置可提供软件包的系统信息库作为该软件包的一个源。首先在全局区域安装软件包，然后在非全局区域安装以进行测试。

在非全局区域中安装软件包的故障排除

本节介绍用户尝试在非全局区域中安装软件包时可能遇到的问题。

具有依赖于自身的 parent 相关项的软件包

如果在非全局区域中安装软件包时遇到问题，请确保全局区域中的以下服务处于联机状态：

```
svc:/application/pkg/zones-proxyd:default
svc:/application/pkg/system-repository:default
```

确保非全局区域中的以下服务处于联机状态：

```
svc:/application/pkg/zones-proxy-client:default
```

这三个服务向非全局区域提供发布者配置，并提供一个信道，非全局区域可使用该通道向全局区域提供的、指定给系统发布者的系统信息库发送请求。

您不能在非全局区域中更新软件包，因为其具有依赖于自身的 parent 相关项。从全局区域启动更新；pkg 可同时更新非全局区域和全局区域。

在非全局区域中安装软件包后，对软件包功能进行测试。

不具有依赖于自身的 Parent 相关项的软件包

如果软件包没有依赖于自身的 parent 相关项，则无需在全局区域中配置发布者，并且不应在全局区域中安装软件包。在全局区域更新软件包将不会更新非全局区域中的该软件包。这种情况下，在全局区域更新软件包将导致测试旧版本的非全局区域软件包时发生意外结果。

在这种情况下，最简单的解决方案就是使发布者可用于非全局区域，并在非全局区域中安装和更新软件包。

如果该区域无法访问发布者的系统信息库，则在全局区域配置发布者会使 zones-proxy-client 和 system-repository 服务能够通过代理访问非全局区域的发布者。然后，在非全局区域安装并更新软件包。

◆ ◆ ◆ 第 11 章

11 修改已发布的软件包

有时，可能需要修改未生成的软件包。例如，您可能需要覆盖属性、将软件包的某部分替换为内部实现或删除系统上不允许使用的二进制文件。

本章介绍了如何按本地条件修改现有软件包。

重新发布软件包

使用 IPS 可轻松地重新发布包含所做修改的现有软件包，即使最初没有发布该软件包。您也可以重新发布已修改软件包的新版本，以便 `pkg update` 能够继续按用户预期方式工作。已修改的软件包将在映像中正确安装和更新。

当然，如果怀疑观察到的问题与修改的软件包之间存在任何关系，则运行包含已修改软件包的系统可能会对您的支持造成不利影响。

使用以下步骤可修改并重新发布软件包：

1. 使用 `pkgrecv(1)` 下载要以原始格式重新发布到指定目录的软件包。所有文件均由其散列值命名，而清单则命名为 `manifest`。请记住在 `http_proxy` 环境变量中设置所需的任何代理配置。
2. 使用 `pkgmogrify(1)` 对清单做出必要的修改。从内部软件包 FMRI 中删除所有时间戳，以防止在发布期间出现混淆。

   如果更改很显著，请使用 `pkglint(1)` 验证生成的软件包。
3. 使用 `pkgsend(1)` 重新发布软件包。请注意，此重新发布过程将去除软件包中存在的任何签名，并忽略由 `pkg.fmri` 指定的任何时间戳。要防止出现警告消息，请在 `pkgmogrify` 步骤中删除签名操作。

   如果您没有发布到软件包原始源的权限，请使用 `pkgrepo(1)` 创建系统信息库，然后使用以下命令在发布者搜索顺序中将新的发布者设置在原始发布者之前：

   ```
   # pkg set-publisher --search-before=original_publisher new_publisher
   ```

4. 如有必要，使用 pkgsign(1) 对软件包进行签名。要防止出现客户机高速缓存问题，请在安装软件包（即便是测试）之前对软件包进行签名。

更改软件包元数据

在以下示例中，pkg.summary 原始值已更改为 "IPS has lots of features"。使用 pkgrecv 的 --raw 选项下载了软件包。缺省情况下，只下载最新版本的软件包。然后将软件包重新发布到新的系统信息库。

```
$ mkdir republish; cd republish
$ pkgrecv -d . --raw -s http://pkg.oracle.com/solaris/release package/pkg
$ cd package* # The package name contains a '/' and is url-encoded.
$ cd *
$ cat > fix-pkg
# Change the value of pkg.summary
<transform set name=pkg.summary -> edit value '.*' "IPS has lots of features">
# Delete any signature actions
<transform signature -> drop>
# Remove the time stamp from the fmri so that the new package gets a new time stamp
<transform set name=pkg.fmri -> edit value ":20.+" "">
^D
$ pkgmogrify manifest fix-pkg > new-manifest
$ pkgrepo create ./mypkg
$ pkgsend -s ./mypkg publish -d . new-manifest
```

更改软件包发布者

另一种常见使用情况是在新的发布者名称下重新发布软件包。将软件包从多个系统信息库中整合到单个系统信息库时，这很有用。例如，您可能希望将软件包从具有多个不同开发团队的系统信息库中整合到用于集成测试的单个系统信息库中。

要在新的发布者名称下重新发布，请使用上述示例中显示的 pkgrecv、pkgmogrify、pkgrepo 和 pkgsend 步骤。

以下样例转换将发布者更改为 mypublisher：

```
<transform set name=pkg.fmri -> edit value pkg://[^/]+/ pkg://mypublisher/>
```

可以使用简单的 shell 脚本迭代系统信息库中的所有软件包。使用 pkgrecv --newest 命令的输出仅处理系统信息库中的最新软件包。

以下脚本将上述转换保存在名为 change-pub.mog 的文件中，然后从 development-repo 重新发布到新的系统信息库 mypublisher，从而在此过程中更改了软件包发布者：

```
#!/usr/bin/ksh93
pkgrepo create mypublisher
pkgrepo -s mypublisher set publisher/prefix=mypublisher
```

```
mkdir incoming
for package in $(pkgrecv -s ./development-repo --newest); do
    pkgrecv -s development-repo -d incoming --raw $package
done
for pdir in incoming/*/* ; do
    pkgmogrify $pdir/manifest change-pub.mog > $pdir/manifest.newpub
    pkgsend -s mypublisher publish -d $pdir $pdir/manifest.newpub
done
```

可以修改此脚本，使其能够执行各种任务，如仅选择某些软件包、对软件包的版本控制方案做出其他更改以及在其重新发布每个软件包时显示进度。

◆ ◆ ◆ 附 录 A

对软件包进行分类

本附录显示了软件包信息分类方案定义。

指定分类

软件包管理器 GUI 使用 `info.classification` 软件包属性，以方案 `org.opensolaris.category.2008` 按类别显示软件包。用户还可以使用 `pkg search` 命令来显示具有给定分类的软件包。

使用 `set` 操作为软件包指定一个分类，如下例所示：

```
set name=info.classification \
    value="org.opensolaris.category.2008:System/Administration and Configuration"
```

类别和子类别由正斜杠字符分隔。属性值中有空格时需要用引号括起来。

一个软件包可以有多个分类，如下例所示：

```
set name=info.classification \
    value="org.opensolaris.category.2008:Meta Packages/Group Packages" \
    value="org.opensolaris.category.2008:Web Services/Application and Web Servers"
```

分类值

已定义了以下类别和子类别值：

Meta Packages

- Group Packages
- Incorporations

Applications

- Accessories
- Configuration and Preferences
- Games
- Graphics and Imaging
- Internet
- Office
- Panels and Applets
- Plug-ins and Run-times
- Sound and Video
- System Utilities
- Universal Access

Desktop (GNOME)

- Documentation
- File Managers
- Libraries
- Localizations
- Scripts
- Sessions
- Theming
- Trusted Extensions
- Window Managers

Development

- C
- C++
- Databases
- Distribution Tools
- Editors
- Fortran
- GNOME and GTK+
- GNU
- High Performance Computing
- Java
- Objective C
- Other Languages
- PHP
- Perl
- Python
- Ruby

- Source Code Management
- Suites
- System
- X11

Drivers

- Display
- Media
- Networking
- Other Peripherals
- Ports
- Storage

System

- Administration and Configuration
- Core
- Databases
- Enterprise Management
- File System
- Fonts
- Hardware
- Internationalization
- Libraries
- Localizations
- Media
- Multimedia Libraries
- Packaging
- Printing
- Security
- Services
- Shells
- Software Management
- Text Tools
- Trusted
- Virtualization
- X11

Web Services

- Application and Web Servers
- Communications

◆ ◆ ◆ 附 录 B

如何使用 IPS 打包 Oracle Solaris OS

本附录介绍了 Oracle 如何使用 IPS 功能打包 Oracle Solaris OS，以及如何使用各种相关项类型定义 OS 的软件包工作集。

本附录还提供另一个有关使用 IPS 管理一组复杂软件的具体示例。

Oracle Solaris 软件包版本控制

第 16 页中的 “软件包标识符：FMRI”介绍了 `pkg.fmri` 属性以及版本字段的不同组件，包括如何使用版本字段支持软件开发的不同模型。本节介绍了 Oracle Solaris OS 如何使用版本字段，并带您深入了解细粒度版本控制方案为何很有用的原因。在您的软件包中，无需遵循 Oracle Solaris OS 使用的同一版本控制方案。

以下样例软件包 FMRI 中版本字符串的每个部分的含义如下所示：

```
pkg://solaris/system/library@0.5.11,5.11-0.175.1.0.0.2.1:20120919T082311Z
```

`0.5.11`
组件版本。对于属于 Oracle Solaris OS 的软件包，这是 OS `major.minor` 版本。对于其他软件包，这是上游版本。例如，以下 Apache Web Server 软件包的组件版本为 2.2.22：

```
pkg:/web/server/apache-22@2.2.22,5.11-0.175.1.0.0.2.1:20120919T122323Z
```

`5.11`
内部版本。此版本用于定义为其构建此软件包的 OS 发行版。对于为 Oracle Solaris 11 创建的软件包，内部版本应始终为 5.11。

`0.175.1.0.0.2.1`
分支版本。Oracle Solaris 软件包显示软件包 FMRI 中版本字符串的分支版本部分中的以下信息：

`0.175` 主发行编号。主要或市场开发发行版的内部版本号。在此示例中，`0.175` 表示 Oracle Solaris 11。

`1`
: 更新发行编号。此 Oracle Solaris 发行版的更新发行版号。对于 Oracle Solaris 发行版的第一个客户交付版本，更新值为 0，该发行版第一次更新后值为 1，第二次更新后值为 2，以此类推。在此示例中，`1` 表示 Oracle Solaris 11.1。

`0`
: SRU 编号。此更新发行版的 Support Repository Update (SRU) 编号。SRU 仅包含错误修复，不包括新增功能。Oracle Support Repository 仅可用于具有支持合同的系统。

`0`
: 保留。该字段当前未用于 Oracle Solaris 软件包。

`2`
: SRU 内部版本号。SRU 的内部版本号，或主要发行版的更新编号。

`1`
: 夜间生成的内部版本号。单个每日内部版本的内部版本号。

`20120919T082311Z`
: 时间戳。时间戳在发布软件包时定义。

Oracle Solaris Incorporation 软件包

Oracle Solaris 通过一组软件包交付，每组软件包都受 incorporation 约束。

每个 incorporation 基本上可以代表开发每组软件包的组织，尽管在软件包内部存在一些跨 incorporation 的相关项。Oracle Solaris 中包含以下 incorporation 软件包 (`pkg list *incorporation`)：

```
pkg:/consolidation/SunVTS/SunVTS-incorporation
pkg:/consolidation/X/X-incorporation
pkg:/consolidation/admin/admin-incorporation
pkg:/consolidation/cacao/cacao-incorporation
pkg:/consolidation/cde/cde-incorporation
pkg:/consolidation/cns/cns-incorporation
pkg:/consolidation/dbtg/dbtg-incorporation
pkg:/consolidation/desktop/desktop-incorporation
pkg:/consolidation/desktop/gnome-incorporation
pkg:/consolidation/gfx/gfx-incorporation
pkg:/consolidation/install/install-incorporation
pkg:/consolidation/ips/ips-incorporation
pkg:/consolidation/java/java-incorporation
pkg:/consolidation/jdmk/jdmk-incorporation
pkg:/consolidation/l10n/l10n-incorporation
pkg:/consolidation/ldoms/ldoms-incorporation
pkg:/consolidation/man/man-incorporation
pkg:/consolidation/nspg/nspg-incorporation
pkg:/consolidation/nvidia/nvidia-incorporation
pkg:/consolidation/osnet/osnet-incorporation
pkg:/consolidation/sfw/sfw-incorporation
pkg:/consolidation/sic_team/sic_team-incorporation
pkg:/consolidation/solaris_re/solaris_re-incorporation
pkg:/consolidation/sunpro/sunpro-incorporation
pkg:/consolidation/ub_javavm/ub_javavm-incorporation
pkg:/consolidation/userland/userland-incorporation
```

```
pkg:/consolidation/vpanels/vpanels-incorporation
pkg:/consolidation/xvm/xvm-incorporation
```

其中每个 incorporation 都包含以下信息：

- 软件包元数据。
- 类型为 `incorporate` 的相关项，有时包含 `variant.arch` 变量，用以指示特定于给定体系结构的相关项。有关 `incorporate` 相关项和 `variant.arch` 变量的更多信息，请参见第 54 页中的 “incorporate 相关项”和第 57 页中的 “互斥软件组件”。
- `license` 操作，可确保在安装 incorporation 时显示许可证。有关 `license` 操作的更多信息，请参见第 26 页中的 “许可证操作”。

系统上提供的每个软件包都包含依赖于上述其中一个 incorporation 的 `require` 相关项。有关更多信息，请参见第 51 页中的 “require 相关项”。

Oracle Solaris 还包括名为 `entire` 的特殊 incorporation。`entire` incorporation 通过同时包含依赖于每个 incorporation 软件包的 `require` 和 `incorporate` 相关项，将所有其他 incorporation 限制为同一内部版本。这样，`entire` incorporation 将定义一个软件表面，以便所有软件包作为单个组升级。

释放相关项约束

上述列出的一些 incorporation 使用 `facet.version-lock.*` 侧面，允许管理员使用 `pkg change-facet` 命令释放对指定软件包的 incorporation 的约束。有关更多信息，请参见第 56 页中的 “放宽对可安装的软件包版本的约束”。

例如，`pkg:/consolidation/userland/userland-incorporation` 软件包包含以下 `facet.version-lock.*` 定义：

```
..
depend type=incorporate \
    fmri=pkg:/library/python-2/subversion@1.6.16-0.175.0.0.0.2.537 \
    facet.version-lock.library/python-2/subversion=true
depend type=incorporate \
    fmri=pkg:/library/security/libassuan@2.0.1-0.175.0.0.0.2.537 \
    facet.version-lock.library/security/libassuan=true
depend type=incorporate \
    fmri=pkg:/library/security/openssl/openssl-fips-140@1.2-0.175.0.0.0.2.537 \
    facet.version-lock.library/security/openssl/openssl-fips-140=true
depend type=incorporate fmri=pkg:/mail/fetchmail@6.3.21-0.175.0.0.0.2.537 \
    facet.version-lock.mail/fetchmail=true
depend type=incorporate \
    fmri=pkg:/network/chat/ircii@0.2006.7.25-0.175.0.0.0.2.537 \
    facet.version-lock.network/chat/ircii=true
depend type=incorporate \
    fmri=pkg:/print/cups/filter/foomatic-db-engine@0.20080903-0.175.0.0.0.2.537 \
    facet.version-lock.print/cups/filter/foomatic-db-engine=true
depend type=incorporate \
    fmri=pkg:/print/filter/gutenprint@5.2.4-0.175.0.0.0.2.537 \
```

```
    facet.version-lock.print/filter/gutenprint=true
depend type=incorporate fmri=pkg:/runtime/erlang@12.2.5-0.175.0.0.0.2.537 \
facet.version-lock.runtime/erlang=true
    ..
```

entire 软件包还包含 version-lock 侧面。在此情况下，侧面允许从 entire incorporation 中删除指定的 incorporation。但这样会导致系统不受支持。这些软件包应仅在 Oracle 支持人员的设备上处于未锁定状态。

Oracle Solaris 组软件包

Oracle Solaris 定义了包含 group 相关项的多个组软件包。有关 group 相关项的更多信息，请参见第 53 页中的 “group 相关项”。通过这些组软件包，可以方便地安装常见软件包集。

Oracle Solaris 中包含以下软件包 (pkg list -a group*)：

```
pkg:/group/feature/amp
pkg:/group/feature/developer-gnu
pkg:/group/feature/multi-user-desktop
pkg:/group/feature/storage-avs
pkg:/group/feature/storage-nas
pkg:/group/feature/storage-server
pkg:/group/feature/trusted-desktop
pkg:/group/system/solaris-auto-install
pkg:/group/system/solaris-desktop
pkg:/group/system/solaris-large-server
pkg:/group/system/solaris-small-server
```

solaris-small-server 组软件包将按用于安装非全局区域 (/usr/share/auto_install/manifest/zone_default.xml) 的缺省 AI 清单安装。有关更多信息，请参见 solaris(5)。

属性和标记

本节介绍了常规和 Oracle Solaris 操作属性以及 Oracle Solaris 属性标记。

信息属性

以下属性不是正确安装软件包所必需的属性，但具有共享约定可以降低混淆发布者和用户的可能性。

info.classification
: 有关 info.classification 属性的信息，请参见第 22 页中的 “设置操作”。请参见附录 A，对软件包进行分类中的分类列表。

info.keyword
: 其他术语列表，应使搜索返回此软件包。

`info.maintainer`
用户可阅读的字符串，描述了提供软件包的实体。此字符串应该为名称、个人姓名和电子邮件或组织名称。

`info.maintainer-url`
与提供软件包的实体关联的 URL。

`info.upstream`
用户可阅读的字符串，描述了创建软件的实体。此字符串应该为名称、个人姓名和电子邮件或组织名称。

`info.upstream-url`
与实体关联的 URL，此实体可以创建在软件包中交付的软件。

`info.source-url`
软件包对应的源代码包的 URL（如果适用）。

`info.repository-url`
软件包对应的源代码系统信息库的 URL（如果适用）。

`info.repository-changeset`
`info.repository-url` 中包含的源代码的版本的变更集 ID。

Oracle Solaris 属性

`org.opensolaris.arc-caseid`	与 ARC（Architecture Review Committee，体系结构审查委员会）案例关联的一个或多个案例标识符（如 PSARC/2008/190），或与软件包交付的组件关联的多个案例。
`org.opensolaris.smf.fmri`	一个或多个 FMRI，代表由此软件包交付的 SMF 服务。`pkgdepend` 将自动为包含 SMF 服务清单的软件包生成这些属性。请参见 `pkgdepend(1)` 手册页。

特定于组织的属性

要为软件包提供其他元数据，请对属性名称使用特定于组织的前缀。组织可以使用此方法为在该组织中开发的软件包提供其他元数据，或修改现有软件包的元数据。要修改现有软件包的元数据，您必须对在其中发布该软件包的系统信息库具有控制权。例如，服务组织可能会引入一个名为 `service.example.com,support-level` 或 `com.example.service,support-level` 的属性，用以说明对软件包及其内容的支持级别。

Oracle Solaris 标记

`variant.opensolaris.zone`	指定可以将软件包中的哪些操作可以安装在非全局区域、全局区域或者既可以安装在非全局区域也可以安装在全局区域。有关更多信息，请参见第 10 章，处理非全局区域。