应用程序包开发者指南

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

文件号码 820–5500–12
2009 年 4 月

090511@22510

目录

前言

《应用程序包开发者指南》提供设计、生成和验证软件包的逐步说明以及相关的背景信息。本指南还包括在软件包创建过程中可能非常有用的高级技术。

注 – 此 Solaris™ 发行版支持使用以下 SPARC® 和 x86 系列处理器体系结构的系统：UltraSPARC®、SPARC64、AMD64、Pentium 和 Xeon EM64T。支持的系统可以在 `http://www.sun.com/bigadmin/hcl` 上的 Solaris OS: Hardware Compatibility Lists 中找到。本文档列举了在不同类型的平台上进行实现时的所有差别。

在本文档中，这些与 x86 相关的术语表示以下含义：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 指出了有关 AMD64 或 EM64T 系统的特定 64 位信息。
- “32 位 x86”指出了有关基于 x86 的系统的特定 32 位信息。

若想了解本发行版支持哪些系统，请参见 Solaris OS: Hardware Compatibility Lists。

目标读者

本书面向负责设计和生成软件包的应用程序开发者。

虽然本书的许多内容都针对软件包开发新手，但也包含对经验丰富的软件包开发者有用的信息。

本书的结构

下表介绍了本书中的各章。

章节名称	章节说明
第 1 章，设计软件包	介绍软件包组件和软件包设计准则，此外还介绍了相关的命令、文件和脚本。

章节名称	章节说明
第 2 章，生成软件包	介绍生成软件包的过程和所需任务，此外还提供了每项任务的逐步说明。
第 3 章，增强软件包的功能（任务）	提供向软件包中添加可选功能的逐步说明。
第 4 章，验证和转换软件包	介绍如何验证软件包的完整性，以及如何将软件包转换为分发介质。
第 5 章，软件包创建案例研究	提供创建软件包的案例研究。
第 6 章，创建软件包的高级技术	介绍创建软件包的高级技术。
词汇表	定义本书中使用的术语。

相关书籍

以下文档可从零售书商处购买，这些书籍可以提供有关生成 System V 软件包的其他背景信息。

- 《System V Application Binary Interface》
- 《System V Application Binary Interface - SPARC Processor Supplement》
- 《System V Application Binary Interface - Intel386 Processor Supplement》

文档、支持和培训

Sun Web 站点提供有关以下附加资源的信息：

- 文档 (`http://www.sun.com/documentation/`)
- 支持 (`http://www.sun.com/support/`)
- 培训 (`http://www.sun.com/training/`)

Sun 欢迎您提出意见

Sun 致力于提高其文档的质量，并十分乐意收到您的意见和建议。要分享您的意见，请访问 `http://docs.sun.com` 并单击 "Feedback"。

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:`
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

下表列出了 C shell、Bourne shell 和 Korn shell 的缺省 UNIX® 系统提示符和超级用户提示符。

表 P–2 shell 提示符

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

◆ ◆ ◆ 第 1 章

1 设计软件包

在生成软件包之前，您需要了解需要创建哪些文件以及需要执行哪些命令。您还需要考虑您的应用程序软件的要求以及客户的需要。您的客户是将要安装您的软件包的管理员。本章讨论生成软件包之前您应该了解和考虑的文件、命令和准则。

以下是本章中信息的列表。

在何处找到打包任务

使用以下任务图可以找到用于生成和验证软件包的逐步说明。

什么是软件包？

应用程序软件是以被称为**软件包**的单位交付的。软件包是软件产品所需的文件和目录的集合。软件包通常是在完成应用程序代码开发后由应用程序开发者设计和生成的。软件产品需要生成到一个或多个软件包中，以便可以轻松地将其转换为分发介质。然后，便可以大量生产该软件产品，并由管理员进行安装。

软件包是具有所定义格式的文件和目录的集合。该格式符合应用程序二进制接口 (application binary interface, ABI)，ABI 是对系统 V 接口定义的补充。

软件包组件

软件包组件分为两类。

- **软件包对象**是要安装的应用程序文件。
- **控制文件**控制是否安装软件包以及软件包的安装方式和位置。

控制文件又分为两类：**信息文件**和**安装脚本**。有些控制文件是必需的，有些控制文件是可选的。

要打包您的应用程序，必须首先创建组成软件包所必需的组件以及任何可选组件。然后，您可以使用 `pkgmk` 命令来生成软件包。

要生成软件包，您必须提供以下各项：

- 软件包对象（应用程序软件的文件和目录）
- 两个必需的信息文件（`pkginfo` 和 `prototype` 文件）
- 可选的信息文件
- 可选的安装脚本

下图描述了软件包的内容。

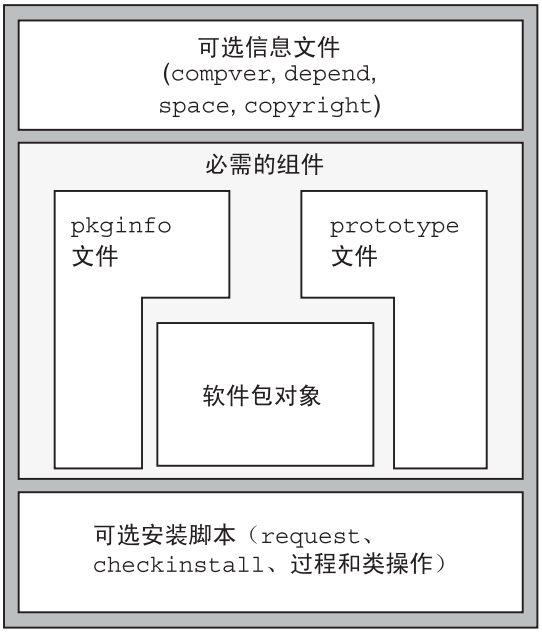


图 1–1 软件包的内容

必需的软件包组件

在生成软件包之前，必须创建以下组件：

- 软件包对象

  这些组件用于组成应用程序。它们可由以下各项组成：

  - 文件（可执行文件或数据文件）
  - 目录
  - 命名管道
  - 链接
  - 设备

- `pkginfo` 文件

  `pkginfo` 文件是必需的软件包信息文件，用于定义参数值。参数值包括软件包缩写、软件包全名和软件包体系结构。有关更多信息，请参见第 23 页中的 “创建 pkginfo 文件”和 `pkginfo(4)` 手册页。

  注 – 有两个 `pkginfo(1)` 手册页。第一个手册页介绍了用于显示有关已安装软件包信息的第 1 节中的命令。第二个手册页介绍了用于描述软件包特征的第 4 节中的文件。访问手册页时，请确保指定适用的手册页节。例如： `man -s 4 pkginfo`。

- `prototype` 文件

  `prototype` 文件是必需的软件包信息文件，用于列出软件包的组件。每个软件包对象、信息文件和安装脚本都存在一个相应的条目。这类条目由描述各个组件的若干信息字段组成，包括组件的位置、属性和文件类型。有关更多信息，请参见第 28 页中的 “创建 prototype 文件”和 `prototype(4)` 手册页。

可选软件包组件

软件包信息文件

您可以在软件包中包含四个可选软件包信息文件：

- `compver` 文件

  定义与此版本软件包兼容的以前软件包版本。

- `depend` 文件

  指出与您的软件包有特殊关系的其他软件包。

- `space` 文件

  定义目标环境的磁盘空间要求，这会超过 `prototype` 文件中定义的对象所要求的空间。例如，对于在安装时动态创建的文件，可能需要额外的空间。

- `copyright` 文件

  定义安装软件包时所显示的版权信息文本。

每个软件包信息文件都应在 `prototype` 文件中有一个对应条目。关于创建这些文件的更多信息，请参见第 46 页中的 “创建信息文件”。

软件包安装脚本

安装脚本并不是必需的。但是，您可以提供可在安装软件包期间执行自定义操作的脚本。安装脚本具有以下特征：

- 该脚本由 Bourne shell 命令组成。
- 该脚本的文件权限应设为 0644。
- 该脚本不需要包含 shell 标识符 (`#! /bin/sh`)。

四种脚本类型如下：

- `request` 脚本

  `request` 脚本请求来自要安装软件包的管理员的输入。

- `checkinstall` 脚本

  `checkinstall` 脚本执行特殊的文件系统验证。

  注 – `checkinstall` 脚本仅在 Solaris™ 2.5 发行版和兼容发行版中可用。

- 过程脚本

  过程脚本定义在软件包安装和删除过程中的特定时刻所发生的操作。您可以使用这些预定义的名称创建四个过程脚本： `preinstall`、`postinstall`、`preremove` 和 `postremove`。

- 类操作脚本

  类操作脚本定义一组要对对象组执行的操作。

有关安装脚本的更多信息，请参见第 52 页中的 “创建安装脚本”。

生成软件包之前的注意事项

生成软件包之前，您需要确定您的产品将由一个还是多个软件包组成。请注意，许多小软件包会比一个大软件包需要更长的安装时间。虽然创建单个软件包是一个不错的主意，但不一定总能实现。如果您决定生成多个软件包，则需要确定如何对应用程序代码分段。本节提供了计划生成软件包时使用的准则列表。

许多打包准则在彼此之间都存在着一定的折衷。均衡地满足所有要求通常很困难。这些准则是按重要性顺序呈现的。不过，需要根据具体环境将此顺序用作灵活的指导。虽然每个准则都很重要，但终究取决于您来优化这些要求，以生成一组优质的软件包。

有关更多设计理念，请参见第 6 章，创建软件包的高级技术。

使软件包可远程安装

所有软件包必须**可远程安装**。可远程安装是指要安装软件包的管理员可以尝试将其安装在客户机系统上，而不一定安装到执行 `pkgadd` 命令的根 (`/`) 文件系统。

针对客户机/服务器配置进行优化

布置软件包时，请考虑不同类型的系统软件配置（例如，独立系统和服务器）。良好的打包设计会分离受影响的文件，以优化每个配置类型的安装。例如，应该将根 (`/`) 和 `/usr` 文件系统的内容进行分段，以便可以轻松地支持服务器配置。

根据功能边界打包

软件包应该是一个独立的整体，并且可通过一组功能清楚地标识。例如，一个包含 UFS 的软件包应包含所有 UFS 实用程序，并只限于采用 UFS 二进制代码。

应该根据客户的视角将软件包组织成功能单元。

根据版税边界打包

将因有合同而需要支付版税的代码放到专用软件包或软件包组中。请勿将代码分散放到并非必需的多个软件包中。

根据系统相关性打包

将系统相关的二进制代码保存在专用软件包中。例如，应该将内核代码放在专用软件包中，使每个实现体系结构都由不同的软件包实例组成。本规则也适用于不同体系结构的二进制代码。例如，SPARC 系统的二进制代码应放在一个软件包中，而 x86 系统的二进制代码应放在另一个软件包中。

消除软件包中的重叠

构建软件包时，请尽一切可能消除重复的文件。不必要的文件重复会导致支持和版本方面的困难。如果您的产品有多个软件包，请反复比较这些软件包的内容以查看是否存在重复的文件。

根据本地化边界打包

本地化特定的项目应放在其自己的软件包中。理想的软件包模型应根据每种语言环境将产品的本地化项目作为一个软件包交付。遗憾的是，在某些情况下组织边界与功能和产品边界准则相冲突。

国际缺省设置也可以在一个软件包中交付。此设计可以将需要本地化更改的文件隔离开来，并对本地化软件包的交付格式进行标准化。

映像包管理系统 (Image Packaging System, IPS) 软件包

本文档讨论 SVR4 软件包。如果要交付到 OpenSolaris OS，请考虑使用映像包管理系统 (Image Packaging System, IPS) 软件包。OpenSolaris OS 既支持 SVR4，又支持 IPS 软件包。IPS 软件可与网络系统信息库进行交互，并使用 ZFS 文件系统。在 OpenSolaris OS 中，可以使用 `pkgsend(1)` 命令将现有的 SVR4 软件包发布到 IPS 系统信息库。

下表对 SVR4 包管理系统和 IPS 包管理系统所使用的命令进行了比较。有关 IPS 的详细信息，请参见 Getting Started With the Image Packaging System (`http://dlc.sun.com/osol/docs/content/IPS/ggcph.html`)（映像包管理系统入门）。

表 1–1 打包任务：IPS 和 SVR4

任务	IPS 命令	SVR4 命令
安装新软件包	`pkg install`	`pkgadd -a`
显示有关软件包状态的信息	`pkg list`	`pkginfo`
验证软件包安装是否正确	`pkg verify`	`pkgchk -v`
显示有关软件包的信息	`pkg info`	`pkginfo -l`
列出软件包的内容	`pkg contents`	`pkgchk -l`
卸载软件包	`pkg uninstall`	`pkgrm`

打包命令、文件和脚本

本节介绍在处理软件包时可能会用到的命令、文件和脚本。手册页中会对这些内容进行介绍，本书也将对这些内容进行详细介绍，以及其执行的特定任务。

下表显示的命令可帮助您生成、验证、安装软件包并获取有关软件包的信息。

表 1–2 打包命令

任务	命令/手册页	说明	更多信息
创建软件包	pkgproto(1)	生成 prototype 文件以作为 pkgmk 命令的输入	第 33 页中的 “示例－使用 pkgproto 命令创建 prototype 文件”
	pkgmk(1)	创建可安装的软件包	第 40 页中的 “生成软件包”
安装、删除和转换软件包	pkgadd(1M)	将软件包安装到系统中	第 78 页中的 “安装软件包”
	pkgask(1M)	将答复存储到 request 脚本	第 56 页中的 “request 脚本的设计规则”
	pkgtrans(1)	将软件包复制到分发介质	第 87 页中的 “将软件包转换为分发介质”
	pkgrm(1M)	从系统中删除软件包	第 86 页中的 “删除软件包”
获取有关软件包的信息	pkgchk(1M)	验证软件包的完整性	第 80 页中的 “验证软件包的完整性”
	pkginfo(1)	显示软件包信息	第 83 页中的 “pkginfo 命令”
	pkgparam(1)	显示软件包参数值	第 81 页中的 “pkgparam 命令”
修改已安装的软件包	installf(1M)	将新软件包对象并入到已安装的软件包中	第 61 页中的 “过程脚本的设计规则”和第 5 章，软件包创建案例研究
	removef(1M)	从已安装的软件包中删除软件包对象	第 61 页中的 “过程脚本的设计规则”

下表显示的信息文件可帮助您生成软件包。

表 1–3 软件包信息文件

文件	说明	更多信息
admin(4)	软件包安装缺省文件	第 115 页中的 “缺省管理文件”
compver(4)	软件包兼容性文件	第 46 页中的 “定义软件包相关性”
copyright(4)	软件包版权信息文件	第 49 页中的 “编写版权信息”

表 1–3 软件包信息文件 （续）

文件	说明	更多信息
depend(4)	软件包相关性文件	第 46 页中的 “定义软件包相关性”
pkginfo(4)	软件包特征文件	第 23 页中的 “创建 pkginfo 文件”
pkgmap(4)	软件包内容说明文件	第 40 页中的 “pkgmap 文件”
prototype(4)	软件包信息文件	第 28 页中的 “创建 prototype 文件”
space(4)	软件包磁盘空间要求文件	第 50 页中的 “在目标系统上保留额外空间”

下表介绍了一些可选安装脚本，您可以编写这样的脚本以影响是否以及如何安装软件包。

表 1–4 软件包安装脚本

脚本	说明	更多信息
request	请求来自安装人员的信息	第 56 页中的 “编写 request 脚本”
checkinstall	收集文件系统数据	第 58 页中的 “使用 checkinstall 脚本收集文件系统数据”
preinstall	在安装类之前执行任何自定义安装要求	第 60 页中的 “编写过程脚本”
postinstall	在安装所有卷之后执行任何自定义安装要求	第 60 页中的 “编写过程脚本”
preremove	在删除类之前执行任何自定义删除要求	第 60 页中的 “编写过程脚本”
postremove	在删除所有类之后执行任何自定义删除要求	第 60 页中的 “编写过程脚本”
类操作	对特定对象组执行一系列操作。	第 62 页中的 “编写类操作脚本”

◆ ◆ ◆ 第 2 章

2 生成软件包

本章介绍生成软件包涉及的过程和任务。其中一些任务是必需的，一些任务是可选的。必需的任务会在本章中进行详细讨论。有关可选任务（可用于为软件包添加更多功能）的信息，请参见第 3 章，增强软件包的功能（任务）和第 6 章，创建软件包的高级技术。

以下是本章中信息的列表。

软件包生成过程（任务图）

表 2–1 描述了生成软件包时所需遵循的过程，特别是如果您没有丰富的软件包生成经验时更应该遵循该过程。虽然您不必完全按照前四个任务的列出顺序来完成这些任务，但是如果遵循此顺序，您将可以更轻松地体验软件包的生成过程。一旦您成为经验丰富的软件包设计员，就可以根据您的需要重排这些任务的顺序。

作为一位经验丰富的软件包设计员，您可以使用 make 命令和 makefile 来自动化软件包生成过程。有关更多信息，请参见 make(1S) 手册页。

表 2–1 软件包生成过程任务图

任务	说明	参考
1. 创建一个 pkginfo 文件	创建 pkginfo 文件以描述软件包的特征。	第 26 页中的 “如何创建 pkginfo 文件”

表 2–1 软件包生成过程任务图 （续）

任务	说明	参考
2. 组织软件包内容	将软件包组件安排为分层目录结构。	第 27 页中的 “组织软件包的内容”
3. （可选）创建信息文件	定义软件包相关性，提供版权信息，并在目标系统上保留额外空间。	第 3 章，增强软件包的功能（任务）
4. （可选）创建安装脚本	定制软件包的安装和删除过程。	第 3 章，增强软件包的功能（任务）
5. 创建一个 prototype 文件	在 prototype 文件中描述软件包中的对象。	第 28 页中的 “创建 prototype 文件”
6. 生成软件包	使用 pkgmk 命令生成软件包。	第 40 页中的 “生成软件包”
7. 验证并转换软件包	在将软件包复制到分发介质之前验证其完整性。	第 4 章，验证和转换软件包

软件包环境变量

您可以使用必需的信息文件 pkginfo 和 prototype 中的变量。也可以使用 pkgmk 命令的选项，该命令用于生成软件包。随着本章对这些文件和命令的讨论，将会提供更多有关这些变量的上下文相关信息。不过，在开始生成软件包之前，您应该了解不同类型的变量，以及它们如何影响软件包的成功创建。

变量可以分为两种类型：

- 生成变量

  生成变量以小写字母开头，并在**生成时**（当使用 pkgmk 命令生成软件包时）进行计算。

- 安装变量

  安装变量以大写字母开头，并在**安装时**（当使用 pkgadd 命令安装软件包时）进行计算。

关于使用环境变量的一般规则

在 pkginfo 文件中，变量定义采用 *PARAM=value* 形式，其中，*PARAM* 的首字母是大写字母。仅在安装时对这些变量进行计算。如果这些变量中的任一个不能被计算，则 pkgadd 命令将出错而异常中止。

在 prototype 文件中，变量定义可采用 !*PARAM=value* 或 $ *variable* 形式。*PARAM* 和 *variable* 均可以一个大写或小写字母开头。只有在生成时值已知的变量才会被计算。如果 *PARAM* 或 *variable* 是在生成时值未知的生成变量或安装变量，则 pkgmk 命令将出错而异常中止。

您还可以将 *PARAM=value* 选项用作 pkgmk 命令的选项。此选项的作用与在 prototype 文件中基本相同，只不过其作用域是全局的，适用于整个软件包。 prototype 文件中的 !*PARAM=value* 定义是局部的，仅适用于该文件及其所定义的软件包组成部分。

如果 *PARAM* 是安装变量，而 *variable* 是值已知的安装变量或生成变量，则 pkgmk 命令会将定义插入到 pkginfo 文件中，以便该定义在安装时可用。但是，pkgmk 命令不会计算在 prototype 文件中指定的任何路径名中的 *PARAM* 变量。

软件包环境变量汇总

下表汇总了变量的规范格式、位置和作用域。

表 2–2 软件包环境变量汇总

变量的定义位置	变量定义格式	定义的变量类型	变量计算时间	变量计算位置	变量可以替换的项目
pkginfo 文件	*PARAM=value*	生成	在生成时忽略	N/A	无
		安装	安装时	在 pkgmap 文件中	*owner*、*group*、*path* 或链接目标
prototype 文件	!*PARAM=value*	生成	生成时	在 prototype 文件和任何内含的文件中	*mode*、*owner*、*group* 或 *path*
		安装	生成时	在 prototype 文件和任何内含的文件中	仅 !search 和 !command 命令
pkgmk 命令行	*PARAM=value*	生成	生成时	在 prototype 文件中	*mode*、*owner*、*group* 或 *path*
		安装	生成时	在 prototype 文件中	仅 !search 命令
			安装时	在 pkgmap 文件中	*owner*、*group*、*path* 或链接目标

创建 pkginfo 文件

pkginfo 文件是一种 ASCII 文件，用于描述软件包的特征以及帮助控制安装流程的信息。

pkginfo 文件中的每个条目均占一行，使用 *PARAM=value* 格式设定参数值。*PARAM* 可以是在 pkginfo(4) 手册页中描述的任一个标准参数。对于参数的指定顺序没有确定要求。

注 – 每个 *value* 可以由单引号或双引号括起（例如，'*value*' 或 "*value*"）。如果 *value* 包含任何对 shell 环境而言被视为特殊字符的字符，您应该使用引号。本书中的示例和案例分析不使用引号。请参见 pkginfo(4) 手册页，了解使用双引号的示例。

您还可以通过在 pkginfo 文件中为软件包参数赋值，创建您自己的软件包参数。您的参数必须以大写字母开头，后跟大写或小写字母。大写字母表明参数（变量）将在安装时（与生成时相对）被计算。有关安装变量与生成变量之间区别的信息，请参见第 22 页中的 “软件包环境变量”。

注 – 任何参数值之后的结尾空格都会被忽略。

您必须在 pkginfo 文件中定义以下五个参数：PKG、NAME、ARCH、VERSION 和 CATEGORY。当生成软件包时，软件会自动插入 PATH、PKGINST 和 INSTDATE 参数。不要修改这八个参数。有关其余参数的信息，请参见 pkginfo(4) 手册页。

定义软件包实例

同一个软件包可以有不同的版本，可以与不同的体系结构兼容，或者同时符合这两种情况。软件包的每个变体称为一个**软件包实例**。软件包实例通过组合 pkginfo 文件中的 PKG、ARCH 和 VERSION 参数定义来确定。

pkgadd 命令可在安装时为每个软件包实例指定一个**软件包标识符**。软件包标识符是软件包缩写后跟一个数字后缀，例如 SUNWadm.2。此标识符可区分不同软件包的实例，还可区分同一个软件包的不同实例。

定义软件包缩写 (PKG)

软件包缩写是软件包的简短名称，由 pkginfo 文件中的 PKG 参数定义。软件包缩写必须具有以下特征：

- 缩写必须由字母数字字符组成。第一个字符不能是数字。
- 缩写长度不能超过 32 个字符。
- 缩写不能是以下保留的缩写之一：install、new 或 all。

注 – 前四个字符对您公司来说应该是唯一的。例如，由 Sun Microsystems™ 生成的所有软件包均以 "SUNW" 作为其软件包缩写的前四个字符。

pkginfo 文件中软件包缩写条目的示例为 PKG=SUNWcadap。

指定软件包体系结构 (ARCH)

pkginfo 文件中的 ARCH 参数标识与软件包相关联的体系结构。体系结构名称的最大长度为 16 个字母数字字符。如果一个软件包与多个体系结构相关联，请以逗号分隔的列表形式指定这些体系结构。

以下是 pkginfo 文件中一个软件包体系结构规范的示例：

```
ARCH=sparc
```

指定软件包指令集体系结构 (SUNW_ISA)

pkginfo 文件中的 SUNW_ISA 参数标识与 Sun Microsystems 软件包相关联的指令集体系结构。值如下所示：

- sparcv9，表示包含 64 位对象的软件包
- sparc，表示包含 32 位对象的软件包

例如，在 pkginfo 文件中，一个包含 64 位对象的软件包的 SUNW_ISA 值是：

```
SUNW_ISA=sparcv9
```

如果没有设置 SUNW_ISA，则软件包的缺省指令集体系结构会设置为 ARCH 参数的值。

指定软件包版本 (VERSION)

pkginfo 文件中的 VERSION 参数标识软件包的版本。版本的最大长度为 256 个 ASCII 字符，且不能以左括号开头。

以下是 pkginfo 文件中的一个版本规范示例：

```
VERSION=release 1.0
```

定义软件包名称 (NAME)

软件包名称是软件包的全名，由 pkginfo 文件中的 NAME 参数定义。

由于系统管理员通常会根据软件包名称确定是否需要安装该软件包，因此使用清晰、简明且完整的软件包名称是很重要的。软件包名称必须满足以下条件：

- 指明什么时候需要软件包（例如，用于提供某些命令或功能，或者指明该软件包是否为特定硬件所需要）。
- 指明软件包的用途（例如，用于设备驱动程序的开发）。
- 包括软件包缩写助记符的说明，使用的关键字应该表明缩写是说明的简短形式。例如，软件包缩写 SUNWbnuu 对应的软件包名称是“Basic Networking UUCP Utilities, (Usr)”。

- 指定安装软件包的分区名称。
- 使用的术语符合其行业含义。
- 遵守 256 字符数限制。

以下是 pkginfo 文件中定义的软件包名称示例：

```
NAME=Chip designers need CAD application software to design
abc chips.  Runs only on xyz hardware and is installed in the
usr partition.
```

定义软件包类别 (CATEGORY)

pkginfo 文件中的 CATEGORY 参数指定软件包所属的类别。一个软件包至少必须属于 system 或 application 类别。类别名称由字母数字字符组成。类别名称的最大长度为 16 个字符，且不区分大小写。

如果一个软件包属于多个类别，请以逗号分隔的列表形式指定这些类别。

以下是 pkginfo 文件中的 CATEGORY 规范示例：

```
CATEGORY=system
```

▼ 如何创建 pkginfo 文件

1. **使用您喜爱的文本编辑器，创建一个名为 pkginfo 的文件。**
   可在系统上的任意位置创建此文件。

2. **编辑该文件并定义五个必需的参数。**
   这五个必需的参数是： PKG、NAME、 ARCH、VERSION 和 CATEGORY。有关这些参数的更多信息，请参见第 23 页中的 “创建 pkginfo 文件”。

3. **向文件中添加任何可选参数。**
   创建您自己的参数，或参见 pkginfo(4) 手册页了解有关标准参数的信息。

4. **保存所做更改，然后退出编辑器。**

示例 2–1 创建 pkginfo 文件

此示例显示了一个有效 pkginfo 文件的内容，其中定义了五个必需的参数以及 BASEDIR 参数。将会在第 30 页中的 “*path* 字段”中更详细地讨论 BASEDIR 参数。

```
PKG=SUNWcadap
NAME=Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.
ARCH=sparc
VERSION=release 1.0
CATEGORY=system
BASEDIR=/opt
```

另请参见 请参见第 27 页中的“如何组织软件包的内容”。

组织软件包的内容

将软件包对象以分层目录结构进行组织，该结构会模仿软件包对象安装到目标系统上之后的结构。如果您在创建 `prototype` 文件之前执行此步骤，可节省创建该文件的时间和工作量。

▼ 如何组织软件包的内容

1. **确定您需要创建的软件包数目，以及哪些软件包对象应位于每个软件包中。**

   要获得完成此步骤的帮助，请参见第 16 页中的“生成软件包之前的注意事项”。

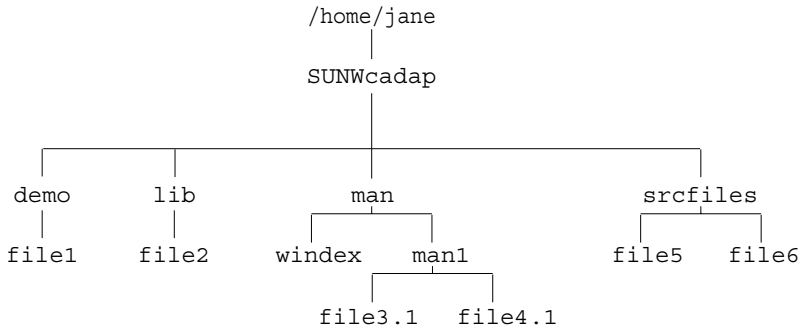
2. **为每个需要生成的软件包创建一个目录。**

   可在系统上的任意位置创建该目录，并随意进行命名。本章中的示例假定软件包目录的名称与软件包缩写相同。

   ```
   $ cd /home/jane
   $ mkdir SUNWcadap
   ```

3. **将每个软件包中的软件包对象组织为相应软件包目录下的目录结构。该目录结构必须模仿软件包对象在目标系统上的结构。**

   例如，CAD 应用程序软件包 `SUNWcadap` 必须具有以下目录结构。



4 **确定您将保存信息文件的位置。如果合适，请创建一个目录，将这些文件保存在同一位置。**

以下示例假定来自第 26 页中的 “如何创建 pkginfo 文件”中的 pkginfo 示例文件在 Jane 的主目录中创建。

```
$ cd /home/jane
$ mkdir InfoFiles
$ mv pkginfo InfoFiles
```

另请参见 请参见第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”。

创建 prototype 文件

prototype 文件是一种 ASCII 文件，用于指定有关软件包中对象的信息。prototype 文件中的每个条目描述一个对象，例如数据文件、目录、源文件或可执行对象。prototype 文件中的条目由若干个用空格分隔的信息字段组成。请注意，这些字段**必须**按特定顺序出现。注释行以井号 (#) 开头，将被忽略。

您可以使用文本编辑器或使用 pkgproto 命令创建 prototype 文件。首次创建此文件时，使用 pkgproto 命令可能更容易，因为该命令基于您先前创建的目录分层结构来创建文件。如果您没有根据第 27 页中的 “组织软件包的内容”中所述组织文件，您不得不使用喜爱的文本编辑器从头开始创建 prototype 文件，这一任务比较繁琐。

prototype 文件的格式

以下是 prototype 文件中每一行的格式：

partftypeclasspathmajorminormodeownergroup

part	可选的数字字段，可用于将软件包对象分组为各部分。缺省值为 part 1（第 1 部分）。
ftype	单字符字段，用于指定对象类型。请参见第 29 页中的 “ftype 字段”。
class	表示对象所属的安装类。请参见第 30 页中的 “*class* 字段”。
path	绝对或相对路径名，指示软件包对象在目标系统上的驻留位置。请参见第 30 页中的 “*path* 字段”。
major	块特殊设备或字符特殊设备的主设备编号。
minor	块特殊设备或字符特殊设备的次要设备编号。
mode	对象的八进制模式（例如，0644）。请参见第 32 页中的 “*mode* 字段”。
owner	对象的所有者（例如，bin 或 root）。请参见第 33 页中的 “*owner* 字段”。
group	对象所属的组（例如 bin 或 sys）。请参见第 33 页中的 “*group* 字段”。

通常，只会定义 *ftype*、*class*、*path*、*mode*、*owner* 和 *group* 字段。以下各节将介绍这些字段。有关这些字段的附加信息，请参见 prototype(4) 手册页。

ftype 字段

ftype（文件类型）字段是一个单字符字段，用于指定软件包对象的文件类型。下表中介绍了有效的文件类型。

表 2–3 prototype 文件中的有效文件类型

文件类型字段值	文件类型说明
f	标准可执行文件或数据文件
e	在安装或删除时将编辑的文件（可由几个软件包共享）
v	可变文件（其内容预计会更改，例如日志文件）
d	目录
x	只能由该软件包访问的专用目录（可能包含未注册的日志或数据库信息）
l	链接文件
p	命名管道
c	字符特殊设备

表 2–3 prototype 文件中的有效文件类型 （续）

文件类型字段值	文件类型说明
b	块特殊设备
i	信息文件或安装脚本
s	符号链接

class 字段

class 字段用于指定对象所属的类。使用类是一种可选的软件包设计功能。将会在第 62 页中的 “编写类操作脚本”中详细讨论此功能。

如果您不使用类，则对象属于 none 类。当执行 pkgmk 命令生成软件包时，该命令会在 pkginfo 文件中插入 CLASSES=none 参数。文件类型为 i 的文件必须有一个空的 *class* 字段。

path 字段

path 字段用于定义软件包对象在目标系统上的驻留位置。您可以使用绝对路径名（例如 /usr/bin/mail）或相对路径名（例如 bin/mail）来指示该位置。使用绝对路径名意味着对象在目标系统上的位置是由软件包定义的，不能更改。使用相对路径名的软件包对象表明该对象是**可重定位的**。

可重定位对象在目标系统上不需要绝对路径位置。此类对象的位置是在安装过程中确定的。

可将所有软件包对象或某些软件包对象定义为可重定位。在编写任何安装脚本或创建 prototype 文件之前，请确定软件包对象将有固定位置（例如 /etc 中的启动脚本）还是可重定位的。

可重定位对象分为两种：**可共同重定位的对象**和**可单独重定位的对象**。

可共同重定位的对象

可共同重定位的对象相对于一个称作**基目录**的通用安装基本位置进行定位。基目录在 pkginfo 文件中由 BASEDIR 参数定义。例如， prototype 文件中一个名为 tests/generic 的可重定位对象要求 pkginfo 文件定义缺省 BASEDIR 参数。例如：

```
BASEDIR=/opt
```

此示例意味着，对象安装后将位于 /opt/tests/generic 目录中。

注 – /opt 目录是不属于基本 Solaris 软件的软件可以交付到的唯一目录。

只要可能，请尽量使用可共同重定位的对象。一般来说，可以使用指定了绝对路径的几个文件（例如 /etc 或 /var 中的文件）来重定位一个软件包的主要部分。但是，如果一个软件包包含许多不同的重定位对象，请考虑将该软件包分成在 pkginfo 文件中具有明显不同 BASEDIR 值的多个软件包。

可单独重定位的对象

可单独重定位的对象不限于定位到可共同重定位的对象所在的目录位置。要定义一个可单独重定位的对象，需要在 prototype 文件的 *path* 字段中指定一个安装变量。指定安装变量之后，创建一个 request 脚本来提示安装人员输入可重定位基目录，或者创建一个 checkinstall 脚本基于文件系统数据确定路径名。有关 request 脚本的更多信息，请参见第 56 页中的 “编写 request 脚本”；有关 checkinstall 脚本的信息，请参见第 59 页中的 “如何收集文件系统数据”。

注意 – 可单独重定位的对象很难管理。使用可单独重定位的对象可能导致软件包组件的位置四处分散，这样在安装软件包的多个版本或体系结构时很难分离这些组件。只要可能，请尽量使用可共同重定位的对象。

参数化路径名

参数化路径名是包含变量规范的路径名。例如，/opt/$PKGINST/*filename* 是一个参数化路径名，因为含有 $PKGINST 变量规范。**必须**在 pkginfo 文件中为该变量规范定义一个缺省值。然后可以通过 request 脚本或 checkinstall 脚本更改该值。

路径中的变量规范必须位于路径名的开头或结尾，或者由斜线 (/) 分隔。有效的参数化路径名采用以下形式：

```
$PARAM/tests
tests/$PARAM/generic
/tests/$PARAM
```

一旦定义了变量规范，可能会导致将路径计算为绝对或可重定位路径。在下面的示例中，prototype 文件包含以下条目：

```
f none $DIRLOC/tests/generic
```

pkginfo 文件包含以下条目：

```
DIRLOC=/myopt
```

路径名 $DIRLOC/tests/generic 计算为绝对路径名 /myopt/tests/generic（无论是否在 pkginfo 文件中设置了 BASEDIR 参数）。

在此示例中，prototype 文件与前面的示例中完全相同，而 pkginfo 文件包含以下条目：

```
DIRLOC=firstcut
BASEDIR=/opt
```

路径名 $DIRLOC/tests/generic 将计算为可重定位的路径名 /opt/firstcut/tests/generic。

有关参数化路径名的更多信息，请参见第 117 页中的 “使用参数化基目录”。

关于对象的源位置和目标位置的简要说明

prototype 文件中的 path 字段定义对象在目标系统上的位置。如果软件包对象的目录结构不模仿目标系统上的预期结构，请在 prototype 文件中指定其当前位置。有关设置软件包对象结构的更多信息，请参见第 27 页中的 “组织软件包的内容”。

如果您开发区域没有将软件包设置为所需的结构，可在 *path* 字段中使用 *path1=path2* 格式。在此格式中，*path1* 是对象应该在目标系统上所处的位置，而 *path2* 是对象在您系统中的位置。

还可以在 *path1=path2* 路径名格式中以 *path1* 表示可重定位对象的名称，而 *path2* 表示该对象在您系统上的完整路径名。

注 – *path1* 不能包含未定义的生成变量，但是可以包含未定义的安装变量。*path2* 不能包含任何未定义的变量，虽然可以使用生成变量和安装变量。有关安装变量与生成变量之间区别的信息，请参见第 22 页中的 “软件包环境变量”。

因为链接是通过 *pkgadd* 命令创建的，因此它们必须使用 path1= path2 格式。通常，链接的 *path2* 决不应是绝对的，而是应该相对于 *path1* 的目录部分。

使用 *path1=path2* 格式的一个替代方法是使用 !search 命令。有关更多信息，请参见第 37 页中的 “为 pkgmk 命令提供搜索路径”。

mode 字段

mode 字段可以包含八进制数、问号 (?) 或变量规范。八进制数指定对象安装在目标系统上时的模式。? 表示在安装对象时模式不会更改，暗指在目标系统中已经存在同名对象。

$*mode* 形式的变量规范（其中变量的首字母必须是小写字母）意味着将会在生成软件包时设置此字段。请注意，此变量必须在生成时在 prototype 文件中或作为 pkgmk 命令的一个选项定义。有关安装变量与生成变量之间区别的信息，请参见第 22 页中的 “软件包环境变量”。

文件类型为 i（信息文件）、l（硬链接）和 s（符号链接）的文件应将该字段留空。

owner 字段

owner 字段可以包含用户名、问号 (?) 或变量规范。用户名最多可包含 14 个字符，并且应该是目标系统中已经存在的名称（例如 bin 或 root）。? 表示在安装对象时所有者不会更改，暗指在目标系统中已经存在同名对象。

变量规范的形式可以是 $*Owner* 或 $*owner*，其中变量的首字母是大写字母或小写字母。如果变量首字母为一个小写字母，则必须在生成软件包时在 prototype 文件中或作为 pkgmk 命令的一个选项定义该变量。如果变量首字母为大写字母，系统会将变量规范插入 pkginfo 文件中作为缺省值，并且可以在安装时通过 request 脚本重新定义。有关安装变量与生成变量之间区别的信息，请参见第 22 页中的 “软件包环境变量”。

文件类型为 i（信息文件）和 l（硬链接）的文件应将该字段留空。

group 字段

group 字段可以包含组名、问号 (?) 或变量规范。组名最多可包含 14 个字符，并且应该是目标系统中已经存在的名称（例如 bin 或 sys）。? 表示在安装对象时组不会更改，暗指在目标系统中已经存在同名对象。

变量规范的形式可以是 $*Group* 或 $*group*，其中变量的首字母是大写字母或小写字母。如果变量首字母为一个小写字母，则必须在生成软件包时在 prototype 文件中或作为 pkgmk 命令的一个选项定义该变量。如果变量首字母为大写字母，系统会将变量规范插入 pkginfo 文件中作为缺省值，并且可以在安装时通过 request 脚本重新定义。有关安装变量与生成变量之间区别的信息，请参见第 22 页中的 “软件包环境变量”。

文件类型为 i（信息文件）和 l（硬链接）的文件应将该字段留空。

从头创建 prototype 文件

如果您想要从头创建 prototype 文件，可以使用喜爱的文本编辑器创建，为每个软件包对象添加一个条目。有关该文件格式的更多信息，请参见第 28 页中的 “prototype 文件的格式”和 prototype(4) 手册页。不过，在定义了每个软件包对象之后，您可能希望包括第 36 页中的 “向 prototype 文件添加功能”中描述的某些功能。

示例－使用 pkgproto 命令创建 prototype 文件

只要您按照第 27 页中的 “组织软件包的内容”中所述组织了您的软件包目录结构，就可以使用 pkgproto 命令生成基本 prototype 文件。例如，使用在前面各节中所述的样例目录结构和 pkginfo 文件时，创建 prototype 文件的命令如下所示：

```
$ cd /home/jane
$ pkgproto ./SUNWcadap > InfoFiles/prototype
```

prototype 文件如下所示：

```
d none SUNWcadap 0755 jane staff
d none SUNWcadap/demo 0755 jane staff
f none SUNWcadap/demo/file1 0555 jane staff
d none SUNWcadap/srcfiles 0755 jane staff
f none SUNWcadap/srcfiles/file5 0555 jane staff
f none SUNWcadap/srcfiles/file6 0555 jane staff
d none SUNWcadap/lib 0755 jane staff
f none SUNWcadap/lib/file2 0644 jane staff
d none SUNWcadap/man 0755 jane staff
f none SUNWcadap/man/windex 0644 jane staff
d none SUNWcadap/man/man1 0755 jane staff
f none SUNWcadap/man/man1/file4.1 0444 jane staff
f none SUNWcadap/man/man1/file3.1 0444 jane staff
```

注 – 生成软件包的人员的实际所有者和组由 pkgproto 命令记录。一种较好的方法是使用 chown -R 和 chgrp -R 命令，在运行 pkgproto 命令**之前**根据需要设置所有者和组。

此 prototype 示例文件不是完整文件。有关完成该文件的信息，请参见下一节。

优化使用 pkgproto 命令创建的 prototype 文件

虽然 pkgproto 命令在创建初始 prototype 文件时很有用，但它不会为每个需要定义的软件包对象创建条目。此命令不创建全部条目。pkgproto 命令不执行以下任一工作：

- 为文件类型为 v（可变文件）、e（可编辑的文件）、x（专用目录）或 i（信息文件或安装脚本）的对象创建全部条目
- 使用一个调用支持多个类

创建文件类型为 v、e、x 和 i 的对象条目

至少，您需要修改 prototype 文件，以添加文件类型为 i 的对象。如果您将信息文件和安装脚本存储在软件包目录的第一级中（例如 /home/jane/SUNWcadap/pkginfo），则 prototype 文件中的一个条目将如下所示：

```
i pkginfo
```

如果您没有将信息文件和安装脚本存储在软件包目录的第一级中，则需要指定它们的源位置。例如：

```
i pkginfo=/home/jane/InfoFiles/pkginfo
```

或者，可以使用 !search 命令指定位置，供 pkgmk 命令在生成软件包时进行查找。有关更多信息，请参见第 37 页中的 “为 pkgmk 命令提供搜索路径”。

要为文件类型为 v、e 和 x 的对象添加条目，请遵循第 28 页中的 “prototype 文件的格式”中所述的格式，或参阅 prototype(4) 手册页。

注 – 切记在任何时候都为文件类型为 e（可编辑）的文件指定一个类，并为该类指定一个关联的类操作脚本。否则，在删除软件包期间这类文件将被删除，即使与其他软件包共享路径名时也是如此。

使用多个类定义

如果使用 pkgproto 命令创建基本 prototype 文件，可以将所有软件包对象指定给 none 类或一个特定类。如第 33 页中的 “示例－使用 pkgproto 命令创建 prototype 文件”中所示，基本 pkgproto 命令将所有对象指定给 none 类。要将所有对象指定给一个特定类，可使用 -c 选项。例如：

```
$ pkgproto -c classname /home/jane/SUNWcadap > /home/jane/InfoFiles/prototype
```

如果使用多个类，您可能需要手动编辑 prototype 文件，并修改每个对象的 *class* 字段。如果使用类，您还需要在 pkginfo 文件中定义 CLASSES 参数并编写类操作脚本。使用类是可选功能，将在第 62 页中的 “编写类操作脚本”中进行详细讨论。

示例－优化使用 pkgproto 命令创建的 prototype 文件

对于在第 33 页中的 “示例－使用 pkgproto 命令创建 prototype 文件”中使用 pkgproto 命令创建的 prototype 文件，需要进行几处修改。

- 需要有一个 pkginfo 文件的条目。
- 需要将 *path* 字段更改为 *path1=path2* 格式，因为软件包源位置位于 /home/jane。由于软件包源位置是一个分层目录，并且 !search 命令不递归搜索，因此使用 *path1=path2* 格式也许会更轻松。
- *owner* 和 *group* 字段应该包含目标系统上现有用户和组的名称。也就是说，所有者 jane 将会导致错误，因为该所有者不属于 SunOS™ 操作系统。

  修改后的 prototype 文件如下所示：

```
i pkginfo=/home/jane/InfoFiles/pkginfo
d none SUNWcadap=/home/jane/SUNWcadap 0755 root sys
d none SUNWcadap/demo=/home/jane/SUNWcadap/demo 0755 root bin
f none SUNWcadap/demo/file1=/home/jane/SUNWcadap/demo/file1 0555 root bin
d none SUNWcadap/srcfiles=/home/jane/SUNWcadap/srcfiles 0755 root bin
f none SUNWcadap/srcfiles/file5=/home/jane/SUNWcadap/srcfiles/file5 0555 root bin
f none SUNWcadap/srcfiles/file6=/home/jane/SUNWcadap/srcfiles/file6 0555 root bin
d none SUNWcadap/lib=/home/jane/SUNWcadap/lib 0755 root bin
f none SUNWcadap/lib/file2=/home/jane/SUNWcadap/lib/file2 0644 root bin
d none SUNWcadap/man=/home/jane/SUNWcadap/man 0755 bin bin
f none SUNWcadap/man/windex=/home/jane/SUNWcadap/man/windex 0644 root other
d none SUNWcadap/man/man1=/home/jane/SUNWcadap/man/man1 0755 bin bin
f none SUNWcadap/man/man1/file4.1=/home/jane/SUNWcadap/man/man1/file4.1 0444 bin bin
f none SUNWcadap/man/man1/file3.1=/home/jane/SUNWcadap/man/man1/file3.1 0444 bin bin
```

向 prototype 文件添加功能

除了在 prototype 文件中定义每个软件包对象外，您还可以执行以下操作：

- 定义要在安装时创建的其他对象。
- 在安装时创建链接。
- 将软件包分发在多个卷上。
- 嵌套 prototype 文件。
- 为 *mode*、*owner* 和 *group* 字段设置缺省值。
- 为 pkgmk 命令提供搜索路径。
- 设置环境变量。

有关进行这些更改的信息，请参见以下各节。

定义要在安装时创建的其他对象

可以使用 prototype 文件定义在安装介质上实际没有提供的对象。在安装期间，如果这些对象尚不存在，则使用 pkgadd 命令创建的对象将具有所需的文件类型。

要指定将在目标系统上创建对象，请在 prototype 文件中使用适当文件类型为其添加一个条目。

例如，如果您要在目标系统上创建目录，但是不想在安装介质上提供该目录，请在 prototype 文件中为其添加以下条目：

```
d none /directory 0644 root other
```

如果您希望在目标系统上创建一个空文件，prototype 文件中该文件的条目可能如下所示：

```
f none filename=/dev/null 0644 bin bin
```

必须在安装介质上提供的对象只有常规文件和编辑脚本（文件类型 e、v、f）以及包含这些内容所需的目录。所有其他对象的创建都与提供的对象、目录、命名管道、设备、硬链接和符号链接无关。

在安装时创建链接

要在软件包安装期间创建链接，请在链接对象的 prototype 文件条目中定义以下内容：

- 其文件类型：l（链接）或 s（符号链接）。
- 链接对象的 *path1=path2* 格式的路径名，其中 *path1* 是目标文件，*path2* 是源文件。通常，链接的 *path2* 决不应是绝对的，而是应该相对于 *path1* 的目录部分。例如，定义符号链接的 prototype 文件条目可能如下所示：

  ```
  s none etc/mount=../usr/etc/mount
  ```

无论软件包是安装为绝对还是可重定位软件包，都将以这种方式指定相对链接。

将软件包分发到多个卷上

使用 pkgmk 命令生成软件包时，该命令将执行组织多卷软件包所需的计算和操作。一个多卷软件包称为**分段的软件包**。

不过，您可以在 prototype 文件中使用可选 *part* 字段定义希望对象所处的部分。此字段中的一个数字将重写 pkgmk 命令，并强制将组件放置到该字段中指定的部分。请注意，对于设置为文件系统格式的可移除介质，各部分和卷之间存在一对一的对应关系。如果卷由开发者预先指定，当任何卷上存在空间不足情况时，pkgmk 命令会引发错误。

嵌套 prototype 文件

可以创建多个 prototype 文件，然后使用 !include 命令将它们包括在 prototype 文件中。您可能需要嵌套文件以更便于维护。

在以下示例中有三个 prototype 文件。主文件 (prototype) 正被编辑。另外两个文件（proto2 和 proto3）是要包括的文件。

```
!include /source-dir/proto2
!include /source-dir/proto3
```

为 *mode*、*owner* 和 *group* 字段设置缺省值

要为特定软件包对象的 *mode*、*owner* 和 *group* 字段设置缺省值，可将 !default 命令插入到 prototype 文件中。例如：

```
!default 0644 root other
```

注 – !default 命令的作用域从插入位置开始直到文件结束。该命令的作用域不包括内含的文件。

但是，对于您知道在目标系统（例如 /usr 或 /etc/vfstab）上存在的目录（文件类型 d）和可编辑文件（文件类型 e），请确保 *prototype* 文件中的 *mode*、*owner* 和 group 字段设置为问号 (?)。这样就不会销毁站点管理员可能已修改的现有设置。

为 pkgmk 命令提供搜索路径

如果软件包对象的源位置与其目标位置不同，并且您不希望使用第 32 页中的 “关于对象的源位置和目标位置的简要说明”中所述的 *path1=path2* 格式，则可以在 prototype 文件中使用 !search 命令。

例如，如果您在自己的主目录中创建了目录 pkgfiles，并且该目录包含您的所有信息文件和安装脚本，可以指定在使用 pkgmk 命令生成软件包时搜索该目录。

在 prototype 文件中该命令将如下所示：

```
!search /home-dir/pkgfiles
```

注 – 搜索请求不涉及内含的文件。另外，搜索仅限于列出的特定目录，而且不进行递归搜索。

设置环境变量

也可以向 prototype 文件中添加 !*PARAM=value* 形式的命令。此形式的命令可在当前环境中定义变量。如果有多个 prototype 文件，则此命令的作用域是局部的，仅限于定义它的 prototype 文件。

PARAM 变量的首字母可以是小写字母或大写字母。如果 *PARAM* 变量的值在生成时未知，pkgmk 命令将出错而异常中止。有关生成变量与安装变量之间区别的更多信息，请参见第 22 页中的 “软件包环境变量”。

▼ 如何使用 pkgproto 命令创建 prototype 文件

注 – 在创建 prototype 文件之前创建信息文件和安装脚本会更容易，但这种顺序并非是必需的。您始终可在更改软件包内容之后编辑 prototype 文件。有关信息文件和安装脚本的更多信息，请参见第 3 章，增强软件包的功能（任务）。

1. **确定哪些软件包对象将是绝对的，哪些软件包对象是可重定位的（如果尚未确定）。**

   有关帮助您完成此步骤的信息，请参见第 30 页中的 “*path* 字段”。

2. **组织软件包对象，以模仿其在目标系统上的位置。**

   如果您已经按照第 27 页中的 “组织软件包的内容”中所述组织了软件包，请注意，您可能需要根据步骤 1 中所做的决策进行一些更改。如果您尚未组织软件包，现在应该进行组织。如果您不组织软件包，则不能使用 pkgproto 命令创建基本 prototype 文件。

3. **如果您的软件包有可共同重定位的对象，请编辑 pkginfo 文件将 BASEDIR 参数设置为适当的值。**

   例如：

   ```
   BASEDIR=/opt
   ```

   有关可共同重定位对象的信息，请参见第 30 页中的 “可共同重定位的对象”。

4. **如果您的软件包有可单独重定位的对象，请创建 request 脚本，以提示安装人员输入适当的路径名。或者，创建 checkinstall 脚本，以基于文件系统数据确定适当的路径。**

   以下列表提供了关于常见任务的参考资料页码：

   - 要创建 request 脚本，请参见第 57 页中的 “如何编写 request 脚本”。
   - 要创建 checkinstall 脚本，请参见第 59 页中的 “如何收集文件系统数据”。

- 有关可单独重定位对象的更多信息，请参见第 31 页中的 “可单独重定位的对象”。

5 **将您的软件包组件的所有者和组更改为目标系统上所需的所有者和组。**

对软件包目录和信息文件目录使用 chown -R 和 chgrp -R 命令。

6 **执行 pkgproto 命令创建基本 prototype 文件。**

pkgproto 命令会扫描您的目录来创建基本文件。例如：

```
$ cd package-directory
$ pkgproto ./package-directory > prototype
```

prototype 文件可位于您系统上的任意位置。将信息文件和安装脚本保存在一个位置可简化访问和维护。有关 pkgproto 命令的附加信息，请参见 pkgproto(1) 手册页。

7 **使用您喜爱的文本编辑器编辑 prototype 文件，并为类型为 v、e、x 和 i 的文件添加条目。**

有关您可能需要进行的特定更改的信息，请参见第 34 页中的 “优化使用 pkgproto 命令创建的 prototype 文件”。

8 **（可选）如果您使用多个类，请编辑 prototype 和 pkginfo 文件。使用您喜爱的文本编辑器进行必要的更改，并创建相应的类操作脚本。**

有关您可能需要进行的特定更改的信息，请参见第 34 页中的 “优化使用 pkgproto 命令创建的 prototype 文件”和第 62 页中的 “编写类操作脚本”。

9 **使用您喜爱的文本编辑器编辑 prototype 文件，以便重新定义路径名和更改其他字段设置。**

有关更多信息，请参见第 34 页中的 “优化使用 pkgproto 命令创建的 prototype 文件”。

10 **（可选）使用您喜爱的文本编辑器编辑 prototype 文件，以便向 prototype 文件添加功能。**

有关更多信息，请参见第 36 页中的 “向 prototype 文件添加功能”。

11 **保存所做更改，然后退出编辑器。**

另请参见 如果您已准备好执行下一个任务，请参见第 41 页中的 “如何生成软件包”。

生成软件包

可使用 pkgmk 命令生成软件包。pkgmk 命令可执行以下任务：

- 将 prototype 文件中定义的所有对象放入目录格式中。
- 创建 pkgmap 文件，该文件将替换 prototype 文件。
- 生成用作 pkgadd 命令的输入的可安装软件包。

使用最简单的 pkgmk 命令

此命令最简单的形式是不带任何选项的 pkgmk 命令。在使用不带选项的 pkgmk 命令之前，请确保您的当前工作目录包含软件包的 prototype 文件。命令的输出、文件和目录都被写入 /var/spool/pkg 目录。

pkgmap 文件

当使用 pkgmk 命令生成软件包时，它将创建一个替换 prototype 文件的 pkgmap 文件。来自上一个示例中的 pkgmap 文件包含以下内容：

```
$ more pkgmap
: 1 3170
1 d none SUNWcadap 0755 root sys
1 d none SUNWcadap/demo 0755 root bin
1 f none SUNWcadap/demo/file1 0555 root bin 14868 45617 837527496
1 d none SUNWcadap/lib 0755 root bin
1 f none SUNWcadap/lib/file2 0644 root bin 1551792 62372 837527499
1 d none SUNWcadap/man 0755 bin bin
1 d none SUNWcadap/man/man1 0755 bin bin
1 f none SUNWcadap/man/man1/file3.1 0444 bin bin 3700 42989 837527500
1 f none SUNWcadap/man/man1/file4.1 0444 bin bin 1338 44010 837527499
1 f none SUNWcadap/man/windex 0644 root other 157 13275 837527499
1 d none SUNWcadap/srcfiles 0755 root bin
1 f none SUNWcadap/srcfiles/file5 0555 root bin 12208 20280 837527497
1 f none SUNWcadap/srcfiles/file6 0555 root bin 12256 63236 837527497
1 i pkginfo 140 10941 837531104
$
```

此文件格式非常类似于 prototype 文件的格式。但是，pkgmap 文件包含以下信息：

- 第一行指示了软件包所分布的卷的数量，以及软件包安装后的近似大小。

  例如：1 3170 表明软件包分布在一个卷上，在安装后占用大约 3170 个 512 字节块。
- 有三个额外字段定义了每个软件包对象的大小、校验和和修改时间。
- 软件包对象根据类和路径名按照字母顺序列出，从而减少了安装软件包所需的时间。

▼ 如何生成软件包

1 **创建 pkginfo 文件，如果尚未创建。**
有关逐步说明，请参见第 26 页中的 “如何创建 pkginfo 文件”。

2 **创建 prototype 文件，如果尚未创建。**
有关逐步说明，请参见第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”。

3 **将您的当前工作目录设置为包含软件包的 prototype 文件的目录。**

4 **生成软件包。**

```
$ pkgmk [-o] [-a arch] [-b base-src-dir] [-d device]
  [-f filename] [-l limit] [-p pstamp] [-r rootpath]
  [-v version] [PARAM=value] [pkginst]
```

`-o`	覆写软件包的现有版本。
`-a` *arch*	重写 pkginfo 文件中的体系结构信息。
`-b` *base-src-dir*	当 pkgmk 命令在开发系统上搜索对象时，要求将 *base-src-dir* 添加到可重定位路径名的开头。
`-d` *device*	指定应将软件包复制到 *device* 上（可能是一个绝对目录路径名、磁盘或可移除磁盘）。
`-f` *filename*	指定一个文件 *filename* 作为您的 prototype 文件。缺省名称是 prototype 或 Prototype。
`-l` *limit*	以 512 字节块为单位指定输出设备的最大大小。
`-p` *pstamp*	重写 pkginfo 文件中的生产标记定义。
`-r` *rootpath*	请求将根目录 *rootpath* 用于在开发系统上定位对象。
`-v` *version*	重写 pkginfo 文件中的版本信息。
PARAM=value	设置全局环境变量。首字母为小写字母的变量在生成时被解析。首字母为大写字母的变量被放置到 pkginfo 文件中供安装时使用。
pkginst	按软件包缩写或特定实例（例如，SUNWcadap.4）指定软件包。

有关更多信息，请参见 pkgmk(1) 手册页。

5 **验证软件包的内容。**

```
$ pkgchk -d device-name pkg-abbrev
Checking uninstalled directory format package pkg-abbrev
from device-name
## Checking control scripts.
## Checking package objects.
```

```
## Checking is complete.
$
```

-d *device-name*　指定软件包的位置。请注意，*device-name* 可以是完整目录路径名或者磁带或可移除磁盘的标识符。

pkg-abbrev　待检查的一个或多个软件包（由空格分隔）的名称。如果省略，pkgchk 命令将检查所有可用的软件包。

pkgchk 命令会列出检查软件包的哪些方面，并根据情况显示警告或错误。有关 pkgchk 命令的更多信息，请参见第 80 页中的 “验证软件包的完整性”。

注意 – 应该非常认真地对待错误。错误可能意味着需要修正脚本。如果您不认同 pkgchk 命令的输出，请检查所有错误，然后继续。

示例 2–2 生成软件包

此示例使用在第 34 页中的 “优化使用 pkgproto 命令创建的 prototype 文件”中创建的 prototype 文件。

```
$ cd /home/jane/InfoFiles
$ pkgmk
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system990708093144"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
$
```

示例 2–3 指定可重定位文件的源目录

如果您的软件包中包含可重定位文件，可使用 pkgmk 命令的 -b *base-src-dir* 选项，来指定在创建软件包期间将添加到可重定位路径名开头的路径名。如果您没有为可重定位文件使用 *path1=path2* 格式，或没有在 prototype 文件中使用 !search 命令指定搜索路径，该选项很有用。

以下命令将生成具有下列特征的软件包：

- 该软件包是使用 pkgproto 命令创建的 prototype 样例文件生成的。有关更多信息，请参见第 33 页中的 “示例－使用 pkgproto 命令创建 prototype 文件”。
- 生成软件包时没有修改 *path* 字段。
- 软件包会为 pkginfo 文件添加一个条目。

```
$ cd /home/jane/InfoFiles
$ pkgmk -o -b /home/jane
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system960716102636"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
```

在此示例中，通过指定 -o 选项在缺省目录 /var/spool/pkg 中生成软件包。此选项会覆写在示例 2–2 中创建的软件包。

示例 2–4 为信息文件和软件包对象指定不同的源目录

如果您将软件包信息文件（例如 pkginfo 和 prototype）和软件包对象放置在两个不同的目录中，可使用 pkgmk 命令的 -b *base-src-dir* 和 -r *rootpath* 选项创建软件包。如果您的软件包对象位于名为 /product/pkgbin 的目录中，而其他软件包信息文件位于名为 /product/pkgsrc 的目录，可以使用以下命令将软件包放置在 /var/spool/pkg 目录下：

```
$ pkgmk -b /product/pkgbin -r /product/pkgsrc -f /product/pkgsrc/prototype
```

可选择使用以下命令获得相同结果：

```
$ cd /product/pkgsrc
$ pkgmk -o -b /product/pkgbin
```

在此示例中，pkgmk 命令使用当前工作目录查找软件包的剩余部分（例如 prototype 和 pkginfo 信息文件）。

另请参见 如果您要为软件包添加任何可选信息文件和安装脚本，请参见第 3 章，增强软件包的功能（任务）。否则，在生成软件包之后，应该验证其完整性。第 4 章，验证和转换软件包介绍了如何执行此任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

◆ ◆ ◆ 第 3 章

3 增强软件包的功能（任务）

本章介绍如何为软件包创建可选信息文件和安装脚本。第 2 章，生成软件包中讨论了生成软件包的最低要求，而本章将讨论可以生成到软件包的附加功能。此附加功能基于您在计划如何设计软件包时所考虑的准则。有关更多信息，请参见第 16 页中的 “生成软件包之前的注意事项”。

以下是本章中概述信息的列表。

创建信息文件和安装脚本（任务图）

以下任务图描述了您可以生成到软件包的可选功能。

表 3–1 创建信息文件和安装脚本（任务图）

任务	说明	参考
1. 创建信息文件	**定义软件包相关性** 利用软件包相关性的定义，可以指定您的软件包是否与以前的版本兼容，是否依赖于其他软件包，或者其他软件包是否依赖于您的软件包。	第 47 页中的 “如何定义软件包相关性”
	编写版权信息 `copyright` 文件为您的软件应用程序提供法律保护。	第 49 页中的 “如何编写版权信息”

表 3–1 创建信息文件和安装脚本（任务图） （续）

任务	说明	参考
	在目标系统上保留额外空间 space 文件在目标系统上留出了空间块，使您可以在安装期间创建未在 pkgmap 文件中定义的文件。	第 51 页中的 “如何在目标系统上保留额外空间”
2. 创建安装脚本	**获取来自安装人员的信息** 使用 request 脚本可以获取来自软件包安装人员的信息。	第 57 页中的 “如何编写 request 脚本”
	收集安装所需的文件系统数据 使用 checkinstall 脚本可以执行目标系统的分析，并为安装设置正确的环境或完全停止安装。	第 59 页中的 “如何收集文件系统数据”
	编写过程脚本 使用过程脚本可在安装或删除过程的特定阶段提供自定义的安装说明。	第 61 页中的 “如何编写过程脚本”
	编写类操作脚本 使用类操作脚本可指定在安装和删除软件包期间对特定软件包对象组执行的一组指令。	第 68 页中的 “如何编写类操作脚本”

创建信息文件

此节讨论可选软件包信息文件。使用这些文件，您可以定义软件包相关性，提供版权信息，以及在目标系统上保留额外空间。

定义软件包相关性

您需要确定您的软件包是否依赖于其他软件包，以及是否有任何其他软件包依赖于您的软件包。可使用以下两个可选软件包信息文件定义软件包相关性和不兼容性：compver 和 depend。

通过提供 compver 文件，您可以指定与正在安装的软件包兼容的以前软件包版本。

通过提供 depend 文件，您可以定义与您的软件包相关联的三种相关性类型。这些相关性类型如下所示：

- **先决软件包** – 您的软件包依赖于其他软件包的存在
- **反向相关性** – 其他软件包依赖于您的软件包的存在

注 – 只有当无法提供 depend 文件的软件包依赖于您的软件包时，才应使用反向相关性类型。

- **不兼容软件包** – 您的软件包与指定软件包不兼容

depend 文件只解析最基本的相关性。如果您的软件包依赖于一个特定文件、其内容或者其行为，则 depend 文件不能提供足够的精确度。在这种情况下，应该使用 request 脚本或 checkinstall 脚本进行详细的相关性检查。checkinstall 脚本也是唯一能够完全停止软件包安装过程的脚本。

注 – 请确认 depend 和 compver 文件在 prototype 文件中有对应条目。文件类型应该是 i（表示软件包信息文件）。

有关更多信息，请参阅 depend(4) 和 compver(4) 手册页。

▼ 如何定义软件包相关性

1. **将包含信息文件的目录设置为当前工作目录。**

2. **如果存在您软件包的以前版本，并且您需要指定新软件包与其兼容，请使用您喜爱的文本编辑器创建名为 compver 的文件。**

   列出与您的软件包兼容的版本。使用以下格式：

   string string . . .

   对于每个兼容软件包，*string* 值与在 pkginfo 文件中指定给 VERSION 参数的值完全相同。

3. **保存所做更改，然后退出编辑器。**

4. **如果您的软件包依赖于其他软件包的存在、其他软件包依赖于您的软件包的存在，或者您的软件包与另一个软件包不兼容，请使用您喜爱的文本编辑器创建名为 depend 的文件。**

   为每种相关性添加一个条目。使用以下格式：

   type pkg-abbrev pkg-name
   (*arch*) *version*
   (*arch*) *version . . .*

   type 定义相关性类型。必须为以下字符之一：P（先决软件包）、I（不兼容软件包）或 R（反向相关性）。

   pkg-abbrev 指定软件包缩写，例如 SUNWcadap。

pkg-name 指定软件包全名，例如 `Chip designers need CAD application software to design abc chips. Runs only on xyz hardware and is installed in the usr partition.`

(*arch*) 可选。指定运行软件包的硬件类型。例如，`sparc` 或 `x86`。如果您指定体系结构，必须使用括号作为分界符。

version 可选。指定 `pkginfo` 文件中为 `VERSION` 参数所赋的值。

有关更多信息，请参见 `depend(4)`。

5 保存所做更改，然后退出编辑器。

6 完成以下任务之一：

- 如果您要创建其他信息文件和安装脚本，请跳至下一任务：第 49 页中的 “如何编写版权信息”。
- 如果您未创建 `prototype` 文件，请完成第 38 页中的 “如何使用 `pkgproto` 命令创建 `prototype` 文件”过程。跳至步骤 7。
- 如果您已经创建了 `prototype` 文件，请编辑该文件，并为刚刚创建的每个文件添加一个条目。

7 生成您的软件包。

如果需要，请参见第 41 页中的 “如何生成软件包”。

示例 3–1 `compver` 文件

在此示例中，有四个软件包版本：1.0、1.1、2.0 和新软件包 3.0。新软件包与所有三个以前版本兼容。最新版本的 `compver` 文件可能如下所示：

```
release 3.0
release 2.0
version 1.1
1.0
```

这些条目不必按版本大小顺次排序。但是，它们应与每个软件包的 `pkginfo` 文件中 `VERSION` 参数的定义完全匹配。在此示例中，软件包设计人员对于前三个版本使用了不同的格式。

示例 3–2 `depend` 文件

此示例假定样例软件包 `SUNWcadap` 要求 `SUNWcsr` 和 `SUNWcsu` 软件包已经安装在目标系统上。`SUNWcadap` 的 `depend` 文件如下所示：

```
P SUNWcsr Core Solaris, (Root)
P SUNWcsu Core Solaris, (Usr)
```

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

编写版权信息

您需要确定软件包是否应该在安装时显示版权信息。如果是，请创建 `copyright` 文件。

注 – 应该包括 `copyright` 文件，以便为您的软件应用程序提供法律保护。请与您公司的法律部门协商，以确定版权信息的准确措词。

要提供版权信息，必须创建名为 `copyright` 的文件。在安装期间，信息将完全按照文件中的形式显示（无格式设置）。有关更多信息，请参见 `copyright(4)` 手册页。

注 – 请确认 `copyright` 文件在 `prototype` 文件中有对应条目。文件类型应该是 `i`（表示软件包信息文件）。

▼ 如何编写版权信息

1. **将包含信息文件的目录设置为当前工作目录。**
2. **使用您喜爱的文本编辑器创建名为 `copyright` 的文件。**
   根据您希望在安装软件包时显示的信息键入版权信息的文本。
3. **保存所做更改，然后退出编辑器。**
4. **完成以下任务之一：**
   - 如果您要创建其他信息文件和安装脚本，请跳至下一任务：第 51 页中的 “如何在目标系统上保留额外空间”。
   - 如果您**未**创建 `prototype` 文件，请完成第 38 页中的 “如何使用 `pkgproto` 命令创建 `prototype` 文件”过程。跳至步骤 5。
   - 如果您已经创建了 `prototype` 文件，请编辑该文件，并为刚刚创建的信息文件添加一个条目。

5 **生成您的软件包。**

如果需要，请参见第 41 页中的“如何生成软件包”。

示例 3–3 `copyright` 文件

例如，部分版权信息可能如下所示：

```
Copyright (c) 2003 Company Name
All Rights Reserved

This product is protected by copyright and distributed under
licenses restricting copying, distribution, and decompilation.
```

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

在目标系统上保留额外空间

您需要确定软件包在目标系统上是否需要额外磁盘空间。此空间是除软件包对象所需空间之外的空间。如果是，请创建 `space` 信息文件。此任务不同于在安装时创建空文件和目录，如第 36 页中的“定义要在安装时创建的其他对象”所述。

`pkgadd` 命令可根据 `pkgmap` 文件中的对象定义确保有安装软件包所需的足够磁盘空间。但是，除了 `pkgmap` 文件中定义的对象所需的磁盘空间之外，软件包还可能需要更多空间。例如，软件包可能在安装后创建一个文件，该文件中可能包含数据库、日志文件，或者某个不断占用更多磁盘空间的文件。为了保证为软件包保留了空间，您应该引入一个指定磁盘空间需求的 `space` 文件。`pkgadd` 命令可检查在 `space` 文件中指定的额外空间。有关更多信息，请参阅 `space(4)` 手册页。

注 – 请确认 `space` 文件在 `prototype` 文件中有对应条目。文件类型应该是 `i`（表示软件包信息文件）。

▼ 如何在目标系统上保留额外空间

1 **将包含信息文件的目录指定为当前工作目录。**

2 **使用您喜爱的文本编辑器创建名为 space 的文件。**

指定软件包所需的所有额外磁盘空间需求。使用以下格式：

pathname blocks inodes

pathname 指定目录名，该目录可能是也可能不是文件系统的挂载点。

blocks 指定您希望保留的 512 字节块的数目。

inodes 指定所需的 inode 的数目。

有关更多信息，请参见 space(4) 手册页。

3 **保存所做更改，然后退出编辑器。**

4 **完成以下任务之一。**

- 如果您要创建安装脚本，请跳至下一任务：第 57 页中的 “如何编写 request 脚本”。
- 如果您未创建 prototype 文件，请完成第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”中的过程。跳至步骤 5。
- 如果您已经创建了 prototype 文件，请编辑该文件，并为刚刚创建的信息文件添加一个条目。

5 **生成您的软件包。**

如果需要，请参见第 41 页中的 “如何生成软件包”。

示例 3–4 space 文件

以下 space 示例文件指定了将在目标系统的 /opt 目录中保留 1000 个 512 字节块和 1 个 inode。

```
/opt   1000   1
```

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

创建安装脚本

此节讨论可选的软件包安装脚本。pkgadd 命令可自动执行使用软件包信息文件作为输入安装软件包时所需的所有操作。您**不必**提供任何软件包安装脚本。不过，如果您要为软件包创建自定义的安装过程，可以使用安装脚本来实现。安装脚本：

- 必须可由 Bourne shell (sh) 执行
- 必须包含 Bourne shell 命令和文本
- 不必包含 #!/bin/sh shell 标识符
- 不必是可执行文件

可使用四种类型的安装脚本执行自定义的操作：

- request 脚本

  request 脚本请求来自正在安装软件包的管理员的数据，以指定或重新定义环境变量。

- checkinstall 脚本

  checkinstall 脚本检查目标系统上是否有所需数据，可设置或修改软件包环境变量，以及确定安装是否继续。

  注 – checkinstall 脚本自 Solaris 2.5 及兼容发行版开始可用。

- 过程脚本

  过程脚本指定在安装或删除软件包之前或之后将调用的过程。四个过程脚本是 preinstall 、postinstall、preremove 和 postremove。

- 类操作脚本

  类操作脚本定义了应该在安装或删除期间应用于某类文件的一个操作或一组操作。您可以定义自己的类。或者，可以使用四个标准类（sed、awk、build 和 preserve）之一。

软件包安装期间的脚本处理

您使用的脚本类型取决于安装过程中何时需要该脚本的操作。软件包安装后，pkgadd 命令执行以下步骤：

1. 执行 request 脚本。

   软件包只有在此步骤中才能从正在安装软件包的管理员请求输入。

2. 执行 checkinstall 脚本。

   checkinstall 脚本收集文件系统数据，并可创建或修改环境变量定义来控制后续安装。有关软件包环境变量的更多信息，请参见第 22 页中的 “软件包环境变量”。

3. 执行 preinstall 脚本。
4. 为要安装的每个类安装软件包对象。

   这些文件的安装按逐个类进行，并相应执行类操作脚本。所处理的类列表及其安装顺序最初由 pkginfo 文件中的 CLASSES 参数定义。不过，request 脚本或 checkinstall 脚本可更改 CLASSES 参数的值。有关安装期间如何处理类的更多信息，请参见第 62 页中的 “软件包安装期间如何处理类”。

   a. 创建符号链接、设备、命名管道和所需目录。
   b. 根据常规文件的类安装这些文件（文件类型 e、v、f）

      仅针对要安装的常规文件执行类操作脚本。所有其他软件包对象都根据 pkgmap 文件中的信息自动创建。

   c. 创建所有硬链接。
5. 执行 postinstall 脚本。

软件包删除期间的脚本处理

删除软件包时，pkgrm 命令执行以下步骤：

1. 执行 preremove 脚本。
2. 为每个类删除软件包对象

   删除也按逐个类进行。删除脚本按照与安装时相反的顺序，根据 CLASSES 参数定义的序列进行处理。有关安装期间如何处理类的更多信息，请参见第 62 页中的 “软件包安装期间如何处理类”。

   a. 删除硬链接。
   b. 删除常规文件。
   c. 删除符号链接、设备和命名管道。
3. 执行 postremove 脚本。

request 脚本不会在软件包删除时进行处理。但是，该脚本的输出会保留在已安装的软件包中，供删除脚本使用。request 脚本的输出是环境变量列表。

对脚本可用的软件包环境变量

以下环境变量组对所有安装脚本可用。某些环境变量可由 request 脚本或 checkinstall 脚本修改。

- 除了必需的参数之外，request 脚本或 checkinstall 脚本可设置或修改 pkginfo 文件中的任何标准参数。pkginfo(4) 手册页中详细介绍了标准安装参数。

注 – 只能在 Solaris 2.5 发行版和兼容发行版及之后的发行版中修改 BASEDIR 参数。

- 您可定义自己的安装环境变量，方法是在 pkginfo 文件中为这些变量赋值。这种环境变量只能含字母数字字符，且首字母必须大写。这些环境变量中的任何一个都可由 request 脚本或 checkinstall 脚本更改。
- request 脚本和 checkinstall 脚本都可以定义新的环境变量，方法是为其赋值并将其放在安装环境中。
- 下表列出了通过环境对所有安装脚本可用的环境变量。这些环境变量都不能由脚本修改。

环境变量	说明
CLIENT_BASEDIR	相对于目标系统的基目录。BASEDIR 是从安装系统（一般是服务器）引用特定软件包对象时使用的变量，而 CLIENT_BASEDIR 是用于包括客户机系统中放置的文件的路径。如果 BASEDIR 存在，则 CLIENT_BASEDIR 存在，而且在没有 PKG_INSTALL_ROOT 时与 BASEDIR 完全相同。
INST_DATADIR	正在被读取的软件包所处的目录。如果正从磁带读取软件包，此变量将是一个临时目录位置，其中软件包被转换为目录格式。换句话说，假定软件包没有扩展名（例如 SUNWstuff.d），则当前软件包的 request 脚本将位于 $INST_DATADIR/$PKG/install。
PATH	sh 在调用脚本时查找命令所用的搜索列表。 PATH 通常设置为 /sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin。
PKGINST	正在安装的软件包的实例标识符。如果尚未安装软件包的其他实例，则该值是软件包缩写（例如 SUNWcadap）。否则，该值是带后缀的软件包缩写，例如 SUNWcadap.4。
PKGSAV	可保存文件以供删除脚本使用的目录，或以前保存的文件所在的目录。仅在 Solaris 2.5 发行版和兼容发行版中可用。
PKG_CLIENT_OS	安装软件包的客户机的操作系统。此变量的值是 Solaris。
PKG_CLIENT_VERSION	x.y 格式的 Solaris 版本。
PKG_CLIENT_REVISION	Solaris 内部版本修订版。
PKG_INSTALL_ROOT	将软件包安装到的目标系统上的根文件系统。只有当带有 -R 选项调用 pkgadd 和 pkgrm 命令时，此变量才存在。这种有条件的存在有利于在过程脚本中以 ${PKG_INSTALL_ROOT}/*somepath* 格式使用该变量。
PKG_NO_UNIFIED	当带有 -M 和 -R 选项调用 pkgadd 和 pkgrm 命令时设置的环境变量。此环境变量被传递给属于软件包环境的任何软件包安装脚本或软件包命令。

环境变量	说明
UPDATE	该环境变量在大多数安装环境下不存在。如果此变量确实存在（值为 yes），则表明以下两种情况之一。一种情况是系统上已安装了名称、版本和体系结构均相同的软件包。另一种情况是，该软件包正在根据管理员的指令覆写已安装的同名软件包。在这些情况中，始终会使用原始基目录。

为脚本获取软件包信息

在脚本中可使用两个命令请求获得关于软件包的信息：

- `pkginfo` 命令可返回关于软件包的信息，例如实例标识符和软件包名称。
- `pkgparam` 命令可返回请求的环境变量的值。

  有关更多信息，请参见 `pkginfo(1)` 手册页、`pkgparam(1)` 手册页和第 4 章，验证和转换软件包。

脚本的退出代码

每个脚本必须使用下表中所示的退出代码之一退出。

表 3–2 安装脚本退出代码

代码	含义
0	脚本成功完成。
1	致命错误。安装过程在此时终止。
2	警告或可能的错误状态。安装会继续。在完成之时将显示一条警告消息。
3	`pkgadd` 命令完全停止。只有 `checkinstall` 脚本会返回此代码。
10	当完成所有选定软件包的安装时，应该重新引导系统。（此值应该添加到一位数退出代码之一。）
20	在完成当前软件包的安装时应立即重新引导系统。（此值应该添加到一位数退出代码之一。）

有关安装脚本返回的退出代码的示例，请参见第 5 章，软件包创建案例研究。

注 – 随您的软件包一起提供的所有安装脚本均应在 `prototype` 文件中有一个条目。文件类型应该是 `i`（表示软件包安装脚本）。

编写 request 脚本

request 脚本是软件包可直接与安装该软件包的管理员进行交互的唯一途径。例如，该脚本可用于询问管理员是否安装软件包的可选组件。

request 脚本的输出必须是环境变量及其值的列表。此列表可以包括在 pkginfo 文件中创建的任一参数以及 CLASSES 和 BASEDIR 参数。该列表还可引入尚未在其他位置定义的环境变量。不过，适用情况下，pkginfo 文件应该始终提供缺省值。有关软件包环境变量的更多信息，请参见第 22 页中的 “软件包环境变量”。

当 request 脚本为环境变量赋值时，它必须使这些值对 pkgadd 命令和其他软件包脚本可用。

request 脚本行为

- request 脚本不能修改任何文件。该脚本仅与正在安装软件包的管理员交互，并根据这种交互创建环境变量赋值的列表。request 脚本以非特权用户 install 身份运行（如果存在该用户）。否则，该脚本将以 root 身份执行。
- pkgadd 命令使用一个用于命名 request 脚本的响应文件的参数调用该脚本。响应文件存储管理员的响应。
- 在软件包删除期间，request 脚本不会执行。不过，该脚本指定的环境变量将被保存，并在软件包删除期间可用。

request 脚本的设计规则

- 对于每个软件包只能有一个 request 脚本。必须将该脚本命名为 request。
- 应该将环境变量赋值添加到安装环境（通过将其写入到响应文件），供 pkgadd 命令和其他打包脚本使用（对于该脚本称为 $1）。
- request 脚本不能修改除 CLASSES 和 BASEDIR 参数之外的系统环境变量和标准安装环境变量。该脚本可以修改您创建的其他任何环境变量。

注 – request 脚本只能在 Solaris 2.5 和兼容发行版及之后的发行版中修改 BASEDIR 参数。

- 对于 request 脚本可以处理的每个环境变量，应该在 pkginfo 文件中为其指定一个缺省值。
- 输出列表的格式应该是 *PARAM=value*。例如：

```
CLASSES=none class1
```

- 管理员的终端被定义为 request 脚本的标准输入。

- 不要在 request 脚本中对目标系统执行任何特殊分析。根据该分析测试系统中是否存在特定二进制文件或特定行为以及设置环境变量都是有风险的。无法保证 request 脚本在安装时实际执行。安装软件包的管理员可能提供将插入环境变量的响应文件，而不会调用 request 脚本。如果 request 脚本还评估目标文件系统，该评估可能不会发生。为了进行特殊处理而对目标系统进行的分析最好留给 checkinstall 脚本完成。

注 – 如果将安装软件包的管理员可能使用 JumpStart™ 产品，那么不能以交互方式安装该软件包。此时您不应该为软件包提供 request 脚本，或者您需要与管理员沟通，希望他们应该在安装之前使用 pkgask 命令。pkgask 命令存储管理员对 request 脚本的响应。有关 pkgask 命令的更多信息，请参见 pkgask(1M) 手册页。

▼ 如何编写 request 脚本

1. **将包含信息文件的目录设置为当前工作目录。**
2. **使用您喜爱的文本编辑器创建名为 request 的文件。**
3. **完成时保存所做更改并退出编辑器。**
4. **完成以下任务之一。**
   - 如果您要创建其他安装脚本，请跳至下一任务：第 59 页中的 “如何收集文件系统数据”。
   - 如果您未创建 prototype 文件，请完成第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”过程。跳至步骤 5。
   - 如果您已经创建了 prototype 文件，请编辑该文件，并为刚刚创建的安装脚本添加一个条目。
5. **生成您的软件包。**
   如果需要，请参见第 41 页中的 “如何生成软件包”。

示例 3–5 编写 request 脚本

当 request 脚本为环境变量赋值时，它必须使这些值对 pkgadd 命令可用。此示例显示了为以下四个环境变量执行该任务的 request 脚本片段： CLASSES、NCMPBIN、 EMACS 和 NCMPMAN。假定之前已使用该脚本在与管理员的交互会话中定义了这些变量。

```
# make environment variables available to installation
# service and any other packaging script we might have

cat >$1 <<!
```

```
CLASSES=$CLASSES
NCMPBIN=$NCMPBIN
EMACS=$EMACS
NCMPMAN=$NCMPMAN
!
```

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

使用 checkinstall 脚本收集文件系统数据

checkinstall 脚本在可选的 request 脚本执行之后不久执行。checkinstall 脚本以 install 用户身份运行（如果存在这样的用户），或者以 nobody 用户身份运行。checkinstall 脚本没有更改文件系统数据的权限。不过，它可以根据所收集的信息创建或修改环境变量，以控制所发生的安装的过程。该脚本还能够完全停止安装过程。

checkinstall 脚本旨在对文件系统执行基本检查，这些检查无法使用 pkgadd 命令正常执行。例如，此脚本可用于提前检查以确定当前软件包中是否有任何文件将覆写现有文件，或者将管理常规软件相关性。depend 文件只管理软件包级别的相关性。

与 request 脚本不同，无论是否提供了响应文件，checkinstall 脚本都会执行。存在该脚本并不表明软件包是交互式软件包。可使用 checkinstall 脚本的情形包括 request 脚本被禁用或管理交互无法实现。

注 – checkinstall 脚本自 Solaris 2.5 及兼容发行版开始可用。

---

checkinstall 脚本行为

- checkinstall 脚本不能修改任何文件。此脚本只分析系统的状态，并根据该交互创建环境变量赋值的列表。要强制执行此限制，checkinstall 脚本以非特权用户 install 身份执行（如果存在该用户）。否则，此脚本将以非特权用户 nobody 身份执行。checkinstall 脚本没有超级用户权限。
- pkgadd 命令使用一个用于命名 checkinstall 脚本的响应文件的参数调用该脚本。脚本的响应文件用于存储管理员的响应。
- 在软件包删除期间，checkinstall 脚本不会执行。不过，该脚本指定的环境变量将被保存，并在软件包删除期间可用。

checkinstall 脚本的设计规则

- 对于每个软件包只能有一个 checkinstall 脚本。必须将该脚本命名为 checkinstall。

- 应该将环境变量赋值添加到安装环境（通过将其写入到响应文件），供 pkgadd 命令和其他打包脚本使用（对于该脚本称为 $1）。
- checkinstall 脚本不能修改除 CLASSES 和 BASEDIR 参数之外的系统环境变量和标准安装环境变量。该脚本可以修改您创建的其他任何环境变量。
- 对于 checkinstall 脚本可以处理的每个环境变量，应该在 pkginfo 文件中为其指定一个缺省值。
- 输出列表的格式应该是 *PARAM=value*。例如：

```
CLASSES=none class1
```

- 在 checkinstall 脚本执行期间不允许与管理员交互。所有管理员交互仅限于 request 脚本。

▼ 如何收集文件系统数据

1. **将包含信息文件的目录设置为当前工作目录。**
2. **使用您喜爱的文本编辑器创建名为 checkinstall 的文件。**
3. **完成时保存所做更改并退出编辑器。**
4. **完成以下任务之一。**
   - 如果您要创建其他安装脚本，请跳至下一任务：第 61 页中的 “如何编写过程脚本”。
   - 如果您未创建 prototype 文件，请完成第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”过程。跳至步骤 5。
   - 如果您已经创建了 prototype 文件，请编辑该文件，并为刚刚创建的安装脚本添加一个条目。
5. **生成您的软件包。**
   如果需要，请参见第 41 页中的 “如何生成软件包”。

示例 3–6 编写 checkinstall 脚本

此 checkinstall 示例脚本将检查是否安装了 SUNWcadap 软件包所需的数据库软件。

```
# checkinstall script for SUNWcadap
#
# This confirms the existence of the required specU database

# First find which database package has been installed.
```

```
pkginfo -q SUNWspcdA    # try the older one

if [ $? -ne 0 ]; then
   pkginfo -q SUNWspcdB    # now the latest

      if [ $? -ne 0 ]; then    # oops
            echo "No database package can be found. Please install the"
            echo "SpecU database package and try this installation again."
            exit 3        # Suspend
      else
            DBBASE="`pkgparam SUNWsbcdB BASEDIR`/db"    # new DB software
      fi
else
      DBBASE="`pkgparam SUNWspcdA BASEDIR`/db"    # old DB software
fi

# Now look for the database file we will need for this installation
if [ $DBBASE/specUlatte ]; then
      exit 0        # all OK
else
      echo "No database file can be found. Please create the database"
      echo "using your installed specU software and try this"
      echo "installation again."
      exit 3        # Suspend
fi
```

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

编写过程脚本

过程脚本提供了将在软件包安装或删除期间的特定点执行的一组指令。根据这些指令将执行的时间，必须将四个过程脚本命名为预定义的名称之一。这些脚本不带参数执行。

- `preinstall` 脚本

  在开始安装类之前运行。该脚本不应安装任何文件。
- `postinstall` 脚本

  在安装了所有卷之后运行。
- `preremove` 脚本

  在开始删除类之前运行。该脚本不应删除任何文件。
- `postremove` 脚本

在删除了所有类之后运行。

过程脚本行为

过程脚本以 uid=root 和 gid=other 执行。

过程脚本的设计规则

- 每个脚本应该能够多次执行，因为该脚本针对软件包的每个卷都执行一次。这意味着使用同一输入执行脚本任意次数时，会产生与只执行脚本一次相同的结果。
- 用于安装不在 pkgmap 文件中的软件包对象的每个过程脚本必须使用 installf 命令通知软件包数据库它正在添加或修改路径名。完成所有添加或修改之后，应带有 -f 选项调用此命令。只有 postinstall 和 postremove 脚本可以按这种方式安装软件包对象。有关更多信息，请参见 installf(1M) 手册页和第 5 章，软件包创建案例研究。
- 在过程脚本执行期间不允许与管理员交互。所有管理员交互仅限于 request 脚本。
- 用于删除未从 pkgmap 文件安装的文件的每个过程脚本必须使用 removef 命令通知软件包数据库它正在删除路径名。完成删除之后，应带有 -f 选项调用此命令。有关详细信息和示例，请参见 removef(1M) 手册页和第 5 章，软件包创建案例研究。

注 – 必须使用 installf 和 removef 命令，因为过程脚本不会自动与 pkgmap 文件中列出的任何路径名相关联。

▼ 如何编写过程脚本

1. **将包含信息文件的目录设置为当前工作目录。**
2. **使用您喜爱的文本编辑器创建一个或多个过程脚本。**

   必须将过程脚本命名为预定义的名称之一： preinstall、postinstall、 preremove 或 postremove。
3. **保存所做更改，然后退出编辑器。**
4. **完成以下任务之一。**
   - 如果您要创建类操作脚本，请跳至下一任务：第 68 页中的 “如何编写类操作脚本”。
   - 如果您未创建 prototype 文件，请完成第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”过程。跳至步骤 5。
   - 如果您已经创建了 prototype 文件，请编辑该文件，并为刚刚创建的每个安装脚本添加一个条目。

5 **生成您的软件包。**

如果需要，请参见第 41 页中的“如何生成软件包”。

另请参见 在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了这些任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

编写类操作脚本

定义对象类

对象类允许在安装或删除时对一组软件包对象执行一系列操作。您需要在 prototype 文件中将对象指定给一个类。必须为所有软件包对象指定类，但是缺省情况下将为无需特殊操作的对象指定 none 类。

在 pkginfo 文件中定义的安装参数 CLASSES 是将安装的类的列表（包括 none 类）。

注 – pkgmap 文件中定义的、属于未在 pkginfo 文件的此参数中列出的类的对象将**不会**被安装。

CLASSES 列表确定安装顺序。类 none 总是首先安装（如果有）而且最后删除。因为目录是所有其他文件系统对象的基础支持结构，因此应将所有目录指定给 none 类。通常使用 none 类是最安全的，虽然也可能有例外情况。此策略可确保在创建目录包含的对象之前创建目录。此外，不会在一个目录清空之前尝试删除该目录。

软件包安装期间如何处理类

下面描述了安装类时发生的系统操作。操作针对软件包的每个卷都会重复执行一次（在该卷进行安装时）。

1. pkgadd 命令创建路径名列表。

   pkgadd 命令创建操作脚本处理的路径名列表。此列表的每一行都包含源路径名和目标路径名，由空格分隔。源路径名指明要安装的对象在安装卷上所处的位置。目标路径名指明应将该对象安装到的目标系统上的位置。列表的内容受以下条件限制：

   - 列表只包含属于关联类的路径名。
   - 如果尝试创建软件包对象失败，那么目录、命名管道、字符设备、块设备和符号链接将会包括在列表中，且源路径名设置为 /dev/null。通常，这些项目由 pkgadd 命令自动创建（如果尚不存在），并根据 pkgmap 文件中的定义被赋予适当的属性（模式、所有者、组）。
   - 文件类型为 l 的链接文件在任何情况下都不会包括在列表中。指定类中的硬链接在项目 4 中创建。

2. 如果没有针对某个特定类的安装提供类操作脚本，则生成的列表中的路径名将会从卷复制到相应的目标位置。
3. 执行类操作脚本（如果存在）。

   将会使用包含项目 1 中生成的列表的标准输入调用类操作脚本。如果该卷是软件包的最后一个卷，或者该类中不再有对象存在，则将使用单个参数 ENDOFCLASS 执行该脚本。

   注 – 即使软件包中不存在该类的常规文件，也会使用一个空列表和 ENDOFCLASS 参数至少调用类操作脚本一次。

4. pkgadd 命令执行内容和属性审计，并创建硬链接。

   成功执行项目 2 或 3 之后，pkgadd 命令将审计路径名列表的内容和属性信息。pkgadd 命令会自动创建与类关联的链接。将为生成的列表中的所有路径名更正检测到的属性不一致性。

软件包删除期间如何处理类

对象是按逐个类删除的。对于软件包存在但未在 CLASSES 参数中列出的类将首先被删除（例如，使用 installf 命令安装的对象）。CLASSES 参数中列出的类按相反顺序删除。none 类总是最后删除。下面描述了删除类时发生的系统操作：

1. pkgrm 命令创建一个路径名列表。

   pkgrm 命令创建属于指定类的已安装路径名的列表。其他软件包引用的路径名将会从列表中排除，除非其文件类型为 e。e 文件类型表示该文件应该在安装或删除时进行编辑。

   如果正被删除的软件包在安装期间修改了任何类型为 e 的文件，该软件包应只删除它添加的行。请勿删除一个非空的可编辑文件。删除软件包添加的行。
2. 如果不存在类操作脚本，将会删除路径名。

   如果您的软件包没有类的删除类操作脚本，将会删除 pkgrm 命令生成的列表中的所有路径名。

   注 – 文件类型为 e（可编辑）的文件不会指定给类和关联的类操作脚本。这些文件将在此时被删除，即使路径名与其他软件包共享。

3. 如果类操作脚本存在，将会执行该脚本。

   pkgrm 命令使用包含项目 1 中生成的列表的脚本标准输入来调用类操作脚本。
4. pkgrm 命令执行审计。

   成功执行类操作脚本之后，pkgrm 命令会从软件包数据库删除对路径名的引用，除非路径名由其他软件包引用。

类操作脚本

类操作脚本定义一组将在安装或删除软件包期间执行的操作。将会根据其类定义对一组路径名执行这些操作。有关类操作脚本的示例，请参见第 5 章，软件包创建案例研究。

类操作脚本根据它将处理的类以及这些操作是在软件包安装还是删除期间发生来命名。下表中显示了两种名称格式：

名称格式	说明
`i.`*class*	在软件包安装期间，对指定类中的路径名进行处理
`r.`*class*	在软件包删除期间，对指定类中的路径名进行处理

例如，名为 `manpage` 的类的安装脚本名称将是 `i.manpage`，删除脚本名称将是 `r.manpage`。

注 – 这种文件名格式不适用于属于 `sed`、`awk` 或 `build` 系统类的文件。有关这些特殊类的更多信息，请参见第 65 页中的 “特殊系统类”。

类操作脚本行为

- 类操作脚本以 `uid=root` 和 `gid=other` 执行。
- 将会为当前卷上指定类的所有文件执行脚本。
- `pkgadd` 和 `pkgrm` 命令创建 `pkgmap` 文件中列出的属于该类的所有对象的列表。因此，类操作脚本只能处理在 `pkgmap` 中定义的属于特定类的路径名。
- 当最后一次执行类操作脚本时（即不再有属于该类的文件），将会使用关键字参数 `ENDOFCLASS` 执行类操作脚本一次。
- 在类操作脚本执行期间不允许与管理员交互。

类操作脚本的设计规则

- 如果一个软件包分布在多个卷上，则类操作脚本将针对至少包含属于类的一个文件的每个卷执行一次。因此，每个脚本必须能够多次执行。这意味着使用同一输入执行脚本任意次数时，必须产生与只执行脚本一次相同的结果。
- 当文件属于具有类操作脚本的类时，脚本必须安装该文件。`pkgadd` 命令不安装有类操作脚本的文件，虽然它确实执行安装验证。
- 类操作脚本绝不应添加、删除或修改不出现在 `pkgadd` 命令生成的列表中的路径名或系统属性。有关此列表的更多信息，请参见第 62 页中的 “软件包安装期间如何处理类”中的项目 1。
- 当脚本发现 `ENDOFCLASS` 参数时，会将后续处理操作（例如清除）放到脚本中。

- 所有管理员交互仅限于 request 脚本。请勿尝试使用类操作脚本从管理员获取信息。

特殊系统类

系统提供四个特殊类：

- sed 类

  提供一种使用 sed 指令在安装和删除软件包时编辑文件的方法。

- awk 类

  提供一种使用 awk 指令在安装和删除软件包时编辑文件的方法。

- build 类

  提供一种使用 Bourne shell 命令动态构建或修改文件的方法。

- preserve 类

  提供一种保留不应被将来的软件包安装覆写的文件的方法。

- manifest 类

  提供与清单相关联的服务管理工具 (Service Management Facility, SMF) 服务的自动安装和卸载。manifest 类应用于软件包中的所有 SMF 清单。

如果一个软件包中的多个文件需要的特殊处理可完全通过 sed、awk 或 sh 命令定义，则使用系统类安装的速度比使用多个类及其对应类操作脚本更快。

sed 类脚本

sed 类提供了一种修改目标系统上现有对象的方法。如果存在属于 sed 类的文件，sed 类操作脚本将在安装时自动执行。sed 类操作脚本的名称应该与对其执行指令的文件的名称相同。

sed 类操作脚本按以下格式提供 sed 指令：

两个命令指定应执行指令的时间。位于 !install 命令之后的 sed 指令在软件包安装期间执行。位于 !remove 命令之后的 sed 指令在软件包删除期间执行。这些命令在文件中的使用顺序无关紧要。

有关 sed 指令的更多信息，请参见 sed(1) 手册页。有关 sed 类操作脚本的示例，请参见第 5 章，软件包创建案例研究。

awk 类脚本

awk 类提供了一种修改目标系统上现有对象的方法。这些修改将以 awk 类操作脚本中的 awk 指令实现。

如果存在属于 awk 类的文件，awk 类操作脚本将在安装时自动执行。这类文件包含如下格式的 awk 类脚本指令：

两个命令指定应执行指令的时间。位于 !install 命令之后的 awk 指令在软件包安装期间执行。位于 !remove 命令之后的指令在软件包删除期间执行。可按照任意顺序使用这些命令。

awk 类操作脚本的名称应该与对其执行指令的文件的名称相同。

待修改的文件用作 awk 命令的输入，该脚本的输出最终将替换原始对象。使用该语法可能不会将环境变量传递给 awk 命令。

有关 awk 指令的更多信息，请参见 awk(1) 手册页。

build 类脚本

build 类可通过执行 Bourne shell 指令创建或修改软件包对象文件。这些指令作为软件包对象提供。如果该软件包对象属于 build 类，这些指令将会在安装时自动运行。

build 类操作脚本的名称应该与对其执行指令的文件的名称相同。该名称还必须可由 sh 命令执行。该脚本的输出将成为生成或修改后的文件新版本。如果脚本未生成任何输出，表明没有创建或修改该文件。因此，该脚本可以修改或创建文件本身。

例如，如果软件包提供了一个缺省文件 /etc/randomtable，并且目标系统上尚不存在该文件，则 prototype 文件条目可能如下所示：

```
e build /etc/randomtable ? ? ?
```

软件包对象 /etc/randomtable 可能如下所示：

```
!install
# randomtable builder
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        echo "/etc/randomtable is already in place.";
        else
        echo "# /etc/randomtable" > $PKG_INSTALL_ROOT/etc/randomtable
        echo "1121554    # first random number" >> $PKG_INSTALL_ROOT/etc/randomtable
fi

!remove
# randomtable deconstructor
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        # the file can be removed if it's unchanged
        if [ egrep "first random number" $PKG_INSTALL_ROOT/etc/randomtable ]; then
            rm $PKG_INSTALL_ROOT/etc/randomtable;
        fi
fi
```

有关使用 build 类的另一个示例，请参见第 5 章，软件包创建案例研究。

preserve 类脚本

preserve 类通过确定在安装软件包时是否要覆写现有文件来保留软件包对象文件。使用 preserve 类脚本时的两种可能情况是：

- 如果将安装的文件在目标目录中尚不存在，通常将安装该文件。
- 如果将安装的文件存在于目标目录中，将显示一条消息指出该文件已存在，此时将不会安装该文件。

preserve 脚本会将上述两种情况的结果都视为成功。只有在第二种情况中当文件无法复制到目标目录时才会失败。

自 Solaris 7 发行版开始，i.preserve 脚本以及该脚本的一个副本 i.CONFIG.prsv 以及其他类操作脚本都可在 /usr/sadm/install/scripts 目录中找到。

可修改该脚本以包括您希望保留的一个或多个文件名。

manifest 类脚本

manifest 类会自动安装或卸载与 SMF 清单相关联的服务管理工具 (Service Management Facility, SMF) 服务。如果您不熟悉 SMF，请参见《系统管理指南：基本管理》中的第 16 章 “管理服务（概述）”，获取有关如何使用 SMF 来管理服务的信息。

软件包中的的所有服务清单都应由类 manifest 来标识。安装和移除服务清单的类操作脚本包含在软件包子系统中。调用 pkgadd(1M) 时，会导入此服务清单。调用 pkgrm(1M) 时，会删除服务清单中禁用的实例。还会删除清单中没有保留实例的所有服务。如果将 -R 选项提供给 pkgadd(1M) 或 pkgrm(1M)，在系统下一次以备用引导路径重新引导时，才会完成这些服务清单操作。

以下软件包信息文件中的部分代码显示了 manifest 类的用法。

```
# packaging files
i pkginfo
i copyright
i depend
i preinstall
i postinstall
i i.manifest
i r.manifest
#
# source locations relative to the prototype file
#
d none var 0755 root sys
d none var/svc 0755 root sys
d none var/svc/manifest 0755 root sys
d none var/svc/manifest/network 0755 root sys
d none var/svc/manifest/network/rpc 0755 root sys
f manifest var/svc/manifest/network/rpc/smserver.xml 0444 root sys
```

▼ 如何编写类操作脚本

1 **将包含信息文件的目录设置为当前工作目录。**

2 **在 prototype 文件中为软件包对象指定所需类名。**

例如，将对象指定给 application 和 manpage 类将如下所示：

```
f manpage /usr/share/man/manl/myappl.1l
f application /usr/bin/myappl
```

3 **在 pkginfo 文件中修改 CLASSES 参数，使其包含要在软件包中使用的类名。**

例如，application 和 manpage 类的条目将如下所示：

```
CLASSES=manpage application none
```

注 – none 类始终最先安装，最后删除，无论它在 CLASSES 参数的定义中位置如何。

---

4 **如果您要为属于 sed、awk 或 build 类的文件创建类操作脚本，请将包含软件包对象的目录设置为当前工作目录。**

5 **为属于 sed、awk 或 build 类的文件创建类操作脚本或软件包对象。**

例如，名为 application 的类的安装脚本将命名为 i.application，而其删除脚本将命名为 r.application。

切记，当一个文件属于具有类操作脚本的类时，脚本必须安装该文件。pkgadd 命令不安装有类操作脚本的文件，虽然它确实执行安装验证。如果您定义了一个类但是没有提供类操作脚本，则对该类执行的唯一操作是将组件从安装介质复制到目标系统（缺省 pkgadd 行为）。

6 **完成以下任务之一：**

- 如果您**未**创建 prototype 文件，请完成第 38 页中的 “如何使用 pkgproto 命令创建 prototype 文件”过程，然后跳至步骤 7。
- 如果您已经创建了 prototype 文件，请编辑该文件，并为刚刚创建的每个安装脚本添加一个条目。

7 **生成您的软件包。**

如果需要，请参见第 41 页中的 “如何生成软件包”。

更多信息 下一步操作

在生成软件包之后，请安装该软件包以确认它已正确安装并验证其完整性。第 4 章，验证和转换软件包介绍了如何执行此任务，并提供了有关如何将经过验证的软件包转换为分发介质的逐步说明。

创建带签名的软件包

创建带签名软件包的过程涉及若干步骤，您需要领会一些新概念和术语。本节提供有关带签名的软件包、其术语以及证书管理的信息。此外还提供了有关如何创建带签名的软件包的逐步骤过程。

带签名的软件包

带签名的软件包是一种具有数字签名（下面定义的 PEM 编码 PKCS7 数字签名）的标准流格式软件包，该签名用于验证：

- 软件包是否来自对其进行签名的实体
- 软件包是否的确由该实体签名
- 软件包是否在该实体对其签名之后未进行修改
- 对软件包进行签名的实体是否为受信任实体

除了签名，带签名的软件包与不带签名的软件包完全相同。带签名的软件包与不带签名的软件包二进制兼容。因此，带签名的软件包可用于打包工具的旧版本。但是，在这种情况下签名会被忽略。

带签名的打包技术引入了一些新的术语和缩写，下表中介绍了这些内容。

术语	定义
ASN.1	抽象语法表示法 1－一种表示抽象对象的方法。例如，ASN.1 定义公钥证书、组成证书的所有对象以及对象的收集顺序。不过，ASN.1 不指定针对存储或传输序列化对象的方式。
X.509	ITU-T 推荐标准 X.509－指定广泛采用的 X.509 公钥证书语法。
DER	唯一编码规则－ASN.1 对象的一种二进制表示法，用于定义在计算环境中针对存储或传输序列化 ASN.1 对象的方式。
PEM	保密性增强消息－一种使用 base 64 编码和某些可选标头对文件进行编码（采用 DER 或其他二进制格式）的方式。PEM 最初用于对 MIME 类型的电子邮件进行编码。PEM 还广泛用于将证书和私钥编码到文件系统或电子邮件中存在的文件中。
PKCS7	公钥密码学标准 #7－此标准描述可能应用了加密算法的数据的通用语法，例如数字签名和数字信封。带签名的软件包包含一个嵌入的 PKCS7 签名。此签名至少包含软件包的加密摘要，以及签名者的 X.509 公钥证书。带签名的软件包还可能包含链证书。当形成从签名者的证书到本地存储的受信任证书的信任链时可以使用链证书。
PKCS12	公钥密码学标准 #12－此标准描述将加密对象存储在磁盘上的语法。软件包密钥库会以这种格式进行维护。

术语	定义
软件包密钥库	可通过软件包工具查询的证书和密钥的系统信息库。

证书管理

在创建带签名的软件包之前，您必须有软件包密钥库。此软件包密钥库以对象形式包含证书。软件包密钥库中存在两种类型的对象：

受信任证书　受信任证书包含属于其他实体的一个公钥证书。之所以这样命名受信任证书，是因为密钥库属主相信证书中的公钥确实属于由证书的“主题”（所有者）指定的身份。证书的颁发者通过对证书进行签名来确保证书的真实性。

验证签名以及启动与安全 (SSL) 服务器的连接时将使用受信任证书。

用户密钥　用户密钥用于存放敏感的加密密钥信息。加密密钥信息以受保护的格式存储，可防止进行未经授权的访问。用户密钥中既包含用户的私钥又包含与私钥相对应的公钥证书。

创建带签名的软件包时将使用用户密钥。

缺省情况下，软件包密钥库存储在 `/var/sadm/security` 目录中。缺省情况下各用户也可将自己的密钥库存储在 `$HOME/.pkg/security` 目录中。

在磁盘上，软件包密钥库可以采用两种格式：多文件格式和单文件格式。多文件格式将其对象存储在多个文件中。每种类型的对象存储在单独的文件中。必须使用相同的口令短语对所有这些文件进行加密。单文件密钥库将其所有对象存储在文件系统上的单个文件中。

用于管理证书和软件包密钥库的主要实用程序是 `pkgadm` 命令。以下各小节介绍了用于管理软件包密钥库的更为常见的任务。

将受信任证书添加到软件包密钥库

可以使用 `pkgadm` 命令将受信任证书添加到软件包密钥库。证书可以是 PEM 或 DER 格式。例如：

```
$ pkgadm addcert -t /tmp/mytrustedcert.pem
```

在此示例中，名为 `mytrustedcert.pem` 的 PEM 格式证书会添加到软件包密钥库中。

将用户证书和私钥添加到软件包密钥库

pkgadm 命令不会生成用户证书或私钥。用户证书和私钥通常从证书颁发机构（例如 Verisign）获得。此外也可作为自行签名的证书在本地生成。一旦获取了私钥和证书，即可使用 pkgadm 命令将其导入到软件包密钥库中。例如：

```
pkgadm addcert -n myname -e /tmp/myprivkey.pem /tmp/mypubcert.pem
```

在此示例中，使用了以下选项：

-n *myname*	标识软件包密钥库中您希望处理的实体 (*myname*)。*myname* 实体成为对象存储时所用的别名。
-e /tmp/*myprivkey.pem*	指定包含私钥的文件。在本例中，文件是 *myprivkey.pem*，位于 /tmp 目录。
/tmp/*mypubcert.pem*	指定名为 *mypubcert.pem* 的 PEM 格式证书文件。

检验软件包密钥库中的内容

pkgadm 命令还可用于查看软件包密钥库的内容。例如：

```
$ pkgadm listcert
```

此命令显示软件包密钥库中的受信任证书和私钥。

从软件包密钥库中删除受信任证书和私钥

pkgadm 命令可用于从软件包密钥库中删除受信任证书和私钥。

删除用户证书时，必须指定证书/密钥对的别名。例如：

```
$ pkgadm removecert -n myname
```

证书的别名是证书的通用名称，可使用 pkgadm listcert 命令标识。例如，以下命令删除一个名为 Trusted CA Cert 1 的受信任证书：

```
$ pkgadm removecert -n "Trusted CA Cert 1"
```

注 – 如果使用同一别名存储了受信任证书和用户证书，则指定 -n 选项时会将其同时删除。

创建带签名的软件包

创建带签名的软件包的过程涉及三个基本步骤：

1. 创建不带签名的目录格式的软件包。
2. 将签名证书、CA 证书和私钥导入到软件包密钥库。
3. 使用步骤 2 的证书对步骤 1 中创建的软件包进行签名。

注 – 打包工具不会创建证书。必须从证书颁发机构（例如 Verisign 或 Thawte）获取这些证书。

以下过程介绍了创建带签名软件包的每个步骤。

▼ 如何创建不带签名的目录格式的软件包

创建不带签名的目录格式的软件包过程与创建常规软件包的过程相同，如本手册前面所述。以下过程介绍如何创建此不带签名的目录格式的软件包。如果您需要更多信息，请参阅前面关于生成软件包的各节。

1 **创建 `pkginfo` 文件。**

`pkginfo` 文件应该包含以下基本内容：

```
PKG=SUNWfoo
BASEDIR=/
NAME=My Test Package
ARCH=sparc
VERSION=1.0.0
CATEGORY=application
```

2 **创建 `prototype` 文件。**

`prototye` 文件应该包含以下基本内容：

```
$cat prototype
i pkginfo
d none usr 0755 root sys
d none usr/bin 0755 root bin
f none usr/bin/myapp=/tmp/myroot/usr/bin/myapp 0644 root bin
```

3 **列出对象源目录的内容。**

例如：

```
$ ls -lR /tmp/myroot
```

输出可能如下所示：

```
/tmp/myroot:
total 16
drwxr-xr-x   3 abc      other        177 Jun  2 16:19 usr

/tmp/myroot/usr:
total 16
drwxr-xr-x   2 abc      other        179 Jun  2 16:19 bin

/tmp/myroot/usr/bin:
total 16
-rw-------   1 abc      other       1024 Jun  2 16:19 myapp
```

4 创建不带签名的软件包。

```
pkgmk -d ‘pwd‘
```

输出可能如下所示：

```
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "syrinx20030605115507"
WARNING: parameter <CLASSES> set to "none"
## Attempting to volumize 3 entries in pkgmap.
part  1 -- 84 blocks, 7 entries
## Packaging one part.
/tmp/SUNWfoo/pkgmap
/tmp/SUNWfoo/pkginfo
/tmp/SUNWfoo/reloc/usr/bin/myapp
## Validating control scripts.
## Packaging complete.
```

软件包现在会存在于当前目录中。

▼ 如何将证书导入到软件包密钥库

要导入的证书和私钥必须以 PEM 或 DER 编码的 X.509 证书和私钥形式存在。此外，必须先将任何将您的签名证书与证书颁发机构证书相关联的中间或“链”证书导入到软件包密钥库中，然后才能对软件包进行签名。

注 – 各证书颁发机构可能会颁发不同格式的证书。要将证书和私钥从 PKCS12 文件提取到 PEM 编码的 X.509 文件（适于导入到软件包密钥库），请使用免费软件转换实用程序，例如 OpenSSL。

如果您的私钥进行了加密（通常应该如此），系统将提示您输入口令短语。此外，还将提示您输入口令以保护生成的软件包密钥库。您可以选择不提供任何口令，但是这样做会导致软件包密钥库不会加密。

以下过程介绍了证书格式正确时，如何使用 pkgadm 命令导入证书。

1 **导入在 PEM 或 DER 编码的 X.509 证书文件中找到的所有证书颁发机构证书。**

例如，要导入在 ca.pem 文件中找到的所有证书颁发机构证书，应键入以下内容：

```
$ pkgadm addcert -k ~/mykeystore -ty ca.pem
```

输出可能如下所示：

```
Trusting certificate <VeriSign Class 1 CA Individual \
Subscriber-Persona Not Validated>
Trusting certificate </C=US/O=VeriSign, Inc./OU=Class 1 Public \
Primary Certification Authority
Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
For Verification: Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
Certificate(s) from <ca.pem> are now trusted
```

为了将您的签名密钥导入到软件包密钥库，必须提供别名，供以后签名软件包时使用。如果您要从软件包密钥库中删除密钥，也可能会使用该别名。

例如，要从 sign.pem 文件中导入您的签名密钥，应键入以下内容：

```
$ pkgadm addcert -k ~/mykeystore -n mycert sign.pem
```

输出可能如下所示：

```
Enter PEM passphrase:
Enter Keystore Password:
Successfully added Certificate <sign.pem> with alias <mycert>
```

2 **检验证书是否在软件包密钥库中。**

例如，要在密钥库中查看在上一步骤中创建的证书，应键入以下内容：

```
$ pkgadm listcert -k ~/mykeystore
```

▼ 如何对软件包签名

一旦将证书导入到软件包密钥库，即可对软件包签名。实际的软件包签名工作是使用 pkgtrans 命令完成的。

- **使用 pkgtrans 命令对软件包签名。提供不带签名的软件包的位置，并提供用于签名软件包的密钥别名。**

  例如，使用前面过程中的示例，您应键入以下内容创建名为 SUNWfoo.signed 的带签名软件包：

  ```
  $ pkgtrans -g -k ~/mykeystore -n mycert . ./SUNWfoo.signed SUNWfoo
  ```

  此命令的输出可能如下所示：

  ```
  Retrieving signing certificates from keystore </home/user/mykeystore>
  Enter keystore password:
  Generating digital signature for signer <Test User>
  Transferring <SUNWfoot> package instance
  ```

  带签名的软件包会在 SUNWfoo.signed 文件中创建且采用软件包流格式。此带签名的软件包适合于通过 pkgadd 命令和 URL 复制到 Web 站点并进行安装。

◆ ◆ ◆ 第 4 章

4 验证和转换软件包

本章介绍如何验证软件包的完整性以及将其转换为分发介质（例如软盘或 CD-ROM）。

以下是本章中概述信息的列表：

验证和转换软件包（任务图）

下表介绍了为了验证软件包的完整性以及将其转换为分发介质所应遵循的步骤。

表 4–1 验证和转换软件包（任务图）

任务	说明	参考
1. 生成软件包	在磁盘上生成软件包。	第 2 章，生成软件包
2. 安装软件包	通过安装软件包并确保没有安装错误来测试软件包。	第 79 页中的“如何在独立系统或服务器上安装软件包”
3. 验证软件包的完整性	使用 pkgchk 命令验证软件包的完整性。	第 80 页中的“如何验证软件包的完整性”
4. 获取其他软件包信息	**可选**。使用 `pkginfo` 和 `pkgparam` 命令执行特定于软件包的验证。	第 81 页中的“显示有关已安装的软件包的附加信息”
5. 删除已安装的软件包	使用 `pkgrm` 命令从系统中删除已安装的软件包。	第 86 页中的“如何删除软件包”

表 4–1 验证和转换软件包（任务图） （续）

任务	说明	参考
6. 将软件包转换为分发介质	使用 pkgtrans 命令将软件包（以软件包格式）转换为分发介质。	第 87 页中的 “如何将软件包转换为分发介质”

安装软件包

软件包使用 pkgadd 命令进行安装。此命令从分发介质或目录转换软件包的内容并将其安装到系统上。

本节提供有关安装软件包以便验证安装正确的基本安装说明。

安装软件数据库

系统中安装的所有软件包的信息都保存在安装软件数据库中。对于软件包中的每个对象都有一个条目，其中包含组件名、组件的驻留位置及其类型等信息。这类条目包含以下信息：组件所属软件包的记录；可能引用该组件的其他软件包；以及路径名、组件的驻留位置和组件类型等信息。条目由 pkgadd 和 pkgrm 命令自动添加和删除。您可以使用 pkgchk 和 pkginfo 命令来查看数据库中的信息。

与每个软件包组件相关联的信息有两种。属性信息描述了组件本身。例如，组件的访问权限、所有者 ID 和组 ID 是属性信息。内容信息描述组件的内容，例如文件大小和上次修改时间。

安装软件数据库可跟踪软件包的状态。软件包或者是完全安装（它已经成功完成安装过程），或者是部分安装（它没有成功完成安装过程）。

当软件包部分安装时，在安装终止之前可能已安装软件包的某些部分；因此，部分软件包被安装并记录在数据库中，而部分软件包则没有安装。当您重新安装软件包时，系统将提示您从上次安装停止的位置开始安装，因为 pkgadd 命令可以访问数据库并检测哪些部分已经安装。您还可以使用 pkgrm 命令，根据安装软件数据库中的信息删除已经安装的部分。

与 pkgadd 命令交互

如果 pkgadd 命令遇到问题，它会首先检查安装管理文件中的说明。（有关更多信息，请参见 admin(4)。）如果不存在任何说明，或者如果管理文件中的相关参数被设置为 ask，pkgadd 将显示一条消息，描述该问题并提示用户进行答复。该提示通常是 Do you want to continue with this installation? 您应该回答 yes、no 或 quit。

如果您已指定多个软件包，回答 no 将停止正在进行的软件包安装，但 pkgadd 会继续安装其他软件包。quit 指示 pkgadd 应该停止所有软件包的安装。

在同构环境中的独立系统或服务器上安装软件包

本节介绍如何在同构环境中的独立系统或服务器系统上安装软件包。

▼ 如何在独立系统或服务器上安装软件包

1 **生成软件包。**

如果需要，请参见第 40 页中的 “生成软件包”。

2 **以超级用户身份登录系统。**

3 **向系统中添加软件包。**

```
# pkgadd -d device-name [pkg-abbrev...]
```

-d *device-name*	指定软件包的位置。请注意，*device-name* 可以是完整的目录路径名或者磁带、软盘或可移除磁盘的标识符。
pkg-abbrev	要添加的一个或多个软件包的名称（以空格分隔）。如果省略，pkgadd 将安装所有可用的软件包。

示例 4–1 在独立系统和服务器上安装软件包

要从名为 /dev/rmt/0 的磁带设备安装名为 pkgA 的软件包，您需要输入以下命令：

```
# pkgadd -d /dev/rmt/0 pkgA
```

您还可以同时安装多个软件包（只要您使用空格分隔软件包名称），如下所示：

```
# pkgadd -d /dev/rmt/0 pkgA pkgB pkgC
```

如果您不指定软件包所驻留的设备，该命令将检查缺省的假脱机目录 (/var/spool/pkg)。如果软件包不在缺省的假脱机目录中，安装将失败。

另请参见 如果您已准备好执行下一个任务，请参见第 80 页中的 “如何验证软件包的完整性”。

验证软件包的完整性

使用 pkgchk 命令可以检查软件包的完整性、它们是已经安装在系统上还是处于软件包格式（准备使用 pkgadd 命令安装）。它确认软件包结构或已安装的文件和目录，或者显示有关软件包对象的信息。pkgchk 命令可以列出或检查以下内容：

- 软件包安装脚本。
- 系统中当前安装的对象的内容和/或属性。
- 假脱机的已卸载软件包的内容。
- 指定的 pkgmap 文件中描述的对象的内容和/或属性。

有关此命令的更多信息，请参阅 pkgchk(1M)。

pkgchk 命令执行两种检查。它检查文件属性（文件的权限和拥有权以及块或字符特殊设备的主/从设备号）和文件内容（大小、校验和及修改日期）。缺省情况下，该命令同时检查文件属性和文件内容。

pkgchk 命令还将已安装的软件包的文件属性和内容与安装软件数据库进行比较。与软件包有关的条目自安装以来可能已经更改；例如，另一个软件包可能更改了软件包组件。数据库会反映该更改。

▼ 如何验证软件包的完整性

1 **安装软件包。**

如果需要，请参见第 79 页中的 “如何在独立系统或服务器上安装软件包”。

2 **验证软件包的完整性。**

```
# pkgchk  [-v] [-R root-path] [pkg-abbrev...]
```

-v	在处理文件的过程中列出文件。
-R *root-path*	指定客户机系统的根文件系统的位置。
pkg-abbrev	要检查的一个或多个软件包的名称（以空格分隔）。如果省略，pkgchk 将检查所有可用的软件包。

示例 4–2 验证软件包的完整性

本示例演示应该用于验证已安装软件包的完整性的命令。

```
$ pkgchk pkg-abbrev
$
```

如果有错误，pkgchk 命令将列出这些错误。否则，它不会列出任何内容并且返回退出代码 0。如果您不提供软件包缩写，它将检查系统中的所有软件包。

此外，您还可以使用 -v 选项，该选项将在没有发生错误时列出软件包中文件的列表。例如：

```
$ pkgchk -v SUNWcadap
/opt/SUNWcadap
/opt/SUNWcadap/demo
/opt/SUNWcadap/demo/file1
/opt/SUNWcadap/lib
/opt/SUNWcadap/lib/file2
/opt/SUNWcadap/man
/opt/SUNWcadap/man/man1
/opt/SUNWcadap/man/man1/file3.1
/opt/SUNWcadap/man/man1/file4.1
/opt/SUNWcadap/man/windex
/opt/SUNWcadap/srcfiles
/opt/SUNWcadap/srcfiles/file5
/opt/SUNWcadap/srcfiles/file6
$
```

如果您需要验证客户机系统的根文件系统上安装的软件包，请使用以下命令：

```
$ pkgchk -v -R root-path  pkg-abbrev
```

另请参见 如果您已准备好执行下一个任务，请参见第 86 页中的 “如何使用 pkginfo 命令获取信息”。

显示有关已安装的软件包的附加信息

可以使用其他两个命令来显示有关已安装的软件包的信息：

- pkgparam 命令显示参数值。
- pkginfo 命令显示安装软件数据库中的信息。

pkgparam 命令

pkgparam 命令可以显示与您在命令行上指定的参数相关联的值。这些值是从特定软件包的 pkginfo 文件或者从您指定的文件中获取的。每行显示一个参数值。您可以仅显示这些值，也可以同时显示参数及其值。

▼ 如何使用 pkgparam 命令获取信息

1 **安装软件包。**
如果需要，请参见第 79 页中的“如何在独立系统或服务器上安装软件包”。

2 **显示有关软件包的附加信息。**

```
# pkgparam [-v] pkg-abbrev [param...]
```

`-v`	显示参数的名称及值。
pkg-abbrev	特定软件包的名称。
param	指定要显示其值的一个或多个参数。

示例 4–3 使用 pkgparam 命令获取信息

例如，要仅显示值，请使用以下命令。

```
$ pkgparam SUNWcadap
none
/opt
US/Mountain
/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin
/usr/sadm/sysadm
SUNWcadap
Chip designers need CAD application software to design abc
chips.  Runs only on xyz hardware and is installed in the usr
partition.
system
release 1.0
SPARC
venus990706083849
SUNWcadap
/var/sadm/pkg/SUNWcadap/save
Jul 7 1999 09:58
$
```

要显示参数及其值，请使用以下命令。

```
$ pkgparam -v SUNWcadap
pkgparam -v SUNWcadap
CLASSES='none'
BASEDIR='/opt'
TZ='US/Mountain'
PATH='/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin'
OAMBASE='/usr/sadm/sysadm'
```

```
PKG='SUNWcadap'
NAME='Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.'
CATEGORY='system'
VERSION='release 1.0'
ARCH='SPARC'
PSTAMP='venus990706083849'
PKGINST='SUNWcadap'
PKGSAV='/var/sadm/pkg/SUNWcadap/save'
INSTDATE='Jul 7 1999 09:58'
$
```

或者，如果您想要显示特定参数的值，请使用以下格式：

```
$ pkgparam SUNWcadap BASEDIR
/opt
$
```

有关更多信息，请参阅 pkgparam(1)。

另请参见 如果您已准备好执行下一个任务，请参见第 86 页中的 “如何删除软件包”。

pkginfo 命令

您可以使用 pkginfo 命令显示有关已安装的软件包的信息。此命令具有多个选项，这些选项使您能够定制显示的格式和内容。

您可以请求有关任意数量的软件包实例的信息。

缺省 pkginfo 显示

在不带任何选项的情况下执行 pkginfo 命令时，该命令将显示已经完整安装到系统中的所有软件包的类别、软件包实例和软件包名称。显示按照类别组织，如以下示例所示。

```
$ pkginfo
.
.
.
system      SUNWinst       Install Software
system      SUNWipc        Interprocess Communications
system      SUNWisolc      XSH4 conversion for ISO Latin character sets
application SUNWkcspf      KCMS Optional Profiles
application SUNWkcspg      KCMS Programmers Environment
application SUNWkcsrt      KCMS Runtime Environment
.
```

```
.
.
$
```

定制 pkginfo 显示的格式

您可以按以下三种格式之一获得 pkginfo 显示： 短格式、简明格式和长格式。

短格式是缺省格式。如第 83 页中的 “缺省 pkginfo 显示”所示，该格式仅显示类别、软件包缩写和软件包全名。

简明格式显示软件包缩写、软件包名称、软件包体系结构（如果有）和软件包版本（如果有）。使用 -x 选项可以请求简明格式，如下一个示例所示。

```
$ pkginfo -x
.
.
.
SUNWipc         Interprocess Communications
                (sparc) 11.8.0,REV=1999.08.20.12.37
SUNWisolc       XSH4 conversion for ISO Latin character sets
                (sparc) 1.0,REV=1999.07.10.10.10
SUNWkcspf       KCMS Optional Profiles
                (sparc) 1.1.2,REV=1.5
SUNWkcspg       KCMS Programmers Environment
                (sparc) 1.1.2,REV=1.5
.
.
.
$
```

使用 -l 选项可生成长格式显示，其中包括有关软件包的所有可用信息，如下面的示例所示。

```
$ pkginfo -l SUNWcadap
   PKGINST:  SUNWcadap
      NAME:  Chip designers need CAD application software to
design abc chips.  Runs only on xyz hardware and is installed
in the usr partition.
  CATEGORY:  system
      ARCH:  SPARC
   VERSION:  release 1.0
   BASEDIR:  /opt
    PSTAMP:  system980706083849
  INSTDATE:  Jul 7 1999 09:58
    STATUS:  completely installed
     FILES:     13 installed pathnames
                 6 directories
```

```
                3 executables
             3121 blocks used (approx)
$
```

pkginfo 长格式的参数说明

下表介绍了可以为每个软件包显示的软件包参数。只有为参数赋值之后，才会显示参数及其值。

表 4–2 软件包参数

参数	说明
ARCH	此软件包支持的体系结构。
BASEDIR	软件包所驻留的基目录（如果该软件包是可重定位的则显示）。
CATEGORY	此软件包所属的一个或多个软件类别（例如，system 或 application）。
CLASSES	为软件包定义的类的列表。该列表的顺序确定类的安装顺序。首先列出的类将首先安装（逐个介质进行安装）。此参数可能由 request 脚本修改。
DESC	描述软件包的文本。
EMAIL	用户查询所使用的电子邮件地址。
HOTLINE	有关如何获得关于此软件包的热线帮助的信息。
INTONLY	当设置为任何非 NULL 值时，指示仅应该以交互方式安装软件包，
ISTATES	允许软件包安装具有的运行状态的列表（例如， S s 1）。
MAXINST	一台计算机上同时应该允许存在的软件包实例的最大数目。缺省情况下，仅允许存在一个软件包实例。
NAME	软件包名称，通常是描述软件包缩写的文本。
ORDER	一个类列表，定义将这些类放在介质上的顺序。由 pkgmk 命令在创建软件包时使用。此参数中未定义的类将使用标准排序过程放置在介质上。
PKGINST	所安装的软件包的缩写。
PSTAMP	此软件包的生产标记。
RSTATES	允许软件包删除操作具有的运行状态的列表（例如，S s 1）。
ULIMIT	如果设置，此参数将作为参数传递给 ulimit 命令，该命令在安装期间设置最大文件大小。这仅适用于过程脚本所创建的文件。
VENDOR	提供软件包的供应商的名称。
VERSION	此软件包的版本。
VSTOCK	供应商提供的物料编号。

有关 pkginfo 命令的详细信息，请参阅 pkginfo(1) 手册页。

▼ 如何使用 pkginfo 命令获取信息

1 **安装软件包。**

如果需要，请参见第 79 页中的“如何在独立系统或服务器上安装软件包”。

2 **显示有关软件包的附加信息。**

```
# pkginfo [-x | -l] [pkg-abbrev]
```

-x	以简明格式显示软件包信息。
-l	以长格式显示软件包信息。
pkg-abbrev	特定软件包的名称。如果省略，pkginfo 命令将以缺省格式显示有关所有已安装软件包的信息。

更多信息 下一步操作

如果您已准备好执行下一个任务，请参见第 86 页中的“如何删除软件包”。

删除软件包

由于 pkgrm 命令更新软件产品数据库中的信息，因此在删除软件包时一定要使用 pkgrm 命令，即使您可能倾向于使用 rm 命令。例如，您可能会使用 rm 命令删除二进制可执行文件，但这与使用 pkgrm 删除包括该二进制可执行文件的软件包不同。如果使用 rm 命令来删除软件包的文件，则会破坏软件产品数据库。（如果您确实只想删除一个文件，可以使用 removef 命令，该命令将正确地更新软件产品数据库。）

▼ 如何删除软件包

1 **以超级用户身份登录系统。**

2 **删除已安装的软件包。**

```
# pkgrm pkg-abbrev ...
```

pkg-abbrev	一个或多个软件包的名称（以空格分隔）。如果省略， pkgrm 将删除所有可用的软件包。

3 **使用 pkginfo 命令验证软件包已成功删除。**

```
$ pkginfo | egrep pkg-abbrev
```

如果安装了 *pkg-abbrev*，pkginfo 命令将返回有关它的一行信息。否则，pkginfo 返回系统提示符。

将软件包转换为分发介质

pkgtrans 命令移动软件包并且执行软件包格式转换。可以使用 pkgtrans 命令为可安装的软件包执行下列转换：

- 文件系统格式到数据流格式
- 数据流格式到文件系统格式
- 一种文件系统格式到另一种文件系统格式

▼ 如何将软件包转换为分发介质

1 **生成您的软件包，同时创建一个目录格式软件包（如果您尚未这样做）。**

有关更多信息，请参见第 41 页中的 “如何生成软件包”。

2 **安装该软件包以验证其安装正确。**

如果需要，请参见第 79 页中的 “如何在独立系统或服务器上安装软件包”。

3 **验证软件包的完整性。**

如果需要，请参见第 80 页中的 “如何验证软件包的完整性”、第 86 页中的 “如何使用 pkginfo 命令获取信息”和第 82 页中的 “如何使用 pkgparam 命令获取信息”。

4 **从系统中删除已安装的软件包。**

如果需要，请参见第 86 页中的 “如何删除软件包”。

5 **将软件包（以软件包格式）转换为分发介质。**

要执行基本转换，请执行以下命令：

```
$ pkgtrans device1 device2 [pkg-abbrev...]
```

device1	软件包当前驻留的设备的名称。
device2	要将经过转换的软件包写入到的设备的名称。
[*pkg-abbrev*]	一个或多个软件包缩写。

如果没有指定软件包名称，系统会将驻留在 *device1* 中的所有软件包转换并写入到 *device2*。

注 – 如果有多个软件包实例驻留在 *device1* 中，您必须对软件包使用实例标识符。有关软件包标识符的说明，请参见第 24 页中的 “定义软件包实例”。当要转换的软件包的实例已经存在于 *device2* 上时，`pkgtrans` 命令不执行转换。如果实例已经存在，您可以使用 `-o` 选项告诉 `pkgtrans` 命令覆写目标设备上现有的所有实例，并可以使用 `-n` 选项告诉该命令创建一个新实例。请注意，当 *device2* 支持数据流格式时，此检查不适用。

更多信息

下一步操作

至此，您已经完成设计、生成、验证和转换软件包所需的步骤。如果您对某些案例研究感兴趣，请参见第 5 章，软件包创建案例研究。如果您对先进的软件包设计理念感兴趣，请参见第 6 章，创建软件包的高级技术。

◆ ◆ ◆ 第 5 章

5 软件包创建案例研究

本章提供了案例研究以展示打包方案，例如有条件地安装对象、在运行时确定要创建的文件数，以及在软件包安装和删除期间修改现有数据文件。

每个案例研究都以一段描述开始，然后介绍采用的打包技术的列表、使用这些技术时采取的方法，以及与案例研究相关的样例文件和脚本。

以下是本章中案例研究的列表：

请求来自管理员的输入

在此案例研究中，软件包有三种类型的对象。管理员可以选择安装哪一种对象类型，并选择对象在安装计算机上的位置。

技术

此案例研究展示以下技术：

- 使用用于建立多个基目录的参数化路径名（对象路径名中有变量）

  有关参数化路径名的信息，请参见第 31 页中的“参数化路径名”。
- 使用 request 脚本请求来自管理员的输入

有关 request 脚本的信息，请参见第 56 页中的 “编写 request 脚本”。

- 为安装参数设置条件值。

方法

要在此案例研究中设置选择性安装，您必须完成以下任务：

- 为每种可安装对象的类型定义一个类。

  在此案例研究中，三种对象类型是：软件包可执行文件、手册页和 emacs 可执行文件。每种类型都有其自己的类：分别是 bin、man 和 emacs。请注意在 prototype 文件中，所有对象文件均属于这三个类之一。

- 在 pkginfo 文件中将 CLASSES 参数初始化为 null。

  通常在定义类时，应该在 pkginfo 文件的 CLASSES 参数中列出该类。否则，不会安装属于该类的任何对象。对于此案例研究，参数最初设置为 null，意味着不会安装任何对象。 CLASSES 参数将由 request 脚本根据管理员的选择进行更改。这样，CLASSES 参数会仅设置为管理员希望安装的那些对象类型。

  注 – 通常可取的方法是将参数设置为缺省值。如果此软件包有对所有三种对象类型通用的组件，可以将其指定给 none 类，然后将 CLASSES 参数设置为等于 none。

- 将参数化路径名插入到 prototype 文件。

  request 脚本会将这些环境变量设置为管理员提供的值。然后，pkgadd 命令在安装时解析这些环境变量，从而知道软件包的安装位置。

  此示例中使用的三个环境变量在 pkginfo 文件中设置为其缺省值，作用如下：

  - $NCMPBIN 定义对象可执行文件的位置
  - $NCMPMAN 定义手册页的位置
  - $EMACS 定义 emacs 可执行文件的位置

  prototype 示例文件显示了如何定义含变量的对象路径名。

- 创建 request 脚本，用于询问管理员应该安装软件包的哪些部分及其放置位置。

  此软件包的 request 脚本会询问管理员两个问题：

  - 是否应该安装软件包的此部分?

    当答复为是时，相应的类名被添加到 CLASSES 参数。例如，当管理员选择安装与此软件包关联的手册页时，类 man 将会添加到 CLASSES 参数。

  - 如果是，应该将软件包的此部分放置在什么位置?

    此时将设置相应的环境变量来响应此问题。在手册页示例中，变量 $NCMPMAN 设置为响应值。

  对于三种对象类型的每种类型都会重复这两个问题。

在 request 脚本的末尾，参数将设置为对 pkgadd 命令和所有其他打包脚本的安装变量可用。request 脚本通过将这些定义写入调用实用程序提供的文件来实现这一点。对于此案例研究，没有提供其他脚本。

当查看此案例研究的 request 脚本时，请注意，上述问题由数据验证工具 ckyorn 和 ckpath 生成。有关这些工具的更多信息，请参见 ckyorn(1) 和 ckpath(1)。

案例研究文件

pkginfo 文件

```
PKG=ncmp
NAME=NCMP Utilities
CATEGORY=application, tools
BASEDIR=/
ARCH=SPARC
VERSION=RELEASE 1.0, Issue 1.0
CLASSES=""
NCMPBIN=/bin
NCMPMAN=/usr/man
EMACS=/usr/emacs
```

prototype 文件

```
i pkginfo
i request
x bin $NCMPBIN 0755 root other
f bin $NCMPBIN/dired=/usr/ncmp/bin/dired 0755 root other
f bin $NCMPBIN/less=/usr/ncmp/bin/less 0755 root other
f bin $NCMPBIN/ttype=/usr/ncmp/bin/ttype 0755 root other
f emacs $NCMPBIN/emacs=/usr/ncmp/bin/emacs 0755 root other
x emacs $EMACS 0755 root other
f emacs $EMACS/ansii=/usr/ncmp/lib/emacs/macros/ansii 0644 root other
f emacs $EMACS/box=/usr/ncmp/lib/emacs/macros/box 0644 root other
f emacs $EMACS/crypt=/usr/ncmp/lib/emacs/macros/crypt 0644 root other
f emacs $EMACS/draw=/usr/ncmp/lib/emacs/macros/draw 0644 root other
f emacs $EMACS/mail=/usr/ncmp/lib/emacs/macros/mail 0644 root other
f emacs $NCMPMAN/man1/emacs.1=/usr/ncmp/man/man1/emacs.1 0644 root other
d man $NCMPMAN 0755 root other
d man $NCMPMAN/man1 0755 root other
f man $NCMPMAN/man1/dired.1=/usr/ncmp/man/man1/dired.1 0644 root other
f man $NCMPMAN/man1/ttype.1=/usr/ncmp/man/man1/ttype.1 0644 root other
f man $NCMPMAN/man1/less.1=/usr/ncmp/man/man1/less.1 0644 inixmr other
```

request 脚本

```
trap 'exit 3' 15
# determine if and where general executables should be placed
ans=`ckyorn -d y \
-p "Should executables included in this package be installed"
` || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES bin"
   NCMPBIN=`ckpath -d /usr/ncmp/bin -aoy \
   -p "Where should executables be installed"
   ` || exit $?
fi
# determine if emacs editor should be installed, and if it should
# where should the associated macros be placed
ans=`ckyorn -d y \
-p "Should emacs editor included in this package be installed"
` || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES emacs"
   EMACS=`ckpath -d /usr/ncmp/lib/emacs -aoy \
   -p "Where should emacs macros be installed"
   ` || exit $?
fi
```

请注意，request 脚本可以在不在文件系统上留下任何文件的情况下退出。对于 2.5 和兼容版本之前的 Solaris 版本上的安装（其中没有 checkinstall 脚本可用），request 脚本是以确保安装成功所需的任何方式测试文件系统的正确位置。当 request 脚本以代码 1 退出时，安装将完全退出。

这些示例文件显示了使用参数化路径建立多个基目录。然而，首选的方法涉及使用由 pkgadd 命令管理并验证的 BASEDIR 参数。每当使用多个基目录时，在同一个平台上安装多个版本和体系结构时应格外小心。

在安装时创建文件并在删除期间保存文件

在此案例研究中，将在安装时创建一个数据库文件，并在删除软件包时保存数据库副本。

技术

此案例研究展示以下技术：

- 使用类和类操作脚本对不同的对象集执行特殊操作

有关更多信息，请参见第 62 页中的 “编写类操作脚本”。

- 使用 space 文件通知 pkgadd 命令要正确安装此软件包需要额外空间

  有关 space 文件的更多信息，请参见第 50 页中的 “在目标系统上保留额外空间”。
- 使用 installf 命令安装没有在 prototype 和 pkgmap 文件中定义的文件。

方法

要为此案例研究在安装时创建一个数据库文件，并在删除时保存副本，您必须完成以下任务：

- 定义三个类。

  在此案例研究中，软件包要求在 CLASSES 参数中定义以下三个类：
  - 标准类 none，该类包含一组属于子目录 bin 的进程。
  - admin 类，该类包含可执行文件 config 和一个包含数据文件的目录。
  - cfgdata 类，该类包含一个目录。
- 使软件包可共同重定位。

  请注意在 prototype 文件中，没有路径名以斜线或环境变量开头，这表明它们是可共同重定位的。
- 计算数据库文件需要的空间量，并创建一个要随软件包一起提供的 space 文件。此文件通知 pkgadd 命令软件包需要额外空间并指定空间量。
- 为 admin 类创建类操作脚本 (i.admin)。

  该样例脚本使用属于 admin 类的数据文件初始化数据库。要执行此任务，该脚本执行以下操作：
  - 将源数据文件复制到相应目标位置
  - 创建一个名为 config.data 的空文件，并将其指定给 cfgdata 类
  - 执行 bin/config 命令（与软件包一起提供并已安装）以使用属于 admin 类的数据文件填充 config.data 数据库文件
  - 执行 installf -f 命令完成 config.data 的安装

  由于在删除时 admin 类不需要特殊操作，因此不会创建删除类操作脚本。这意味着 admin 类中的所有文件和目录都会从系统中删除。
- 为 cfgdata 类创建删除类操作脚本 (r.cfgdata)。

  删除脚本会在删除数据库文件之前创建该文件的副本。由于在安装时此类不需要特殊操作，因此不需要安装类操作脚本。

  切记删除脚本的输入是要删除的路径名的列表。路径名始终按反向字母顺序显示。此删除脚本将文件复制到名为 $PKGSAV 的目录。当所有路径名都被处理后，脚本会返回并删除与 cfgdata 类关联的所有目录和文件。

此删除脚本的结果是将 config.data 复制到 $PKGSAV，然后删除 config.data 文件和数据目录。

案例研究文件

pkginfo 文件

```
PKG=krazy
NAME=KrAzY Applications
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1
CLASSES=none cfgdata admin
```

prototype 文件

```
i pkginfo
i request
i i.admin
i r.cfgdata
d none bin 555 root sys
f none bin/process1 555 root other
f none bin/process2 555 root other
f none bin/process3 555 root other
f admin bin/config 500 root sys
d admin cfg 555 root sys
f admin cfg/datafile1 444 root sys
f admin cfg/datafile2 444 root sys
f admin cfg/datafile3 444 root sys
f admin cfg/datafile4 444 root sys
d cfgdata data 555 root sys
```

space 文件

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
```

i.admin 类操作脚本

```
# PKGINST parameter provided by installation service
# BASEDIR parameter provided by installation service
while read src dest
do
   cp $src $dest || exit 2
```

```
done
# if this is the last time this script will be executed
# during the installation, do additional processing here.
if [ "$1" = ENDOFCLASS ]
then
# our config process will create a data file based on any changes
# made by installing files in this class; make sure the data file
# is in class ‘cfgdata’ so special rules can apply to it during
# package removal.
   installf -c cfgdata $PKGINST $BASEDIR/data/config.data f 444 root
   sys || exit 2
   $BASEDIR/bin/config > $BASEDIR/data/config.data || exit 2
   installf -f -c cfgdata $PKGINST || exit 2
fi
exit 0
```

这里举例说明了一个很少见的实例，其中 `installf` 适用于类操作脚本。由于 `space` 文件用于在特定文件系统上保留空间，因此即使该新文件没有包括在 `pkgmap` 文件中也可以被安全地添加。

`r.cfgdata` 删除脚本

```
# the product manager for this package has suggested that
# the configuration data is so valuable that it should be
# backed up to $PKGSAV before it is removed!
while read path
do
# path names appear in reverse lexical order.
   mv $path $PKGSAV || exit 2
   rm -f $path || exit 2
done
exit 0
```

定义软件包兼容性和相关性

在此案例研究中，软件包使用可选信息文件来定义软件包兼容性和相关性，以及在安装期间呈现版权新息。

技术

此案例研究展示以下技术：

- 使用 `copyright` 文件
- 使用 `compver` 文件
- 使用 `depend` 文件

有关这些文件的更多信息，请参见第 46 页中的 “创建信息文件”。

方法

要符合描述中的要求，您必须：

- 创建 `copyright` 文件。

  `copyright` 文件包含版权信息的 ASCII 文本。样例文件中所示的信息将在软件包安装期间显示在屏幕上。
- 创建 `compver` 文件。

  下图中显示的 `pkginfo` 文件将此软件包版本定义为 3.0 版。`compver` 文件将 3.0 版定义为与 2.3、2.2、2.1、2.1.1、2.1.3 和 1.7 版兼容。
- 创建 `depend` 文件。

  安装软件包时 `depend` 文件中列出的文件必须已经安装在系统上。示例文件有 11 个软件包，安装时这些软件包必须已安装在系统上。

案例研究文件

`pkginfo` 文件

```
PKG=case3
NAME=Case Study #3
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 3.0
CLASSES=none
```

`copyright` 文件

```
Copyright (c) 1999 company_name
All Rights Reserved.
THIS PACKAGE CONTAINS UNPUBLISHED PROPRIETARY SOURCE CODE OF
company_name.
The copyright notice above does not evidence any
actual or intended publication of such source code
```

`compver` 文件

```
Version 3.0
Version 2.3
Version 2.2
```

```
Version 2.1
Version 2.1.1
Version 2.1.3
Version 1.7
```

depend 文件

```
P acu Advanced C Utilities
Issue 4 Version 1
P cc C Programming Language
Issue 4 Version 1
P dfm Directory and File Management Utilities
P ed Editing Utilities
P esg Extended Software Generation Utilities
Issue 4 Version 1
P graph Graphics Utilities
P rfs Remote File Sharing Utilities
Issue 1 Version 1
P rx Remote Execution Utilities
P sgs Software Generation Utilities
Issue 4 Version 1
P shell Shell Programming Utilities
P sys System Header Files
Release 3.1
```

使用标准类和类操作脚本修改文件

在此案例研究中，使用标准类和类操作脚本在软件包安装期间修改现有文件。它使用三种修改方法之一。另外两种方法将在第 100 页中的 “使用 sed 类和 postinstall 脚本修改文件”和第 102 页中的 “使用 build 类修改文件”中进行介绍。修改的文件是 /etc/inittab。

技术

此案例研究展示如何使用安装类操作脚本和删除类操作脚本。有关更多信息，请参见第 62 页中的 “编写类操作脚本”。

方法

要使用类和类操作脚本在安装期间修改 /etc/inittab，您必须完成以下任务：

- 创建一个类。

  创建名为 inittab 的类。必须为该类提供安装和删除类操作脚本。在 pkginfo 文件的 CLASSES 参数中定义 inittab 类。

- 创建一个 inittab 文件。

  此文件包含将添加到 /etc/inittab 的条目的信息。请注意在 prototype 文件图中，inittab 是 inittab 类的成员并且文件类型为 e（表示可编辑）。

- 创建安装类操作脚本 (i.inittab)。

  切记类操作脚本每次执行时都必须产生相同结果。类操作脚本执行以下过程：

  - 检查以前是否已添加该条目
  - 如果是，删除该条目的所有以前版本
  - 编辑 inittab 文件并添加注释行，以便您知道该条目的来源
  - 将临时文件移回 /etc/inittab
  - 当收到 ENDOFCLASS 指示符时，执行 init q 命令

  请注意，此安装脚本可执行 init q 命令。此方法不需要使用只有一行的 postinstall 脚本。

- 创建删除类操作脚本 (r.inittab)。

  删除脚本与安装脚本非常类似。安装脚本添加的信息会被删除，而且会执行 init q 命令。

此案例研究比下一个更为复杂；请参见第 100 页中的 “使用 sed 类和 postinstall 脚本修改文件”。在此案例研究中，不是提供两个文件而是需要三个文件，而且提供的 /etc/inittab 文件实际上只是包含要插入的条目片段的占位符。此占位符可能已被放置到 i.inittab 文件中，除非 pkgadd 命令必须将一个文件传递到 i.inittab 文件。此外，必须将删除过程放置到一个单独的文件 (r.inittab) 中。虽然此方法效果很好，但最好将其保留为在涉及多个文件的非常复杂的安装情况下使用。请参见第 103 页中的 “在安装期间修改 crontab 文件”。

由于 inittab 条目末尾的注释基于软件包实例，因此第 100 页中的 “使用 sed 类和 postinstall 脚本修改文件”中使用的 sed 程序支持多个软件包实例。第 102 页中的 “使用 build 类修改文件”中的案例研究给出了一种在安装期间编辑 /etc/inittab 的更简化的方法。

案例研究文件

pkginfo 文件

```
PKG=case5
NAME=Case Study #5
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=inittab
```

prototype 文件

```
i pkginfo
i i.inittab
i r.inittab
e inittab /etc/inittab ? ? ?
```

i.inittab 安装类操作脚本

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
sed -e "s/$/#$PKGINST" $src >> /tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
if [ "$1" = ENDOFCLASS ]
then
/sbin/init q ||
exit 2
fi
exit 0
```

r.inittab 删除类操作脚本

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
/sbin/init q ||
exit 2
exit 0
```

inittab 文件

```
rb:023456:wait:/usr/robot/bin/setup
```

使用 sed 类和 postinstall 脚本修改文件

此案例研究在软件包安装期间修改安装计算机上存在的一个文件。它使用三种修改方法之一。另外两种方法将在第 97 页中的 “使用标准类和类操作脚本修改文件”和第 102 页中的 “使用 build 类修改文件”中进行介绍。修改的文件是 /etc/inittab。

技术

此案例研究展示以下技术：

- 使用 sed 类

  有关 sed 类的更多信息，请参见第 65 页中的 “sed 类脚本”。
- 使用 postinstall 脚本

  有关此脚本的更多信息，请参见第 60 页中的 “编写过程脚本”。

方法

要使用 sed 类在安装时修改 /etc/inittab，您必须完成以下任务：

- 将 sed 类脚本添加到 prototype 文件。

  脚本的名称必须是将编辑文件的名称。在本例中，将编辑的文件是 /etc/inittab，因此 sed 脚本被命名为 /etc/inittab。对于 sed 脚本的模式、所有者和组没有要求（在样例 prototype 中由问号表示）。sed 脚本的文件类型必须是 e（表示可编辑）。
- 设置 CLASSES 参数以包括 sed 类。

  如示例文件所示，sed 是待安装的唯一类。不过，类的数目可以是任意的。
- 创建 sed 类操作脚本。

  您的软件包不能提供您希望的形式的 /etc/inittab 副本，因为 /etc/inittab 是一个动态文件，您无法知道它在软件包安装时的外观形式。不过，可使用 sed 脚本在软件包安装期间修改 /etc/inittab 文件。
- 创建 postinstall 脚本。

  您需要执行 init q 命令通知系统 /etc/inittab 已被修改。在此示例中，您可以执行该操作的唯一位置是在 postinstall 脚本中。查看 postinstall 示例脚本，您会发现它的唯一作用是执行 init q 命令。

这种在安装期间编辑 /etc/inittab 的方法有一个缺点：虽然只是为了执行 init q 命令也必须提供一个完整的脚本（postinstall 脚本）。

案例研究文件

pkginfo 文件

```
PKG=case4
NAME=Case Study #4
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=sed
```

prototype 文件

```
i pkginfo
i postinstall
e sed /etc/inittab ? ? ?
```

sed 类操作脚本 (/etc/inittab)

```
!remove
# remove all entries from the table that are associated
# with this package, though not necessarily just
# with this package instance
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
!install
# remove any previous entry added to the table
# for this particular change
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
# add the needed entry at the end of the table;
# sed(1) does not properly interpret the '$a'
# construct if you previously deleted the last
# line, so the command
# $a\
# rb:023456:wait:/usr/robot/bin/setup #ROBOT
# will not work here if the file already contained
# the modification. Instead, you will settle for
# inserting the entry before the last line!
$i\
rb:023456:wait:/usr/robot/bin/setup #ROBOT
```

postinstall 脚本

```
# make init re-read inittab
/sbin/init q ||
exit 2
exit 0
```

使用 build 类修改文件

此案例研究在软件包安装期间修改安装计算机上存在的一个文件。它使用三种修改方法之一。另外两种方法将在第 97 页中的 “使用标准类和类操作脚本修改文件”和第 100 页中的 “使用 sed 类和 postinstall 脚本修改文件”中进行介绍。修改的文件是 /etc/inittab。

技术

此案例研究展示如何使用 build 类。有关 build 类的更多信息，请参见第 66 页中的 “build 类脚本”。

方法

这种修改 /etc/inittab 的方法使用 build 类。build 类脚本作为 shell 脚本执行，其输出成为正在执行的文件的新版本。换句话说，与该软件包一起提供的 /etc/inittab 数据文件将会执行，执行后的输出将成为 /etc/inittab。

build 类脚本在软件包安装和删除期间执行。如果该文件在安装时执行，参数 install 会传递给该文件。请注意在 build 样例类脚本中，安装操作通过测试此参数来定义。

要使用 build 类编辑 /etc/inittab，您必须完成以下任务：

- 在 prototype 文件中定义生成文件。

  prototype 文件中的生成文件条目应将该文件放置在 build 类中，并将其文件类型定义为 e。请确保将 pkginfo 文件中的 CLASSES 参数定义为 build。

- 创建 build 类脚本。

  build 样例类脚本执行以下过程：

  - 编辑 /etc/inittab 文件以删除对该软件包的任何现有更改。请注意，文件名 /etc/inittab 已被硬编码到 sed 命令中。
  - 如果正在安装软件包，将新行添加到 /etc/inittab 的末尾。一个注释标记会包括在该新条目中，用于描述该条目的来源。
  - 执行 init q 命令。

该解决方案解决了在第 97 页中的 “使用标准类和类操作脚本修改文件”和第 100 页中的 “使用 sed 类和 postinstall 脚本修改文件”的案例研究中描述的缺点。只需要一个很短的文件（除 pkginfo 和 prototype 文件之外）。因为使用了 PKGINST 参数，所以该文件适用于软件包的多个实例，而且由于 init q 命令可从 build 类脚本中执行，因此不需要 postinstall 脚本。

案例研究文件

pkginfo 文件

```
PKG=case6
NAME=Case Study #6
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=build
```

prototype 文件

```
i pkginfo
e build /etc/inittab ? ? ?
```

生成文件

```
# PKGINST parameter provided by installation service
# remove all entries from the existing table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" /etc/inittab ||
exit 2
if [ "$1" = install ]
then
# add the following entry to the table
echo "rb:023456:wait:/usr/robot/bin/setup #$PKGINST" ||
exit 2
fi
/sbin/init q ||
exit 2
exit 0
```

在安装期间修改 crontab 文件

此案例研究在软件包安装期间修改 crontab 文件。

技术

此案例研究展示以下技术：

- 使用类和类操作脚本
  有关更多信息，请参见第 62 页中的 “编写类操作脚本”。

- 在类操作脚本中使用 crontab 命令。

方法

在安装期间编辑多个文件的最高效方式是定义一个类，然后提供类操作脚本。如果您使用了 build 类方法，您将需要为每个编辑的 crontab 文件提供一个 build 类脚本。定义 cron 类是一种更通用的方法。要使用这种方法编辑 crontab 文件，您必须：

- 在 prototype 文件中定义将编辑的 crontab 文件。

  在 prototype 文件中为每个将编辑的 crontab 文件创建一个条目。对于每个文件，将类定义为 cron 并将文件类型定义为 e。使用将编辑的文件的实际名称。
- 为软件包创建 crontab 文件。

  这些文件包含您希望添加到同名的现有 crontab 文件的信息。
- 为 cron 类创建安装类操作脚本。

  i.cron 样例脚本执行以下过程：
  - 确定用户 ID (user ID, UID)。

    i.cron 脚本将 *user* 变量设置为正在处理的 cron 类脚本的基名。该名称是 UID。例如，/var/spool/cron/crontabs/root 的基名是 root，该基名也是 UID。
  - 使用 UID 和 -l 选项执行 crontab。

    使用 -l 选项会告诉 crontab 为定义的用户将 crontab 文件内容发送到标准输出。
  - 将 crontab 命令的输出通过管道输出到 sed 脚本，该脚本用于删除以前使用该安装技术添加的所有条目。
  - 将编辑的输出放置到临时文件中。
  - 将 root UID 对应的数据文件（随软件包提供）添加到临时文件，并添加一个标记，以便您知道这些条目的来源。
  - 使用同一 UID 执行 crontab，并使用临时文件作为其输入。
- 为 cron 类创建删除类操作脚本。

  r.cron 脚本与安装脚本基本相同，只不过前者没有用于向 crontab 文件添加信息的过程。

  这些过程针对 cron 类中的每个文件执行。

案例研究文件

下述 i.cron 和 r.cron 脚本由超级用户执行。以超级用户身份编辑其他用户的 crontab 文件可能导致无法预测的结果。如果需要，请在每个脚本中将以下条目：

```
crontab $user < /tmp/$$crontab ||
```

更改为

```
su $user -c "crontab /tmp/$$crontab" ||
```

pkginfo 命令

```
PKG=case7
NAME=Case Study #7
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1.0
CLASSES=cron
```

prototype 文件

```
i pkginfo
i i.cron
i r.cron
e cron /var/spool/cron/crontabs/root ? ? ?
e cron /var/spool/cron/crontabs/sys ? ? ?
```

i.cron 安装类操作脚本

```
# PKGINST parameter provided by installation service
while read src dest
do
user=‘basename $dest‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
sed -e "s/$/#$PKGINST/" $src >> /tmp/$$crontab ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit 0
```

r.cron 删除类操作脚本

```
# PKGINST parameter provided by installation service
while read path
do
user=‘basename $path‘ ||
exit 2
(crontab -l $user |
```

```
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit
```

crontab 文件 #1

```
41,1,21 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /usr/bin/su uucp -c
"/usr/lib/uucp/uudemon.cleanup" >
/dev/null 2>&1
11,31,51 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

crontab 文件 #2

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

注 – 如果对一组文件的编辑将导致文件总大小增加超过 10K，请提供一个 space 文件，以便 pkgadd 命令可以允许这种增加。有关 space 文件的更多信息，请参见第 50 页中的 “在目标系统上保留额外空间”。

使用过程脚本安装和删除驱动程序

此软件包可安装驱动程序。

技术

此案例研究展示以下技术：

- 使用 postinstall 脚本安装和装入驱动程序
- 使用 preremove 脚本卸载驱动程序

有关这些脚本的更多信息，请参见第 60 页中的 “编写过程脚本”。

方法

- 创建 request 脚本。

request 脚本通过询问管理员并将答复指定给 $KERNDIR 参数，确定管理员希望将驱动程序对象安装到的位置。

该脚本以一个例程结束，以使两个参数（即，CLASSES 和 KERNDIR）对于安装环境和 postinstall 脚本可用。

- 创建 postinstall 脚本。

  postinstall 脚本实际上执行驱动程序安装。该脚本在 buffer 和 buffer.conf 两个文件安装后执行。在此示例中显示的 postinstall 文件执行以下操作：

  - 使用 add_drv 命令将驱动程序装入到系统。
  - 使用 installf 命令为设备创建链接。
  - 使用 installf -f 命令完成安装。
  - 创建 preremove 脚本。

    preremove 脚本使用 rem_drv 命令从系统卸载驱动程序，然后删除链接 /dev/buffer0。

案例研究文件

pkginfo 文件

```
PKG=bufdev
NAME=Buffer Device
CATEGORY=system
BASEDIR=/
ARCH=INTEL
VERSION=Software Issue #19
CLASSES=none
```

prototype 文件

要在安装时安装驱动程序，您必须在 prototype 文件中包括驱动程序的对象和配置文件。

在此示例中，驱动程序的可执行模块被命名为 buffer；add_drv 命令处理该文件。内核使用配置文件 buffer.conf 帮助配置驱动程序。

```
i pkginfo
i request
i postinstall
i preremove
f none $KERNDIR/buffer 444 root root
f none $KERNDIR/buffer.conf 444 root root
```

查看此示例中的 prototype 文件，请注意以下事项：

- 因为不需要对软件包对象进行特殊处理，所以您可以将其放置在 none 标准类中。在 pkginfo 文件中 CLASSES 参数被设置为 none。
- buffer 和 buffer.conf 的路径名以变量 $KERNDIR 开头。此变量在 request 脚本中设置，允许管理员决定驱动程序文件的安装位置。缺省目录是 /kernel/drv。
- postinstall 脚本（将执行驱动程序安装的脚本）有一个对应条目。

request 脚本

```
trap 'exit 3' 15
# determine where driver object should be placed; location
# must be an absolute path name that is an existing directory
KERNDIR=`ckpath -aoy -d /kernel/drv -p \
"Where do you want the driver object installed"` || exit $?

# make parameters available to installation service, and
# so to any other packaging scripts
cat >$1 <<!

CLASSES='$CLASSES'
KERNDIR='$KERNDIR'
!
exit 0
```

postinstall 脚本

```
# KERNDIR parameter provided by `request' script
err_code=1                    # an error is considered fatal
# Load the module into the system
cd $KERNDIR
add_drv -m '* 0666 root sys' buffer || exit $err_code
# Create a /dev entry for the character node
installf $PKGINST /dev/buffer0=/devices/eisa/buffer*:0 s
installf -f $PKGINST
```

preremove 脚本

```
err_code=1                    # an error is considered fatal
# Unload the driver
rem_drv buffer || exit $err_code
# remove /dev file
removef $PKGINST /dev/buffer0 ; rm /dev/buffer0
removef -f $PKGINST
```

使用 sed 类和过程脚本安装驱动程序

此案例研究描述如何使用 sed 类和过程脚本安装驱动程序。该案例研究还与前面的案例研究（请参见第 106 页中的 “使用过程脚本安装和删除驱动程序”）有所不同，因为此软件包由绝对和可重定位的对象组成。

技术

此案例研究展示以下技术：

- 使用绝对和可重定位的对象生成 prototype 文件。

  有关生成 prototype 文件的更多信息，请参见第 28 页中的 “创建 prototype 文件”。
- 使用 postinstall 脚本

  有关此脚本的更多信息，请参见第 60 页中的 “编写过程脚本”。
- 使用 preremove 脚本

  有关此脚本的更多信息，请参见第 60 页中的 “编写过程脚本”。
- 使用 copyright 文件

  有关此文件的更多信息，请参见第 49 页中的 “编写版权信息”。

方法

- 创建包含绝对和可重定位的软件包对象的 prototype 文件。

  将会在第 110 页中的 “prototype 文件”中详细讨论此内容。
- 将 sed 类脚本添加到 prototype 文件。

  脚本的名称必须是将编辑文件的名称。在本例中，将编辑的文件是 /etc/devlink.tab，因此 sed 脚本被命名为 /etc/devlink.tab。对于 sed 脚本的模式、所有者和组没有要求（在样例 prototype 中由问号表示）。sed 脚本的文件类型必须是 e（表示可编辑）。
- 设置 CLASSES 参数以包括 sed 类。
- 创建 sed 类操作脚本 (/etc/devlink.tab)。
- 创建 postinstall 脚本。

  postinstall 脚本需要执行 add_drv 命令以向系统添加设备驱动程序。
- 创建 preremove 脚本。

  preremove 脚本需要执行 rem_drv 命令以在删除软件包之前从系统删除设备驱动程序。
- 创建 copyright 文件。

copyright 文件包含版权信息的 ASCII 文本。样例文件中所示的信息将在软件包安装期间显示在屏幕上。

案例研究文件

pkginfo 文件

```
PKG=SUNWsst
NAME=Simple SCSI Target Driver
VERSION=1
CATEGORY=system
ARCH=sparc
VENDOR=Sun Microsystems
BASEDIR=/opt
CLASSES=sed
```

prototype 文件

例如，此案例研究使用软件包对象的分层布局，如下图所示。

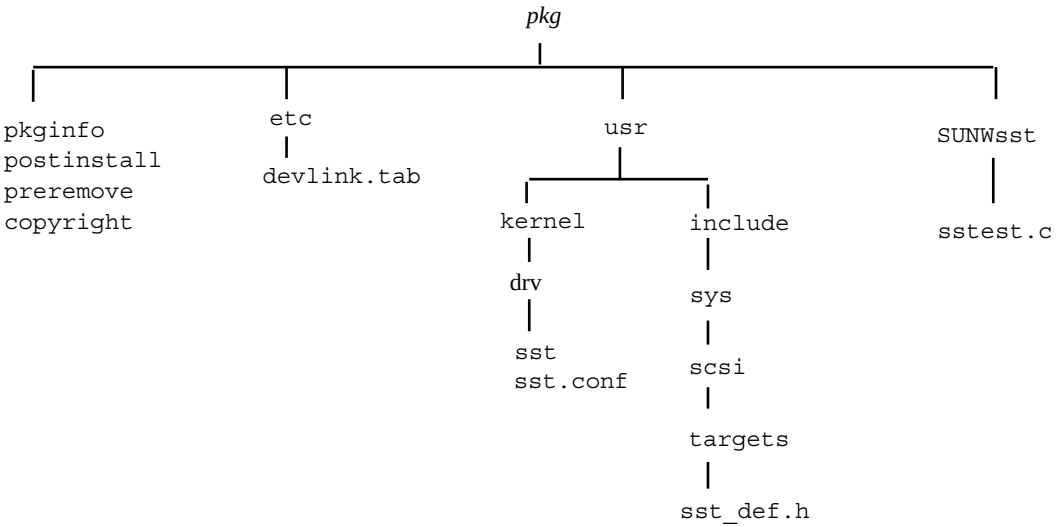


图 5–1 分层软件包目录结构

软件包对象的安装位置与上图 pkg 目录中的位置相同。驱动程序模块（sst 和 sst.conf）安装在 /usr/kernel/drv 中，而头文件安装在

/usr/include/sys/scsi/targets 中。sst、sst.conf 和 sst_def.h 文件是绝对对象。测试程序 sstest.c 及其目录 SUNWsst 是可重定位对象；它们的安装位置由 BASEDIR 参数设置。

软件包的其余组件（所有控制文件）安装在开发计算机上软件包的顶层目录中，但 sed 类脚本除外。它根据所修改的文件命名为 devlink.tab 并安装到 etc，该目录包含实际的 devlink.tab 文件。

从 pkg 目录运行 pkgproto 命令，如下所示：

```
find usr SUNWsst -print | pkgproto > prototype
```

上述命令的输出如下所示：

```
d none usr 0775 pms mts
d none usr/include 0775 pms mts
d none usr/include/sys 0775 pms mts
d none usr/include/sys/scsi 0775 pms mts
d none usr/include/sys/scsi/targets 0775 pms mts
f none usr/include/sys/scsi/targets/sst_def.h 0444 pms mts
d none usr/kernel 0775 pms mts
d none usr/kernel/drv 0775 pms mts
f none usr/kernel/drv/sst 0664 pms mts
f none usr/kernel/drv/sst.conf 0444 pms mts
d none SUNWsst 0775 pms mts
f none SUNWsst/sstest.c 0664 pms mts
```

此 prototype 文件并不是完整文件。要完成此文件，您需要进行以下修改：

- 插入控制文件（文件类型 i）条目，因为它们与其他软件包对象的格式不同。
- 删除目标系统上已存在的目录的条目。
- 更改每个条目的访问权限和拥有权。
- 在绝对软件包对象之前添加一个斜线。

以下是最终的 prototype 文件：

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin
f none /usr/kernel/drv/sst 0755 root sys
f none /usr/kernel/drv/sst.conf 0644 root sys
d none SUNWsst 0775 root sys
f none SUNWsst/sstest.c 0664 root sys
```

sed 脚本条目中的问号表明不应更改安装计算机上现有文件的访问权限和拥有权。

sed 类操作脚本 (/etc/devlink.tab)

在驱动程序示例中，sed 类脚本用于向 /etc/devlink.tab 文件中添加一个驱动程序条目。devlinks 命令使用此文件创建从 /dev 到 /devices 的符号链接。以下是 sed 脚本：

```
# sed class script to modify /etc/devlink.tab
!install
/name=sst;/d
$i\
type=ddi_pseudo;name=sst;minor=character    rsst\\A1

!remove
/name=sst;/d
```

pkgrm 命令不运行该脚本的删除部分。您可能需要向 preremove 脚本中添加一行，以便直接运行 sed 从 /etc/devlink.tab 文件删除条目。

postinstall 安装脚本

在此示例中，该脚本所需要做的只是运行 add_drv 命令。

```
# Postinstallation script for SUNWsst
# This does not apply to a client.
if [$PKG_INSTALL_ROOT = "/" -o -z $PKG_INSTALL_ROOT]; then
   SAVEBASE=$BASEDIR
   BASEDIR=””; export BASEDIR
   /usr/sbin/add_drv sst
   STATUS=$?
   BASEDIR=$SAVEBASE; export BASEDIR
   if [ $STATUS -eq 0 ]
   then
         exit 20
   else
         exit 2
   fi
else
   echo "This cannot be installed onto a client."
   exit 2
fi
```

add_drv 命令使用 BASEDIR 参数，因此该脚本必须在运行此命令之前取消设置 BASEDIR，并在以后恢复它。

add_drv 命令的操作之一是运行 devlinks，它使用由 sed 类脚本放置在 /etc/devlink.tab 中的条目为驱动程序创建 /dev 条目。

postinstall 脚本的退出代码作用很重要。退出代码 20 告诉 pkgadd 命令告知用户重新引导系统（安装驱动程序后必须这样做），而退出代码 2 则告诉 pkgadd 命令告知用户安装部分失败。

preremove 删除脚本

在此驱动程序示例中，该脚本删除 /dev 中的链接并对驱动程序运行 rem_drv 命令。

```
# Pre removal script for the sst driver
echo “Removing /dev entries”
/usr/bin/rm -f /dev/rsst*

echo “Deinstalling driver from the kernel”
SAVEBASE=$BASEDIR
BASEDIR=””; export BASEDIR
/usr/sbin/rem_drv sst
BASEDIR=$SAVEBASE; export BASEDIR

exit
```

该脚本删除 /dev 条目本身； /devices 条目由 rem_drv 命令删除。

copyright 文件

以下是一个包含版权声明文本的简单 ASCII 文件。该声明在软件包安装开始时显示，显示时与此文件中的形式完全相同。

```
    Copyright (c) 1999 Drivers-R-Us, Inc.
    10 Device Drive, Thebus, IO 80586

All rights reserved. This product and related documentation is
protected by copyright and distributed under licenses
restricting its use, copying, distribution and decompilation.
No part of this product or related documentation may be
reproduced in any form by any means without prior written
authorization of Drivers-R-Us and its licensors, if any.
```

◆ ◆ ◆ 第 6 章

6 创建软件包的高级技术

Solaris OS 中实现的 System V 打包的完整功能提供了一个用于安装软件产品的强大工具。作为软件包设计者，您可以利用这些功能。不属于 Solaris OS 的软件包（非随附软件包）可以使用类机制来定制服务器/客户机安装。可针对管理员的需要设计可重定位的软件包。复杂产品可以通过能够自动解决软件包相关性问题的复合软件包集合形式提供。升级和修补可由软件包设计者定制。修补的软件包可以按照与未修补的软件包相同的方式提供，并且产品中还可以包括回退归档。

以下是本章中概述信息的列表。

指定基目录

您可以使用多种方法指定软件包的安装位置，而且很重要的一点是能够在安装时动态更改安装基本位置。如果正确实现了此功能，管理员便可以轻松安装多个版本和多个体系结构。

本节首先讨论常用方法，然后讨论能够改进异构系统上的安装的途径。

缺省管理文件

负责安装软件包的管理员可以使用管理文件来控制软件包安装。然而，作为软件包设计者，您需要了解有关管理文件的信息，并了解管理员如何能够改变您计划的软件包安装。

管理文件告诉 pkgadd 命令是否执行它通常执行的任何检查或提示。因此，管理员在使用管理文件之前应该充分了解软件包的安装过程和涉及的脚本。

缺省基本管理文件随 SunOS 操作系统一起提供，位于 /var/sadm/install/admin/default 中。该文件建立了与软件产品安装有关的最基本级别的管理策略。随操作系统一起提供的该文件如下所示：

```
#ident "@(#)default
1.4 92/12/23 SMI"    /* SVr4.0 1.5.2.1    */
mail=
instance=unique
partial=ask
runlevel=ask
idepend=ask
rdepend=ask
space=ask
setuid=ask
conflict=ask
action=ask
basedir=default
```

管理员可以编辑此文件以建立新的缺省行为，或者创建一个不同的管理文件并使用 pkgadd 命令的 -a 选项指定其存在。

在一个管理文件中可以定义 11 个参数，但并非所有参数都必须定义。有关更多信息，请参见 admin(4)。

basedir 参数指定在安装软件包时如何派生基目录。大多数管理员都将此参数保留为 default，但可以将 basedir 设置为以下值之一：

- ask，表示始终要求管理员提供基目录
- 绝对路径名
- 包含 $PKGINST 构造的绝对路径名，表示始终安装到派生于软件包实例的基目录

注 – 如果带有参数 -a none 调用了 pkgadd 命令，则该命令始终要求管理员提供基目录。遗憾的是，这还会将文件中的**所有**参数设置为缺省值 quit，而这可能会导致其他问题。

适应不确定性

管理员可使用管理文件来控制系统上正安装的所有软件包。遗憾的是，**软件包设计者**经常提供一个替代的缺省管理文件，而没有考虑管理员的愿望。

软件包设计者有时会提供一个替代管理文件，以便他们自己（而不是管理员）能够控制软件包的安装。由于缺省管理文件中的 basedir 条目会覆盖所有其他基目录，因此它提供了一种在安装时选择适当基目录的简单方法。在早于 Solaris 2.5 发行版的所有 Solaris OS 版本中，这种方法被认为是控制基目录的最简单方法。

然而，您必须接受管理员的有关产品安装的希望。提供一个临时的缺省管理文件以便控制安装会导致管理员觉得不受信任。您应该使用 request 脚本和 checkinstall 脚本在管理员的监督下控制这些安装。如果 request 脚本如实地使管理员参与安装过程，System V 打包过程将同时满足管理员和软件包设计者的需求。

使用 BASEDIR 参数

pkginfo 文件都必须以如下所示的条目形式包括一个缺省基目录：

BASEDIR=*absolute_path*

这只是缺省基目录，可由管理员在安装期间更改。

尽管某些软件包需要多个基目录，但使用此参数定位软件包的好处是，当安装开始的时候，能够保证基目录作为一个有效的目录就位并可写。服务器和客户机的基目录的正确路径以保留环境变量的形式供所有过程脚本使用，并且 pkginfo -r SUNWstuf 命令显示软件包的当前安装基本位置。

在 checkinstall 脚本中，BASEDIR 即是 pkginfo 文件中定义的该参数（该参数尚未根据条件赋值）。为了检查目标基目录，需要使用 $ {PKG_INSTALL_ROOT} $BASEDIR 构造。这意味着 request 或 checkinstall 脚本可以在安装环境下更改 BASEDIR 的值，并且可以获得可预测的结果。当调用 preinstall 脚本时，BASEDIR 参数是指向目标系统上实际基目录的完全根据条件设置的指针，即使该系统是客户机也是如此。

注 – 对于不同的 SunOS 操作系统发行版，request 脚本以不同方式利用 BASEDIR 参数。为了在 request 脚本中测试 BASEDIR 参数，应该使用下面的代码来确定正在使用的实际基目录。

```
# request script
constructs base directory
if [ ${CLIENT_BASEDIR} ]; then
      LOCAL_BASE=$BASEDIR
else
      LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi
```

使用参数化基目录

如果一个软件包需要多个基目录，您可以使用参数化路径名建立这些目录。此方法已经相当流行，尽管它具有以下缺点。

- 具有参数化路径名的软件包的行为方式通常类似于绝对软件包，但 pkgadd 命令会像处理可重定位的软件包一样处理该软件包。必须定义 BASEDIR 参数（即使没有使用）。

- 管理员无法使用 System V 实用程序确定软件包的安装基本位置（`pkginfo -r` 命令不起作用）。
- 管理员无法使用既定的方法重定位软件包（它称为可重定位的软件包，但是其行为方式与绝对软件包相同）。
- 多体系结构或多版本的安装要求对每个目标基目录进行意外事件计划，这通常意味着使用多个复杂的类操作脚本。

尽管确定基目录的参数是在 `pkginfo` 文件中定义的，但可以被 `request` 脚本修改。这是此方法广受欢迎的主要原因之一。然而，此方法的缺点会长期存在，您应该在迫不得已的情况下才考虑使用此配置。

示例－使用参数化基目录

pkginfo 文件

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
EZDIR=/usr/stuf/EZstuf
HRDDIR=/opt/SUNWstuf/HRDstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert980707141632
```

pkgmap 文件

```
: 1 1758
1 d none $EZDIR 0775 root bin
1 f none $EZDIR/dirdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/usrdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/filedel 0555 bin bin 40 773 751310229
1 d none $HRDDIR 0775 root bin
1 f none $HRDDIR/mksmart 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mktall 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkcute 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
```

```
1 f none /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

管理基目录

如有必要，任何具有多个版本或多种体系结构的软件包都应该设计为**遍历**基目录。遍历基目录意味着如果基目录中已经存在要安装的软件包的以前版本或不同体系结构，则要安装的软件包会解决此问题，方法可能是使用稍微不同的名称创建一个新的基目录。Solaris 2.5 和兼容发行版中的 `request` 和 `checkinstall` 脚本能够修改 `BASEDIR` 环境变量。对于 Solaris OS 的任何更早版本，情况则不是这样的。

即使在 Solaris OS 的较早版本中，`request` 脚本也有权在安装基本位置中重新定义目录。`request` 脚本可以采用仍然支持多数管理首选项的方法完成此操作。

适应重定位

尽管您可以为各种软件包选择能够保证对某个体系结构和版本唯一的基目录，但这会导致目录分层结构中存在多余的级别。例如，对于为基于 SPARC 和 x86 的处理器设计的产品，您可能会根据处理器和版本组织基目录，如下所示。

基目录	版本和处理器
`/opt/SUNWstuf/sparc/1.0`	1.0 版，SPARC
`/opt/SUNWstuf/sparc/1.2`	1.2 版，SPARC
`/opt/SUNWstuf/x86/1.0`	1.0 版，x86

这样做完全可以而且有效，但是您处理名称和编号的方式就好像它们对管理员有意义一样。一种更好的方法是在向管理员进行解释并获取权限**之后**自动完成此工作。

这意味着您可以在软件包中完成全部工作，而无需要求管理员手动完成。您可以在 `postinstall` 脚本中任意指定基目录，然后透明地建立适当的客户机链接。您还可以在 `postinstall` 脚本中使用 `pkgadd` 命令在客户机上安装软件包的全部或一部分。您甚至可以咨询管理员哪些用户或客户机需要了解此软件包，并自动更新 `PATH` 环境变量和 `/etc` 文件。只要软件包在安装时执行的所有操作能够在删除时撤消，那么该方法是完全可以接受的。

遍历基目录

您可以利用两种方法在安装时控制基目录。第一种方法最适合于仅能安装在 Solaris 2.5 和兼容发行版上的新软件包；它为管理员提供非常有用的数据，支持多个已安装的版本和体系结构，而只需要完成最少的特殊工作。第二种方法可由任何软件包使用，并且利用 request 脚本所固有的对于生成参数的控制来确保安装成功。

使用 BASEDIR 参数

checkinstall 脚本可以在安装时选择适当的基目录，这意味着可以将基目录放置在目录树中非常低的位置。此示例按顺序递增基目录，从而得到如下形式的目录：/opt/SUNWstuf、/opt/SUNWstuf.1 和 /opt/SUNWstuf.2。管理员可以使用 pkginfo 命令确定在每个基目录中所安装的体系结构和版本。

如果 SUNWstuf 软件包（包含一组执行具体工作的实用程序）使用此方法，则它的 pkginfo 和 pkgmap 文件将如下所示。

pkginfo 文件

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt/SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

pkgmap 文件

```
: 1 1758
1 d none EZstuf 0775 root bin
1 f none EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none HRDstuf 0775 root bin
1 f none HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
```

```
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

示例－遍历 BASEDIR 的分析脚本

假定 SUNWstuf 的 x86 版本已经安装在服务器上的 /opt/SUNWstuf 中。当管理员使用 pkgadd 命令安装 SPARC 版本时， request 脚本需要检测是否存在 x86 版本并且与管理员就安装问题进行交互。

注 – 可在 checkinstall 脚本中遍历基目录而无需管理员交互，但是，如果此类任意操作的发生过于频繁，管理员会在此过程中失去信心。

软件包的用于处理此情形的 request 脚本和 checkinstall 脚本可能如下所示。

request 脚本

```
# request script
for SUNWstuf to walk the BASEDIR parameter.

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

GENMSG="The base directory $LOCAL_BASE already contains a \
different architecture or version of $PKG."

OLDMSG="If the option \"-a none\" was used, press the  \
key and enter an unused base directory when it is requested."

OLDPROMPT="Do you want to overwrite this version? "

OLDHELP="\"y\" will replace the installed package, \"n\" will \
stop the installation."

SUSPEND="Suspending installation at user request using error \
code 1."

MSG="This package could be installed at the unused base directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed base directory? "
```

```
HELP="A response of \"y\" will install to the proposed directory and continue,
\"n\" will request a different directory. If the option \"-a none\" was used,
press the  key and enter an unused base directory when it is requested."

DIRPROMPT="Select a preferred base directory ($WRKNG_BASE) "

DIRHELP="The package $PKG will be installed at the location entered."

NUBD_MSG="The base directory has changed. Be sure to update \
any applicable search paths with the actual location of the \
binaries which are at $WRKNG_BASE/EZstuf and $WRKNG_BASE/HRDstuf."

OldSolaris=""
Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0
      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older version it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR
            OldSolaris="true"
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi

WRKNG_BASE=$LOCAL_BASE

    # See if the base directory is already in place and walk it if
    # possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
```

```
                # So this is a different architecture or
                # version than what is already there.
                # Walk the base directory
                Suffix=‘expr $Suffix + 1‘
                WRKNG_BASE=$LOCAL_BASE.$Suffix
                Changed="true"
        fi
done

    # So now we can propose a base directory that isn’t claimed by
    # any of our other versions.
if [ $Changed ]; then
        puttext "$GENMSG"
        if [ $OldSolaris ]; then
              puttext "$OLDMSG"
              result=‘ckyorn -Q -d "a" -h "$OLDHELP" -p "$OLDPROMPT"‘
              if [ $result="n" ]; then
                    puttext "$SUSPEND"
                    exit 1    # suspend installation
              else
                    exit 0
              fi
        else    # The latest functionality is available
              puttext "$MSG"
              result=‘ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"‘
              if [ $? -eq 3]; then
                    echo quitinstall >> $1
                    exit 0
              fi

              if [ $result="n" ]; then
                    WRKNG_BASE=‘ckpath -ayw -d "$WRKNG_BASE" \
                    -h "$DIRHELP" -p "$DIRPROMPT"‘
              else if [ $result="a" ]
                    exit 0
              fi
           fi
           echo "BASEDIR=$WRKNG_BASE" >> $1
           puttext "$NUBD_MSG"
      fi
fi
exit 0
```

checkinstall脚本

```
# checkinstall
script for SUNWstuf to politely suspend
```

```
grep quitinstall $1
if [ $? -eq 0 ]; then
    exit 3        # politely suspend installation
fi

exit 0
```

如果基目录仅仅是 /opt，则此方法不会非常有效。因为 /opt 难以遍历，所以此软件包必须更加准确地调用 BASEDIR。实际上，根据挂载方案的不同，可能无法实现这一点。此示例通过在 /opt 下创建一个新目录来遍历基目录，这种方法不会导致任何问题。

此示例使用 request 脚本和 checkinstall 脚本，即使 Solaris 2.5 发行版以前的版本无法运行 checkinstall 脚本。此示例中的 checkinstall 脚本用于正常停止安装，以响应一个采用 quitinstall 字符串形式的专用消息。如果此脚本在 Solaris 2.3 发行版中执行，checkinstall 脚本将被忽略，而 request 脚本将停止安装并显示错误消息。

请记住，在 Solaris 2.5 和兼容发行版之前，BASEDIR 参数是一个只读参数，不能通过 request 脚本更改。因此，如果检测到 SunOS 操作系统的早期版本（通过测试 CLIENT_BASEDIR 条件环境变量），request 脚本只有两种选择－继续或退出。

使用相对参数化路径

如果您的软件产品可能安装在 SunOS 操作系统的早期版本上，则 request 脚本需要完成所有必要的工作。还可以使用此方法处理多个目录。如果需要使用额外的目录，仍然需要将这些包括在单个基目录下，以便提供易于管理的产品。尽管 BASEDIR 参数没有提供最新 Solaris 发行版中的粒度级别，您的软件包仍然可以使用 request 脚本来处理参数化路径，从而遍历基目录。pkginfo 和 pkgmap 文件可能如下所示。

pkginfo 文件

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
SUBBASE=SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

pkgmap 文件

```
: 1 1758
1 d none $SUBBASE/EZstuf 0775 root bin
1 f none $SUBBASE/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none $SUBBASE/HRDstuf 0775 root bin
1 f none $SUBBASE/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

此示例并不完善。`pkginfo -r` 命令为安装基本位置返回 `/opt`，其含义相当模糊。许多软件包位于 `/opt` 中，但起码它是一个有意义的目录。就像上一个示例一样，接下来的这个示例完全支持多个体系结构和版本。`request` 脚本可根据特定软件包的需要进行调整，并可解析任何适用的相关性。

示例－遍历相对参数化路径的 request 脚本

```
# request script
for SUNWstuf to walk a parametric path

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

MSG="The target directory $LOCAL_BASE already contains \
different architecture or version of $PKG. This package \
could be installed at the unused target directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed directory? "

HELP="A response of \"y\" will install to the proposed directory \
and continue, \"n\" will request a different directory. If \
the option \"-a none\" was used, press the <RETURN> key and \
enter an unused base directory when it is requested."

DIRPROMPT="Select a relative target directory under $BASEDIR/"
```

```
DIRHELP="The package $PKG will be installed at the location entered."

SUSPEND="Suspending installation at user request using error \
code 1."

NUBD_MSG="The location of this package is not the default. Be \
sure to update any applicable search paths with the actual \
location of the binaries which are at $WRKNG_BASE/EZstuf \
and $WRKNG_BASE/HRDstuf."

Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0

      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older versions it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR/$SUBBASE
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR/$SUBBASE
      fi

WRKNG_BASE=$LOCAL_BASE

# See if the base directory is already in place and walk it if
# possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
```

```
                Suffix=‘expr $Suffix + 1‘
                WRKNG_BASE=$LOCAL_BASE.$Suffix
                Changed="true"
        fi
done

# So now we can propose a base directory that isn’t claimed by
# any of our other versions.
if [ $Changed ]; then
        puttext "$MSG"
        result=‘ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"‘
        if [ $? -eq 3 ]; then
                puttext "$SUSPEND"
                exit 1
        fi

        if [ $result="n" ]; then
                WRKNG_BASE=‘ckpath -lyw -d "$WRKNG_BASE" -h "$DIRHELP" \
                -p "$DIRPROMPT"‘

           elif [ $result="a" ]; then
                  exit 0
           else
                  exit 1
           fi
           echo SUBBASE=$SUBBASE.$Suffix >> $1
           puttext "$NUBD_MSG"
      fi
fi
exit 0
```

在异构环境中支持重定位

System V 打包背后的原始概念假定每个系统有一个体系结构。服务器的概念在设计中没有发挥作用。当然，现在一台服务器可以支持多个体系结构，这意味着在一台服务器上可能存在同一个软件的多个副本，每个副本适用于不同的体系结构。尽管 Solaris 软件包被隔离在建议的文件系统边界（例如，/ 和 /usr）内，但由于在服务器以及每个客户机上都有产品数据库，因此并非所有安装都一定支持这种划分。某些实现支持完全不同的结构，并且隐含一个公共产品数据库。尽管将客户机指向不同的版本是一项非常简单的工作，但将 System V 软件包实际安装到不同的基目录可能给管理员带来困难。

当您设计软件包时，还应该考虑管理员用来引入新的软件版本的常用方法。管理员通常寻求并行安装和测试最新版本与当前安装的版本。该过程涉及到将新版本安装到与当前版本不同的基目录，以及将一些非关键性客户机定向到新版本以便进行测试。随着信心不断增加，管理员将越来越多的客户机重定向到新版本。最终，管理员仅仅为了应对紧急情况而保留旧版本，并最终将其删除。

这意味着以现代异构系统为目标的软件包必须支持真正的重定位：即管理员可以将这些软件包放置到文件系统中的任意合理位置，而仍然可以使用其完整的功能。Solaris 2.5 和兼容发行版提供了许多有用的工具，允许将多个体系结构和版本完全安装到同一个系统中。Solaris 2.4 和兼容发行版也支持真正的重定位，但完成该任务的方法不是很明显。

传统方法

可重定位软件包

System V ABI 表明，可重定位软件包背后的原始意图是使安装软件包对于管理员而言更加便利。现在，对可重定位软件包的需求更进了一大步。便利与否不是唯一的问题，更有可能发生的问题是在安装期间，一个活动的软件产品已经安装在缺省目录中。不能处理这种情况的软件包会覆写现有产品，或者安装失败。然而，可处理多个体系结构和多个版本的软件包可以顺利安装，并且为管理员提供与现有管理传统完全兼容的大量选项。

在某些方面，多个体系结构的问题和多个版本的问题是同一问题。必须可以并行安装现有软件包的变体与其他变体，并且能够在不影响功能的情况下将客户机或导出的文件系统的独立使用者定向到其中任一变体。尽管 Sun 已经制定了在服务器上处理多个体系结构的方法，但管理员可以不遵守这些推荐方法。所有软件包都必须能够符合管理员的合理安装期望。

示例－传统的可重定位软件包

此示例演示一个传统的可重定位软件包内容。该软件包将位于 `/opt/SUNWstuf` 中，其 `pkginfo` 文件和 `pkgmap` 文件可能如下所示。

`pkginfo` 文件

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

pkgmap 文件

```
: 1 1758
1 d none SUNWstuf 0775 root bin
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

因为每个软件包对象都会安装到 pkginfo 文件中的 BASEDIR 参数所定义的基目录，所以这称为传统方法。例如，pkgmap 文件中的第一个对象被安装到目录 /opt/SUNWstuf。

绝对软件包

绝对软件包是安装到特定根 (/) 文件系统的软件包。这些软件包难以从多个版本和体系结构的角度进行处理。通常，所有软件包都应该是可重定位的。然而，有非常充足的理由在可重定位软件包中包括绝对元素。

示例－传统绝对软件包

如果 SUNWstuf 软件包是绝对软件包，则不应该在 pkginfo 文件中定义 BASEDIR 参数，而 pkgmap 文件将如下所示。

pkgmap 文件

```
: 1 1758
1 d none /opt ? ? ?
1 d none /opt/SUNWstuf 0775 root bin
1 d none /opt/SUNWstuf/EZstuf 0775 root bin
1 f none /opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none /opt/SUNWstuf/HRDstuf 0775 root bin
1 f none /opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
```

```
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

在此示例中，如果管理员在安装期间指定了一个备用基目录，pkgadd 命令将忽略该目录。此软件包总是安装到目标系统的 /opt/SUNWstuf。

pkgadd 命令的 -R 参数按预期方式工作。例如，

```
pkgadd -d . -R /export/opt/client3 SUNWstuf
```

将对象安装在 /export/opt/client3/opt/SUNWstuf 中；但这是此软件包最接近可重定位软件包之处。

请注意，在 pkgmap 文件中，对 /opt 目录使用了问号 (?)。这表明不应该更改现有属性。这并不意味着“使用缺省属性创建目录”，尽管在某些情况下可能发生这种情况。特定于新软件包的任何目录都必须明确指定所有属性。

复合软件包

任何包含可重定位对象的软件包称为可重定位软件包。这可能令人产生误解，因为可重定位软件包可能在其 pkgmap 文件中包含绝对路径。在 pkgmap 文件中使用根 (/) 条目可以增强软件包的可重定位特性。同时具有可重定位条目和根条目的软件包称为**复合**软件包。

示例－传统解决方案

假定 SUNWstuf 软件包中的一个对象是在运行级别 2 执行的启动脚本。文件 /etc/rc2.d/S70dostuf 需要作为该软件包的一部分安装，但不能将其放置到基目录中。假定可重定位软件包是唯一的解决方案，pkginfo 和 pkgmap 可能如下所示。

pkginfo 文件

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

pkgmap 文件

```
: 1 1758
1 d none opt/SUNWstuf/EZstuf 0775 root bin
1 f none opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none opt/SUNWstuf/HRDstuf 0775 root bin
1 f none opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none etc    ? ? ?
1 d none etc/rc2.d ? ? ?
1 f none etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

此方法和绝对软件包方法之间没有太大差异。实际上，作为绝对软件包使用会更好，因为如果管理员为此软件包提供备用基目录，它将无法正常工作！

实际上，此软件包中只有一个文件需要相对于根保持固定，其余文件都可以任意移动。本节其余部分将讨论如何使用复合软件包解决此问题。

超越传统

本节描述的方法不适用于所有软件包，但是，在将软件包安装到异构环境期间该方法确实能够改善性能。该方法很少适用于作为 Solaris OS 的一部分提供的软件包（随附的软件包）；然而，非随附软件包可以实行非传统打包。

鼓励采用可重定位软件包的原因是支持以下需求：

在添加或删除软件包时，已安装的软件产品的现有理想行为将保持不变。

非随附软件包应该驻留在 `/opt` 下，以便保证新软件包不会干扰现有产品。

复合软件包另一特性

在构建功能复合软件包时，需要遵循两个规则：

- 根据软件包对象的绝大部分的安装位置来建立基目录。
- 如果某个软件包对象安装到基目录以外的其他公共目录（例如 `/etc`），请在 `prototype` 文件中将其指定为绝对路径名。

换句话说，因为“可重定位”意味着该对象可以安装到任何位置而仍然能够正常工作，所以不能将 init 在引导时运行的任何启动脚本视为可重定位的！尽管在提供的软件包中将 /etc/passwd 指定为相对路径没有任何问题，但它只有一个安装位置。

使绝对路径名看起来像可重定位对象

如果您要构建一个复合软件包，绝对路径必须以不干扰已安装的现有软件的方式工作。可以完全包含在 /opt 中的软件包可避免此问题，因为不存在形成阻碍的现有文件。当 /etc 中的一个文件包括在软件包中时，您必须确保绝对路径名的行为方式与相对路径名的预期行为方式相同。请考虑下面两个示例。

示例－修改文件

说明

将一个条目添加到表中，或者对象是可能要由其他程序或软件包修改的一个新表。

实现

将该对象定义为文件类型 e 并定义为属于 build、awk 或 sed 类。执行此任务的脚本必须可与添加自身一样高效地删除自身。

示例

需要将一个条目添加到 /etc/vfstab 中，以便支持新的固态硬盘。

pkgmap 文件中的条目可能是

```
1 e sed /etc/vfstab ? ? ?
```

request 脚本询问操作员 /etc/vfstab 是否应该由软件包修改。如果操作员回答“否”，那么 request 脚本将列出有关如何手动完成该工作的说明，并将执行：

```
echo "CLASSES=none" >> $1
```

如果操作员回答“是”，那么它将执行：

```
echo "CLASSES=none sed" >> $1
```

该命令将激活将要执行必要修改的类操作脚本。sed 类意味着软件包文件 /etc/vfstab 是一个 sed 程序，该程序包含目标系统上同名文件的安装和删除操作。

示例－创建新文件

说明

对象是一个不太可能在以后进行编辑的全新文件，或者它将替换另一个软件包拥有的某个文件。

实现

将软件包对象定义为文件类型 `f`，并使用能够撤消更改的类操作脚本来安装该对象。

示例

`/etc` 中需要一个全新的文件来提供必要的信息，以便支持名为 `/etc/shdisk.conf` 的固态硬盘。`pkgmap` 文件中的条目可能如下所示：

```
.
.
.
1 f newetc /etc/shdisk.conf
.
.
.
```

类操作脚本 `i.newetc` 负责安装此文件以及需要安装到 `/etc` 的所有其他文件。该脚本将执行检查以确保该位置不存在其他文件。如果不存在其他文件，它只是将新文件复制到该位置。如果该位置已经有文件，该脚本将在安装新文件之前备份该文件。脚本 `r.newetc` 可根据需要删除这些文件并恢复原始文件。以下是安装脚本的关键片段。

```
# i.newetc
while read src dst; do
      if [ -f $dst ]; then
        dstfile=‘basename $dst‘
        cp $dst $PKGSAV/$dstfile
      fi
      cp $src $dst
done

if [ "${1}" = "ENDOFCLASS" ]; then
      cd $PKGSAV
      tar cf SAVE.newetc .
      $INST_DATADIR/$PKG/install/squish SAVE.newetc
fi
```

请注意，此脚本使用 PKGSAV 环境变量存储将被替换的文件的备份。当参数 ENDOFCLASS 传递给该脚本时，即 pkgadd 命令通知该脚本这些条目是此类中的最后一批条目，此时，该脚本将归档并压缩使用专用压缩程序（存储在软件包的安装目录中）保存的文件。

尽管在软件包更新期间使用 PKGSAV 环境变量不可靠，但如果软件包未更新（例如，未通过修补程序更新），则备份文件将是安全的。下面的删除脚本包含的代码用于处理另一个问题：pkgrm 命令的旧版本不向脚本传递 PKGSAV 环境变量的正确路径。

删除脚本可能如下所示。

```
# r.newetc

# make sure we have the correct PKGSAV
if [ -d $PKG_INSTALL_ROOT$PKGSAV ]; then
      PKGSAV="$PKG_INSTALL_ROOT$PKGSAV"
fi

# find the unsquish program
UNSQUISH_CMD=‘dirname $0‘/unsquish

while read file; do
      rm $file
done

if [ "${1}" = ENDOFCLASS ]; then
      if [ -f $PKGSAV/SAVE.newetc.sq ]; then
         $UNSQUISH_CMD $PKGSAV/SAVE.newetc
      fi

      if [ -f $PKGSAV/SAVE.newetc ]; then
         targetdir=dirname $file    # get the right directory
         cd $targetdir
            tar xf $PKGSAV/SAVE.newetc
            rm $PKGSAV/SAVE.newetc
      fi
fi
```

此脚本使用软件包数据库的安装目录中的专用卸载算法 (unsquish)。这由 pkgadd 命令在安装时自动完成。所有未被 pkgadd 命令特别识别为仅限于安装的脚本都将保留在此目录中，供 pkgrm 命令使用。您无法肯定该目录位于何处，但您可以确定该目录是无层次的，并且包含该软件包的所有适当信息文件和安装脚本。此脚本根据以下事实查找该目录：即类操作脚本肯定从包含 unsquish 程序的目录中执行。

另外请注意，此脚本并不只是假定目标目录是 /etc。该目录实际上可能是 /export/root/client2/etc。可以用以下两种方式之一构建正确的目录。

- 使用 ${PKG_INSTALL_ROOT}/etc 构造，或者。

- 获取由 pkgadd 命令传递的文件目录名（就是此脚本的作用）。

通过对软件包中的每个绝对对象使用此方法，您可以确信当前的理想行为保持不变或者至少是可恢复的。

示例 − 复合软件包

以下是复合软件包的 pkginfo 和 pkgmap 文件的示例。

pkginfo 文件

```
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

pkgmap 文件

```
: 1 1758
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 e daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i i.daemon 509 39560 752978103
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i r.daemon 320 24573 742152591
```

尽管 S70dostuf 属于 daemon 类，但指向它的目录（这些目录在安装时已经就位）属于 none 类。即使这些目录对于此软件包是唯一的，您也应该将它们保留在 none 类中。这样做的原因是，这些目录需要首先被创建而且最后删除，而对于 none 类而言始终如此。pkgadd 命令创建这些目录；它们不是从软件包复制的，而且不会传递给将创建的类操作脚本。相反，这些目录由 pkgadd 命令在调用安装类操作脚本之前创建，而 pkgrm 命令会在删除类操作脚本完成之后删除这些目录。

这意味着，如果一个特殊类中的目录包含 none 类中的对象，pkgrm 命令在尝试删除该目录时会失败，因为该目录无法及时变成空目录。如果类 none 的一个对象将插入到某个特殊类的目录中，该目录不会及时出现以便接受该对象。pkgadd 命令将在对象安装期间动态创建该目录，并且可能无法在最终看到 pkgmap 定义时同步该目录的属性。

注 – 在将目录指定给类时，请一定记住创建和删除顺序。

使软件包可远程安装

所有软件包都**必须**可远程安装。可远程安装意味着您不假定安装软件包的管理员正在向运行 pkgadd 命令的系统的根 (/) 文件系统进行安装。如果在某个过程脚本中，需要到达目标系统的 /etc/vfstab 文件，则需要使用 PKG_INSTALL_ROOT 环境变量。换句话说，路径名 /etc/vfstab 会将您引导至运行 pkgadd 命令的系统的 /etc/vfstab 文件，但管理员可能正在向位于 /export/root/client3 的客户机进行安装。路径 ${PKG_INSTALL_ROOT}/etc/vfstab 肯定能够将您引导至目标文件系统。

示例－安装到客户机系统

在此示例中，SUNWstuf 软件包被安装到 client3，该客户机在其根 (/) 文件系统中配置有 /opt。此软件包的另一个版本已经安装在 client3 上，并且基目录从管理文件 thisadmin 中设置为 basedir=/opt/$PKGINST。（有关管理文件的更多信息，请参见第 115 页中的 “缺省管理文件”。）服务器上执行的 pkgadd 命令是：

```
# pkgadd -a thisadmin -R /export/root/client3 SUNWstuf
```

下表列出传递给过程脚本的环境变量及其值。

表 6–1 传递给过程脚本的值

环境变量	值
PKGINST	SUNWstuf.2
PKG_INSTALL_ROOT	/export/root/client3

表 6–1 传递给过程脚本的值 （续）

环境变量	值
CLIENT_BASEDIR	/opt/SUNWstuf.2
BASEDIR	/export/root/client3/opt/SUNWstuf.2

示例－安装到服务器或独立系统

要在与上一个示例相同的环境下安装到服务器或独立系统，所使用的命令是：

```
# pkgadd -a thisadmin SUNWstuf
```

下表列出传递给过程脚本的环境变量及其值。

表 6–2 传递给过程脚本的值

环境变量	值
PKGINST	SUNWstuf.2
PKG_INSTALL_ROOT	未定义。
CLIENT_BASEDIR	/opt/SUNWstuf.2
BASEDIR	/opt/SUNWstuf.2

示例－挂载共享文件系统

假定 SUNWstuf 软件包在服务器上的 /export/SUNWstuf/share 创建并共享了一个文件系统。在将软件包安装到客户机系统时，需要更新客户机系统的 /etc/vfstab 文件以挂载这一共享文件系统。在这种情况下可以使用 CLIENT_BASEDIR 变量。

客户机上的条目需要根据客户机的文件系统呈现挂载点。无论安装是从服务器进行还是从客户机进行，都应该正确构建该行。假定服务器的系统名是 $SERVER。您可以转至 $PKG_INSTALL_ROOT/etc/vfstab，并使用 sed 或 awk 命令为客户机的 /etc/vfstab 文件构建以下行。

```
$SERVER:/export/SUNWstuf/share - $CLIENT_BASEDIR/usr nfs - yes ro
```

例如，对于服务器 universe 和客户机系统 client9，客户机系统的 /etc/vfstab 文件中的该行可能如下所示：

```
universe:/export/SUNWstuf/share - /opt/SUNWstuf.2/usr nfs - yes ro
```

正确使用这些参数时，该条目始终挂载客户机的文件系统，而无论它是在本地构建还是从服务器构建。

修补软件包

软件包的修补程序只是一个专门用于覆写原始软件包中某些文件的稀疏软件包。除了在交付介质上节省空间以外，发布稀疏软件包没有其他真正原因。您还可以发布更改了几个文件的完整原始软件包，或者提供通过网络访问经过修改的软件包的权限。只要只有这些新文件实际上是不同的（其他文件未重新编译），`pkgadd` 命令就会安装这些区别。请查看以下有关修补软件包的准则。

- 如果系统足够复杂，那么明智的做法是建立一种修补程序标识系统，以便确保任意两个修补程序在试图更正不同的异常行为时不会替换同一个文件。例如，Sun 修补程序基本号被指定它们所负责的文件互斥集。
- 能够回退修补程序是必要的。

至关重要的一点是，修补程序软件包的版本号必须与原始软件包的版本号相同。您应该使用如下形式的单独 `pkginfo` 文件条目来跟踪软件包的修补程序状态：

`PATCH=`*patch_number*

如果软件包版本在进行修补后发生改变，则相当于您创建了该软件包的另一个实例，这样，管理修补后的产品将变得极为困难。这种渐进式实例修补方法可以将早期 Solaris OS 发行版的某些优势传递下去，但却使更复杂系统的管理变得繁琐。

修补程序中的所有区域参数都必须与软件包中的区域参数相匹配。

对于组成 Solaris OS 的软件包而言，虽然可能有多个修补后的实例，但在软件包数据库中应该只有该软件包的一个副本。为了使用 `removef` 命令从已安装的软件包中删除某个对象，您需要判断哪些实例拥有该文件。

然而，如果您的软件包（该软件包不属于 Solaris OS）需要确定**属于** Solaris OS 的特定软件包的修补级别，那么这就成为一个需要在此处解决的问题。安装脚本可能非常大但不会产生重大影向，因为它们不存储在目标文件系统中。通过类操作脚本和其他各种过程脚本，您可以使用 `PKGSAV` 环境变量保存经过更改的文件（或者保存到其他某个更加持久的目录），以便能够回退已安装的修补程序。您还可以通过 `request` 脚本设置适当的环境变量，以监视修补程序历史记录。以下各节中的脚本假定可能存在多个修补程序，其编号方案在应用于单个软件包时具有某种含义。在这种情况下，各个修补程序编号表示软件包内的功能相关文件的一个子集。两个编号不同的修补程序不能更改同一个文件。

为了将一个常规的稀疏软件包转换成修补程序软件包，可将以下各节描述的脚本简单地封装到该软件包中。这些脚本都是公认的标准软件包组件，只有最后两个名为 `patch_checkinstall` 和 `patch_postinstall` 的脚本除外。如果您要包括回退修补程序功能，可以将这两个脚本合并到回退软件包中。这些脚本十分简单，它们的各种任务也容易完成。

注 – 此修补方法可用于修补客户机系统，但服务器上的客户机根目录必须拥有正确的权限，以允许用户 `install` 或 `nobody` 读取。

checkinstall 脚本

`checkinstall` 脚本验证修补程序是否适合于此特定软件包。一旦确认这一点，该脚本将构建**修补程序列表**和**修补程序信息**列表，然后将它们插入到响应文件中，以便合并到软件包数据库中。

修补程序列表是已经影响当前软件包的修补程序的列表。这一修补程序列表记录在已安装的软件包的 `pkginfo` 文件中，记录该列表的行可能如下所示：

```
PATCHLIST=patch_id  patch_id  ...
```

修补程序信息列表是当前修补程序所依赖的修补程序的列表。这一修补程序列表也记录在 `pkginfo` 文件中，记录该列表的行可能如下所示。

```
PATCH_INFO_103203-01=Installed... Obsoletes:103201-01 Requires: \ Incompatibles: 120134-01
```

注 – 这些行（及其格式）被声明为公共接口。任何为 Solaris 软件包发行修补程序的公司都应该相应地更新此列表。当修补程序交付时，修补程序内的每个软件包都包含一个执行此任务的 `checkinstall` 脚本。该同一个 `checkinstall` 脚本还会更新其他一些特定于修补程序的参数。这是新的修补程序体系结构，称为“直接实例修补”。

在此示例中，原始软件包及其修补程序都存在于同一个目录中。两个原始软件包命名为 `SUNWstuf.v1` 和 `SUNWstuf.v2`，而它们的修补程序分别命名为 `SUNWstuf.p1` 和 `SUNWstuf.p2`。这意味着，过程脚本可能很难判断这些文件来自哪个目录，因为对于 `PKG` 参数除去了软件包名称中的点 (".") 之后的所有内容，并且 `PKGINST` 环境变量是指已安装的实例，而不是源实例。因此，过程脚本可以找到源目录，`checkinstall` 脚本（始终从源目录执行）执行查询并且将位置作为变量 `SCRIPTS_DIR` 传递。如果源目录中只有一个名为 `SUNWstuf` 的软件包，则过程脚本可能已经使用 `$INSTDIR/$PKG` 找到它。

```
# checkinstall script to control a patch installation.
# directory format options.
#
#       @(#)checkinstall 1.6 96/09/27 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
```

```
PATH=/usr/sadm/bin:$PATH

INFO_DIR=‘dirname $0‘
INFO_DIR=‘dirname $INFO_DIR‘    # one level up

NOVERS_MSG="PaTcH_MsG 8 Version $VERSION of $PKG is not installed on this system."
ALRDY_MSG="PaTcH_MsG 2 Patch number $Patch_label is already applied."
TEMP_MSG="PaTcH_MsG 23 Patch number $Patch_label cannot be applied until all \
restricted patches are backed out."

# Read the provided environment from what may have been a request script
. $1

# Old systems can’t deal with checkinstall scripts anyway
if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        exit 0
fi

#
# Confirm that the intended version is installed on the system.
#
if [ "${UPDATE}" != "yes" ]; then
        echo "$NOVERS_MSG"
        exit 3
fi

#
# Confirm that this patch hasn’t already been applied and
# that no other mix-ups have occurred involving patch versions and
# the like.
#
Skip=0
active_base=‘echo $Patch_label | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $Patch_label | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+Patchvers_pfx_lnth) } ’‘

# Is this a restricted patch?
if echo $active_base | egrep -s "Patchstrict_str"; then
        is_restricted="true"
        # All restricted patches are backoutable
        echo "PATCH_NO_UNDO=" >> $1
else
        is_restricted="false"
fi

for patchappl in ${PATCHLIST}; do
        # Is this an ordinary patch applying over a restricted patch?
```

```
        if [ $is_restricted = "false" ]; then
                if echo $patchappl | egrep -s "Patchstrict_str"; then
                        echo "$TEMP_MSG"
                        exit 3;
                fi
        fi

        # Is there a newer version of this patch?
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")\
+Patchvers_pfx_lnth) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        echo "PaTcH_MsG 1 Patch number $Patch_label is \
superceded by the already applied $patchappl."
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        # Not newer, it’s the same
                        if [ "$PATCH_UNCONDITIONAL" = "true" ]; then
                                if [ -d $PKGSAV/$Patch_label ]; then
                                        echo "PATCH_NO_UNDO=true" >> $1
                                fi
                        else
                                echo "$ALRDY_MSG"
                                exit 3;
                        fi
                fi
        fi
done

# Construct a list of applied patches in order
echo "PATCHLIST=${PATCHLIST} $Patch_label" >> $1

#
# Construct the complete list of patches this one obsoletes
#
ACTIVE_OBSOLETES=$Obsoletes_label

if [ -n "$Obsoletes_label" ]; then
        # Merge the two lists
        echo $Obsoletes_label | sed ’y/\ /\n/’ | \
        nawk -v PatchObsList="$PATCH_OBSOLETES" ’
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");
```

```
                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                Inserted=0;
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                if (Inserted == 0) {
                                        PatchObsCnt[PatchIndex]=PatchObsCnt\
[PatchIndex]+1;
                                        Inserted=1;
                                } else {
                                        PatchObsCnt[PatchIndex]=0;
                                }
                        }
                }
                if (Inserted == 0) {
                        printf ("%s@1 ", $0);
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] != 0) {
                                printf("%s@%d ", PatchObs[PatchIndex], \
PatchObsCnt[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1
        # Clear the parameter since it has already been used.
        echo "Obsoletes_label=" >> $1

        # Pass it's value on to the preinstall under another name
        echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $1
fi

#
# Construct PATCH_INFO line for this package.
#

tmpRequire=`nawk -F= ' $1 ~ /REQUIR/ { print $2 } ' $INFO_DIR/pkginfo `
```

```
tmpIncompat=‘nawk -F= ’ $1 ~ /INCOMPAT/ { print $2 } ’ $INFO_DIR/pkginfo ‘

if [ -n "$tmpRequire" ] && [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
          Incompatibles: $tmpIncompat" >> $1
elif [ -n "$tmpRequire" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
Incompatibles: " >> $1
elif [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: \
$tmpIncompat" >> $1
else
        echo "PATCH_INFO_$Patch_label=Installed: ‘date‘ From: ‘uname -n‘ \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: " >> $1
fi

#
# Since this script is called from the delivery medium and we may be using
# dot extensions to distinguish the different patch packages, this is the
# only place we can, with certainty, trace that source for our backout
# scripts. (Usually $INST_DATADIR would get us there).
#
echo "SCRIPTS_DIR=‘dirname $0‘" >> $1

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

preinstall 脚本

preinstall 脚本可初始化 prototype 文件、信息文件和安装脚本，以便构建回退软件包。此脚本非常简单，并且此示例中的其余脚本只允许回退软件包描述常规文件。

如果您要在回退软件包中恢复符号链接、硬链接、设备和命名管道，可以修改 preinstall 脚本，以便使用 pkgproto 命令将交付的 pkgmap 文件与安装的文件进行比较，然后为要在回退软件包中更改的每个非文件创建一个 prototype 文件条目。应该使用的方法类似于类操作脚本中的方法。

脚本 patch_checkinstall 和 patch_postinstall 会从 preinstall 脚本插入到软件包源树中。这两个脚本撤消了修补程序执行的操作。

```
# This script initializes the backout data for a patch package
# directory format options.
#
#       @(#)preinstall 1.5 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH
recovery="no"

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
    if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] || [ -d \
"$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
        recovery="yes"
    fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$BUILD_DIR/files/reloc
ROOT_DIR=$BUILD_DIR/files/root
PROTO_FILE=$BUILD_DIR/prototype
PKGINFO_FILE=$BUILD_DIR/pkginfo
THIS_DIR=‘dirname $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi
```

```
#
# Unless specifically denied, initialize the backout patch data by
# creating the build directory and copying over the original pkginfo
# which pkgadd saved in case it had to be restored.
#
if [ "$PATCH_NO_UNDO" != "true" ] && [ "$recovery" = "no" ]; then
        if [ -d $BUILD_DIR ]; then
                rm -r $BUILD_DIR
        fi

        # If this is a retry of the same patch then recovery is set to
        # yes. Which means there is a build directory already in
        # place with the correct backout data.

        if [ "$recovery" = "no" ]; then
                mkdir $BUILD_DIR
                mkdir -p $RELOC_DIR
                mkdir $ROOT_DIR
        fi

        #
        # Here we initialize the backout pkginfo file by first
        # copying over the old pkginfo file and themn adding the
        # ACTIVE_PATCH parameter so the backout will know what patch
        # it's backing out.
        #
        # NOTE : Within the installation, pkgparam returns the
        # original data.
        #
        pkgparam -v $PKGINST | nawk '
                $1 ~ /PATCHLIST/        { next; }
                $1 ~ /PATCH_OBSOLETES/  { next; }
                $1 ~ /ACTIVE_OBSOLETES/ { next; }
                $1 ~ /Obsoletes_label/  { next; }
                $1 ~ /ACTIVE_PATCH/     { next; }
                $1 ~ /Patch_label/      { next; }
                $1 ~ /UPDATE/   { next; }
                $1 ~ /SCRIPTS_DIR/      { next; }
                $1 ~ /PATCH_NO_UNDO/    { next; }
                $1 ~ /INSTDATE/ { next; }
                $1 ~ /PKGINST/  { next; }
                $1 ~ /OAMBASE/  { next; }
                $1 ~ /PATH/     { next; }
                { print; } ' > $PKGINFO_FILE
        echo "ACTIVE_PATCH=$Patch_label" >> $PKGINFO_FILE
        echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $PKGINFO_FILE
```

```
# And now initialize the backout prototype file with the
# pkginfo file just formulated.
echo "i pkginfo" > $PROTO_FILE

# Copy over the backout scripts including the undo class
# action scripts
for script in $SCRIPTS_DIR/*; do
        srcscript=‘basename $script‘
        targscript=‘echo $srcscript | nawk ’
                { script=$0; }
                /u\./ {
                        sub("u.", "i.", script);
                        print script;
                        next;
                }
                /patch_/ {
                        sub("patch_", "", script);
                        print script;
                        next;
                }
                { print "dont_use" } ’‘
        if [ "$targscript" = "dont_use" ]; then
                continue
        fi

        echo "i $targscript=$FILE_DIR/$targscript" >> $PROTO_FILE
        cp $SCRIPTS_DIR/$srcscript $FILE_DIR/$targscript
done
 #
# Now add entries to the prototype file that won’t be passed to
# class action scripts. If the entry is brand new, add it to the
# deletes file for the backout package.
#
Our_Pkgmap=‘dirname $SCRIPTS_DIR‘/pkgmap
BO_Deletes=$FILE_DIR/deletes

nawk -v basedir=${BASEDIR:-/} ’
        BEGIN { count=0; }
        {
                token = $2;
                ftype = $1;
        }
        $1 ~ /[#\!:]/ { next; }
        $1 ~ /[0123456789]/ {
                if ( NF >= 3) {
                        token = $3;
                        ftype = $2;
                } else {
```

```
                                next;
                        }
                }
                { if (ftype == "i" || ftype == "e" || ftype == "f" || ftype == \
"v" || ftype == "d") { next; } }
                {
                        equals=match($4, "=")-1;
                        if ( equals == -1 ) { print $3, $4; }
                        else { print $3, substr($4, 0, equals); }
                }
                ' < $Our_Pkgmap | while read class path; do
                        #
                        # NOTE: If pkgproto is passed a file that is
                        # actually a hard link to another file, it
                        # will return ftype "f" because the first link
                        # in the list (consisting of only one file) is
                        # viewed by pkgproto as the source and always
                        # gets ftype "f".
                        #
                        # If this isn't replacing something, then it
                        # just goes to the deletes list.
                        #
                        if valpath -l $path; then
                                Chk_Path="$BASEDIR/$path"
                                Build_Path="$RELOC_DIR/$path"
                                Proto_From="$BASEDIR"
                        else    # It's an absolute path
                                Chk_Path="$PKG_INSTALL_ROOT$path"
                                Build_Path="$ROOT_DIR$path"
                                Proto_From="$PKG_INSTALL_ROOT"
                        fi
                         #
                        # Hard links have to be restored as regular files.
                        # Unlike the others in this group, an actual
                        # object will be required for the pkgmk.
                        #
                        if [ -f "$Chk_Path" ]; then
                                mkdir -p `dirname $Build_Path`
                                cp $Chk_Path $Build_Path
                                cd $Proto_From
                                pkgproto -c $class "$Build_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                                cd $THIS_DIR
                        elif [ -h "$Chk_Path" -o \
                             -c "$Chk_Path" -o \
                             -b "$Chk_Path" -o \
                             -p "$Chk_Path" ]; then
                                pkgproto -c $class "$Chk_Path=$path" 1>> \
```

```
$PROTO_FILE 2> /dev/null
                        else
                                echo $path >> $BO_Deletes
                        fi
                done
fi

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

类操作脚本

类操作脚本创建每个替换现有文件的文件的副本，并为回退软件包向 `prototype` 文件中添加一个相应的行。这一切都由十分简单的 `nawk` 脚本完成。类操作脚本接收一个源/目标对列表，其中包括不与相应的已安装文件匹配的普通文件。必须在 `preinstall` 脚本中处理符号链接和其他非文件。

```
# This class action script copies the files being replaced
# into a package being constructed in $BUILD_DIR. This class
# action script is only appropriate for regular files that
# are installed by simply copying them into place.
#
# For special package objects such as editable files, the patch
# producer must supply appropriate class action scripts.
#
# directory format options.
#
#       @(#)i.script 1.6 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

ECHO="/usr/bin/echo"
SED="/usr/bin/sed"
PKGPROTO="/usr/bin/pkgproto"
EXPR="/usr/bin/expr"    # used by dirname
MKDIR="/usr/bin/mkdir"
CP="/usr/bin/cp"
RM="/usr/bin/rm"
```

```
MV="/usr/bin/mv"

recovery="no"
Pn=$$
procIdCtr=0

CMDS_USED="$ECHO $SED $PKGPROTO $EXPR $MKDIR $CP $RM $MV"
LIBS_USED=""

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
        if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] ||
\
[ -d "$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
                recovery="yes"
        fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
PROGNAME=‘basename $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        PATCH_NO_UNDO="true"
fi

# Since this is generic, figure out the class.
Class=‘echo $PROGNAME | nawk ’ { print substr($0, 3)  }’‘

# Since this is an update, $BASEDIR is guaranteed to be correct
BD=${BASEDIR:-/}

cd $BD

#
# First, figure out the dynamic libraries that can trip us up.
```

```
#
if [ -z "$PKG_INSTALL_ROOT" ]; then
        if [ -x /usr/bin/ldd ]; then
                LIB_LIST=‘/usr/bin/ldd $CMDS_USED | sort -u | nawk ’
                        $1 ~ /\// { continue; }
                        { printf "%s ", $3 } ’‘
        else
                LIB_LIST="/usr/lib/libc.so.1 /usr/lib/libdl.so.1
\
/usr/lib/libw.so.1 /usr/lib/libintl.so.1 /usr/lib/libadm.so.1 \
/usr/lib/libelf.so.1"
        fi
fi

#
# Now read the list of files in this class to be replaced. If the file
# is already in place, then this is a change and we need to copy it
# over to the build directory if undo is allowed. If it’s a new entry
# (No $dst), then it goes in the deletes file for the backout package.
#
procIdCtr=0
while read src dst; do
        if [ -z "$PKG_INSTALL_ROOT" ]; then
                Chk_Path=$dst
                for library in $LIB_LIST; do
                        if [ $Chk_Path = $library ]; then
                                $CP $dst $dst.$Pn
                                LIBS_USED="$LIBS_USED $dst.$Pn"
                                LD_PRELOAD="$LIBS_USED"
                                export LD_PRELOAD
                        fi
                done
        fi

        if [ "$PATCH_PROGRESSIVE" = "true" ]; then
                # If this is being used in an old-style patch, insert
                # the old-style script commands here.

                #XXXOld_CommandsXXX#
                echo >/dev/null # dummy
        fi

        if [ "${PATCH_NO_UNDO}" != "true" ]; then
                #
                # Here we construct the path to the appropriate source
                # tree for the build. First we try to strip BASEDIR. If
                # there’s no BASEDIR in the path, we presume that it is
                # absolute and construct the target as an absolute path
```

```
# by stripping PKG_INSTALL_ROOT. FS_Path is the path to
# the file on the file system (for deletion purposes).
# Build_Path is the path to the object in the build
# environment.
#
if [ "$BD" = "/" ]; then
        FS_Path=‘$ECHO $dst | $SED s@"$BD"@@‘
else
        FS_Path=‘$ECHO $dst | $SED s@"$BD/"@@‘
fi

# If it’s an absolute path the attempt to strip the
# BASEDIR will have failed.
if [ $dst = $FS_Path ]; then
        if [ -z "$PKG_INSTALL_ROOT" ]; then
                FS_Path=$dst
                Build_Path="$ROOT_DIR$dst"
        else
                Build_Path="$ROOT_DIR‘echo $dst | \
                    sed s@"$PKG_INSTALL_ROOT"@@‘"
                FS_Path=‘echo $dst | \
                    sed s@"$PKG_INSTALL_ROOT"@@‘
        fi
else
        Build_Path="$RELOC_DIR/$FS_Path"
fi

if [ -f $dst ]; then    # If this is replacing something
        cd $FILE_DIR
        #
        # Construct the prototype file entry. We replace
        # the pointer to the filesystem object with the
        # build directory object.
        #
        $PKGPROTO -c $Class $dst=$FS_Path | \
            $SED -e s@=$dst@=$Build_Path@ >> \
            $BUILD_DIR/prototype

        # Now copy over the file
        if [ "$recovery" = "no" ]; then
                DirName=‘dirname $Build_Path‘
                $MKDIR -p $DirName
                $CP -p $dst $Build_Path
        else
                # If this file is already in the build area skip it
                if [ -f "$Build_Path" ]; then
                        cd $BD
                        continue
```

```
                                else
                                        DirName=‘dirname $Build_Path‘
                                        if [ ! -d "$DirName" ]; then
                                                $MKDIR -p $DirName
                                        fi
                                        $CP -p $dst $Build_Path
                                fi
                        fi

                        cd $BD
                else    # It’s brand new
                        $ECHO $FS_Path >> $BO_Deletes
                fi
        fi

        # If special processing is required for each src/dst pair,
        # add that here.
        #
        #XXXSpecial_CommandsXXX#
        #

        $CP $src $dst.$$$procIdCtr
        if [ $? -ne 0 ]; then
                $RM $dst.$$$procIdCtr 1>/dev/null 2>&1
        else
                $MV -f $dst.$$$procIdCtr $dst
                for library in $LIB_LIST; do
                        if [ "$library" = "$dst" ]; then
                                LD_PRELOAD="$dst"
                                export LD_PRELOAD
                        fi
                done
        fi
        procIdCtr=‘expr $procIdCtr + 1‘
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

#
# Release the dynamic libraries
#
for library in $LIBS_USED; do
        $RM -f $library
done

exit 0
```

postinstall 脚本

postinstall 脚本使用其他脚本提供的信息创建回退软件包。因为 pkgmk 和 pkgtrans 命令不需要软件包数据库，所以可以在软件包安装时执行这些命令。

在此示例中，通过在保存目录中（使用 PKGSAV 环境变量）构建一个流格式软件包来允许撤消修补程序。这并不显而易见，但此软件包必须采用流格式，因为在 pkgadd 操作期间，保存目录会发生移动。如果 pkgadd 命令应用于自己的保存目录中的一个软件包，那么有关软件包源在任何指定时刻所在位置的假设将变得非常不可靠。流格式软件包被解压缩到一个临时目录中并在此处安装。（目录格式软件包将从保存目录开始安装，并且发现自己在 pkgadd 失败安全操作期间突然被重定位。）

要确定应用于软件包的修补程序，请使用以下命令：

```
$ pkgparam SUNWstuf PATCHLIST
```

除 PATCHLIST（一个 Sun 公共接口）之外，此示例中的参数名称中没有什么重要内容。您可以取代 PATCH 使用传统 SUNW_PATCHID，而其他各种列表（例如 PATCH_EXCL 和 PATCH_REQD）可以相应地重命名。

如果某些修补程序软件包依赖于可从同一个介质中获得的其他修补程序软件包，那么 checkinstall 脚本可以确定这一点，并且按照与升级示例（请参见 第 161 页中的 “升级软件包”）相同的方式创建一个将由 postinstall 脚本执行的脚本。

```
# This script creates the backout package for a patch package
#
# directory format options.
#
# @(#) postinstall 1.6 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

# Description:
#       Set the TYPE parameter for the remote file
#
# Parameters:
#       none
#
# Globals set:
#       TYPE

set_TYPE_parameter () {
        if [ ${PATCH_UNDO_ARCHIVE:?????} = "/dev" ]; then
                # handle device specific stuff
                TYPE="removable"
```

```
        else
                TYPE="filesystem"
        fi
}

#
# Description:
#       Build the remote file that points to the backout data
#
# Parameters:
#       $1:     the un/compressed undo archive
#
# Globals set:
#       UNDO, STATE

build_remote_file () {
        remote_path=$PKGSAV/$Patch_label/remote
        set_TYPE_parameter
        STATE="active"

        if [ $1 = "undo" ]; then
                UNDO="undo"
        else
                UNDO="undo.Z"
        fi

        cat > $remote_path << EOF
# Backout data stored remotely
TYPE=$TYPE
FIND_AT=$ARCHIVE_DIR/$UNDO
STATE=$STATE
EOF
}

PATH=/usr/sadm/bin:$PATH

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

if [ ! -n "$PATCH_UNDO_ARCHIVE" ]; then
        PATCH_UNDO_ARCHIVE="none"
```

```
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
THIS_DIR=‘dirname $0‘
PROTO_FILE=$BUILD_DIR/prototype
TEMP_REMOTE=$PKGSAV/$Patch_label/temp

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # remove the scripts that are left behind
        install_scripts=‘dirname $0‘
        rm $install_scripts/checkinstall \
$install_scripts/patch_checkinstall $install_scripts/patch_postinstall

        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi
#
# At this point we either have a deletes file or we don’t. If we do,
# we create a prototype entry.
#
if [ -f $BO_Deletes ]; then
        echo "i deletes=$BO_Deletes" >> $BUILD_DIR/prototype
fi

#
# Now delete everything in the deletes list after transferring
# the file to the backout package and the entry to the prototype
# file. Remember that the pkgmap will get the CLIENT_BASEDIR path
# but we have to actually get at it using the BASEDIR path. Also
# remember that removef will import our PKG_INSTALL_ROOT
#
Our_Deletes=$THIS_DIR/deletes
if [ -f $Our_Deletes ]; then
        cd $BASEDIR

        cat $Our_Deletes | while read path; do
                Reg_File=0

                if valpath -l $path; then
                        Client_Path="$CLIENT_BASEDIR/$path"
                        Build_Path="$RELOC_DIR/$path"
```

```
                        Proto_Path=$BASEDIR/$path
                else    # It's an absolute path
                        Client_Path=$path
                        Build_Path="$ROOT_DIR$path"
                        Proto_Path=$PKG_INSTALL_ROOT$path
                fi

                # Note: If the file isn't really there, pkgproto
                # doesn't write anything.
                LINE=`pkgproto $Proto_Path=$path`
                ftype=`echo $LINE | nawk '{ print $1 }'`
                if [ $ftype = "f" ]; then
                        Reg_File=1
                fi

                if [ $Reg_File = 1 ]; then
                        # Add source file to the prototype entry
                        if [ "$Proto_Path" = "$path" ]; then
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@2`
                        else
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@`
                        fi

                        DirName=`dirname $Build_Path`
                        # make room in the build tree
                        mkdir -p $DirName
                        cp -p $Proto_Path $Build_Path
                fi

                # Insert it into the prototype file
                echo $LINE 1>>$PROTO_FILE 2>/dev/null

                # Remove the file only if it's OK'd by removef
                rm `removef $PKGINST $Client_Path` 1>/dev/null 2>&1
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Unless specifically denied, make the backout package.
#
if [ "$PATCH_NO_UNDO" != "true" ]; then
        cd $BUILD_DIR   # We have to build from here.

        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                STAGE_DIR="$PATCH_UNDO_ARCHIVE"
```

```
                ARCHIVE_DIR="$PATCH_UNDO_ARCHIVE/$Patch_label/$PKGINST"
                mkdir -p $ARCHIVE_DIR
                mkdir -p $PKGSAV/$Patch_label
        else
                if [ -d $PKGSAV/$Patch_label ]; then
                        rm -r $PKGSAV/$Patch_label
                fi
                STAGE_DIR=$PKGSAV
                ARCHIVE_DIR=$PKGSAV/$Patch_label
                mkdir $ARCHIVE_DIR
        fi

        pkgmk -o -d $STAGE_DIR 1>/dev/null 2>&1
        pkgtrans -s $STAGE_DIR $ARCHIVE_DIR/undo $PKG 1>/dev/null 2>&1
        compress $ARCHIVE_DIR/undo
        retcode=$?
        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                if [ $retcode != 0 ]; then
                        build_remote_file "undo"
                else
                        build_remote_file "undo.Z"
                fi
        fi
        rm -r $STAGE_DIR/$PKG

        cd ..
        rm -r $BUILD_DIR
        # remove the scripts that are left behind
        install_scripts=`dirname $0`
        rm $install_scripts/checkinstall $install_scripts/patch_\
checkinstall $install_scripts/patch_postinstall
fi

#
# Since this apparently worked, we'll mark as obsoleted the prior
# versions of this patch - installpatch deals with explicit obsoletions.
#
cd ${PKG_INSTALL_ROOT:-/}
cd var/sadm/pkg

active_base=`echo $Patch_label | nawk '
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } '`

List=`ls -d $PKGINST/save/${active_base}*`
if [ $? -ne 0 ]; then
        List=""
fi
```

```
for savedir in $List; do
        patch=‘basename $savedir‘
        if [ $patch = $Patch_label ]; then
                break
        fi

        # If we get here then the previous patch gets deleted
        if [ -f $savedir/undo ]; then
                mv $savedir/undo $savedir/obsolete
                echo $Patch_label >> $savedir/obsoleted_by
        elif [ -f $savedir/undo.Z ]; then
                mv $savedir/undo.Z $savedir/obsolete.Z
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/remote ]; then
                ‘grep . $PKGSAV/$patch/remote | sed ’s/STATE=.*/STATE=obsolete/
’ > $TEMP_REMOTE‘
                rm -f $PKGSAV/$patch/remote
                mv $TEMP_REMOTE $PKGSAV/$patch/remote
                rm -f $TEMP_REMOTE
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/obsolete -o -f $savedir/obsolete.Z ]; then
                echo $Patch_label >> $savedir/obsoleted_by
        fi
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

patch_checkinstall 脚本

```
# checkinstall script to validate backing out a patch.
# directory format option.
#
#       @(#)patch_checkinstall 1.2 95/10/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

LATER_MSG="PaTcH_MsG 6 ERROR: A later version of this patch is applied."
NOPATCH_MSG="PaTcH_MsG 2 ERROR: Patch number $ACTIVE_PATCH is not installed"
```

```
NEW_LIST=""

# Get OLDLIST
. $1

#
# Confirm that the patch that got us here is the latest one installed on
# the system and remove it from PATCHLIST.
#
Is_Inst=0
Skip=0
active_base=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
for patchappl in ${OLDLIST}; do
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        puttext "$LATER_MSG"
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        Is_Inst=1
                        Skip=1
                fi
        fi
        if [ $Skip = 1 ]; then
                Skip=0
        else
                NEW_LIST="${NEW_LIST} $patchappl"
        fi
done

if [ $Is_Inst = 0 ]; then
        puttext "$NOPATCH_MSG"
        exit 3
fi

#
# OK, all’s well. Now condition the key variables.
#
echo "PATCHLIST=${NEW_LIST}" >> $1
echo "Patch_label=" >> $1
echo "PATCH_INFO_$ACTIVE_PATCH=backed out" >> $1
```

```
# Get the current PATCH_OBSOLETES and condition it
Old_Obsoletes=$PATCH_OBSOLETES

echo $ACTIVE_OBSOLETES | sed 'y/\ /\n/' | \
nawk -v PatchObsList="$Old_Obsoletes" '
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                PatchObsCnt[PatchIndex]=PatchObsCnt[PatchIndex]-1;
                        }
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] > 0 ) {
                                printf("%s@%d ", PatchObs[PatchIndex], PatchObsCnt\
[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1

        # remove the used parameters
        echo "ACTIVE_OBSOLETES=" >> $1
        echo "Obsoletes_label=" >> $1

exit 0
```

patch_postinstall脚本

```
# This script deletes the used backout data for a patch package
# and removes the deletes file entries.
#
```

```
# directory format options.
#
#       @(#)patch_postinstall 1.2 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
PATH=/usr/sadm/bin:$PATH
THIS_DIR=‘dirname $0‘

Our_Deletes=$THIS_DIR/deletes

#
# Delete the used backout data
#
if [ -f $Our_Deletes ]; then
        cat $Our_Deletes | while read path; do
                if valpath -l $path; then
                        Client_Path=‘echo "$CLIENT_BASEDIR/$path" | sed s@//@/@‘
                else    # It’s an absolute path
                        Client_Path=$path
                fi
                rm ‘removef $PKGINST $Client_Path‘
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Remove the deletes file, checkinstall and the postinstall
#
rm -r $PKGSAV/$ACTIVE_PATCH
rm -f $THIS_DIR/checkinstall $THIS_DIR/postinstall

exit 0
```

升级软件包

升级软件包的过程与覆写软件包的过程极为不同。尽管有一些特殊工具支持对作为 Solaris OS 的一部分交付的标准软件包进行升级，但可以设计一个非随附软件包来支持它自己的升级－前面的多个示例描述了一些可在管理员指导下具有前瞻性并且控制精确安装方法的软件包。您还可以设计 `request` 脚本以支持软件包的直接升级。如果管理员选择安装一个软件包以便完全替换另一个软件包，并且不留下残余的过时文件，软件包脚本可以执行此任务。

此示例中的 request 脚本和 postinstall 脚本提供了一个简单的可升级软件包。request 脚本与管理员通信，然后在 /tmp 目录中设置一个简单的文件以删除旧的软件包实例。（虽然 request 脚本创建了一个被禁止的文件，但这不会有什么问题，因为每个人都有权访问 /tmp。）

然后， postinstall 脚本执行 /tmp 中的 shell 脚本，该脚本对旧软件包执行必要的 pkgrm 命令，然后删除它自身。

此示例演示基本升级。该示例少于 50 行代码，其中包括一些相当长的消息。可以对其进行扩展以回退升级，或者根据设计者的要求对软件包进行其他重要转换。

升级选项的用户界面设计必须完全确保管理员充分了解升级过程，并且已经主动请求升级而不是并行安装。只要用户界面能够清楚地说明操作，执行诸如升级这样的复杂操作就不会出现什么错误。

request 脚本

```
# request script
control an upgrade installation

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

UPGRADE_MSG="Do you want to upgrade the installed version ?"

UPGRADE_HLP="If upgrade is desired, the existing version of the \
    package will be replaced by this version. If it is not \
    desired, this new version will be installed into a different \
    base directory and both versions will be usable."

UPGRADE_NOTICE="Conflict approval questions may be displayed. The \
    listed files are the ones that will be upgraded. Please \
    answer \"y\" to these questions if they are presented."

pkginfo -v 1.0 -q SUNWstuf.\*

if [ $? -eq 0 ]; then
      # See if upgrade is desired here
      response=‘ckyorn -p "$UPGRADE_MSG" -h "$UPGRADE_HLP"‘
      if [ $response = "y" ]; then
            OldPkg=‘pkginfo -v 1.0 -x SUNWstuf.\* | nawk ’ \
            /SUNW/{print $1} ’‘
            # Initiate upgrade
            echo "PATH=/usr/sadm/bin:$PATH" > $UPGR_SCRIPT
            echo "sleep 3" >> $UPGR_SCRIPT
            echo "echo Now removing old instance of $PKG" >> \
```

```
            $UPGR_SCRIPT
            if [ ${PKG_INSTALL_ROOT} ]; then
                  echo "pkgrm -n -R $PKG_INSTALL_ROOT $OldPkg" >> \
                  $UPGR_SCRIPT
            else
                  echo "pkgrm -n $OldPkg" >> $UPGR_SCRIPT
            fi
            echo "rm $UPGR_SCRIPT" >> $UPGR_SCRIPT
            echo "exit $?" >> $UPGR_SCRIPT

            # Get the original package's base directory
            OldBD=`pkgparam $OldPkg BASEDIR`
            echo "BASEDIR=$OldBD" > $1
            puttext -l 5 "$UPGRADE_NOTICE"
      else
            if [ -f $UPGR_SCRIPT ]; then
                  rm -r $UPGR_SCRIPT
            fi
      fi
fi

exit 0
```

postinstall 脚本

```
# postinstall
to execute a simple upgrade

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

if [ -f $UPGR_SCRIPT ]; then
      sh $UPGR_SCRIPT &
fi

exit 0
```

创建类归档软件包

类归档软件包是对应用称序二进制接口 (Application Binary Interface, ABI) 的增强，其中，某些文件集已经组合为单个文件（即归档文件），并且可能已选择性地压缩或加密。类归档格式最多可使初始安装速度提高 30%，并且在将软件包和修补程序安装到可能活动的文件系统上的过程中提高可靠性。

以下各节提供有关归档软件包目录结构、关键字和 `faspac` 实用程序的信息。

归档软件包目录的结构

下图中显示的软件包条目表示包含软件包文件的目录。此目录必须与软件包同名。

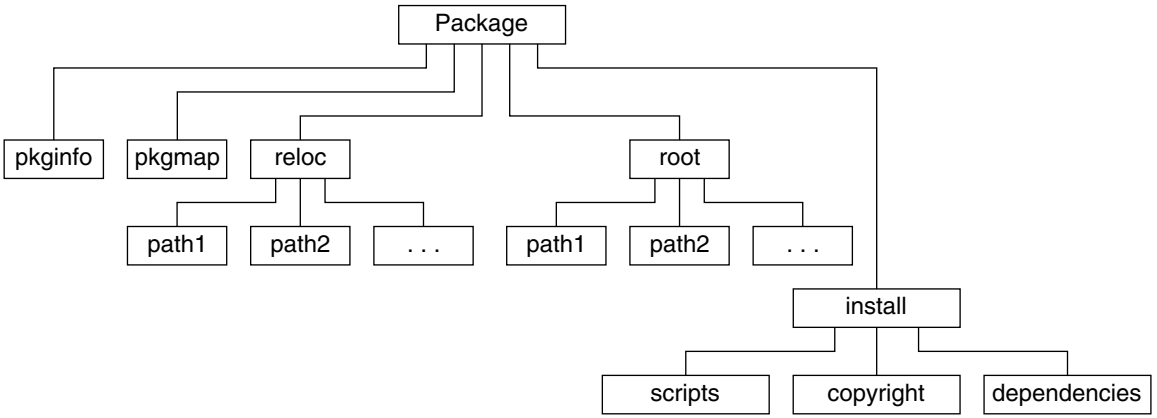


图 6–1 软件包目录结构

下面列出了软件包目录内包含的文件和目录的功能。

项	说明
`pkginfo`	文件，对软件包进行总体描述，包括特殊环境变量和安装指令
`pkgmap`	描述每个要安装对象的文件，如文件、目录或管道
`reloc`	可选目录，包含要相对于基目录安装的文件（可重定位的对象）
`root`	可选目录，包含要相对于 `root` 目录安装的文件（根对象）
`install`	可选目录，包含脚本和其他辅助文件（除了 `pkginfo` 和 `pkgmap`，所有 `ftype i` 文件都位于此处）

使用类归档格式，软件包生成器可以将 `reloc` 和 `root` 目录中的文件组合到归档文件中，然后对其进行压缩、加密或以所需的任何方式进行其他处理，以便提高安装速度，减小软件包大小，或者增加软件包的安全性。

ABI 允许将软件包内的任何文件指定给某个类。特定类中的所有文件都可以使用类操作脚本定义的自定义方法安装到磁盘中。此自定义方法可以利用目标系统中提供的程序或随软件包一起提供的程序。得到的格式很像标准 ABI 格式。如下图所示，另一个目录被添加。将要归档的任何文件类只是组合为单个文件，并且放置到 `archive` 目录中。系统将从 `reloc` 和 `root` 目录中删除所有归档文件，并且将一个安装类操作脚本放置到 `install` 目录中。

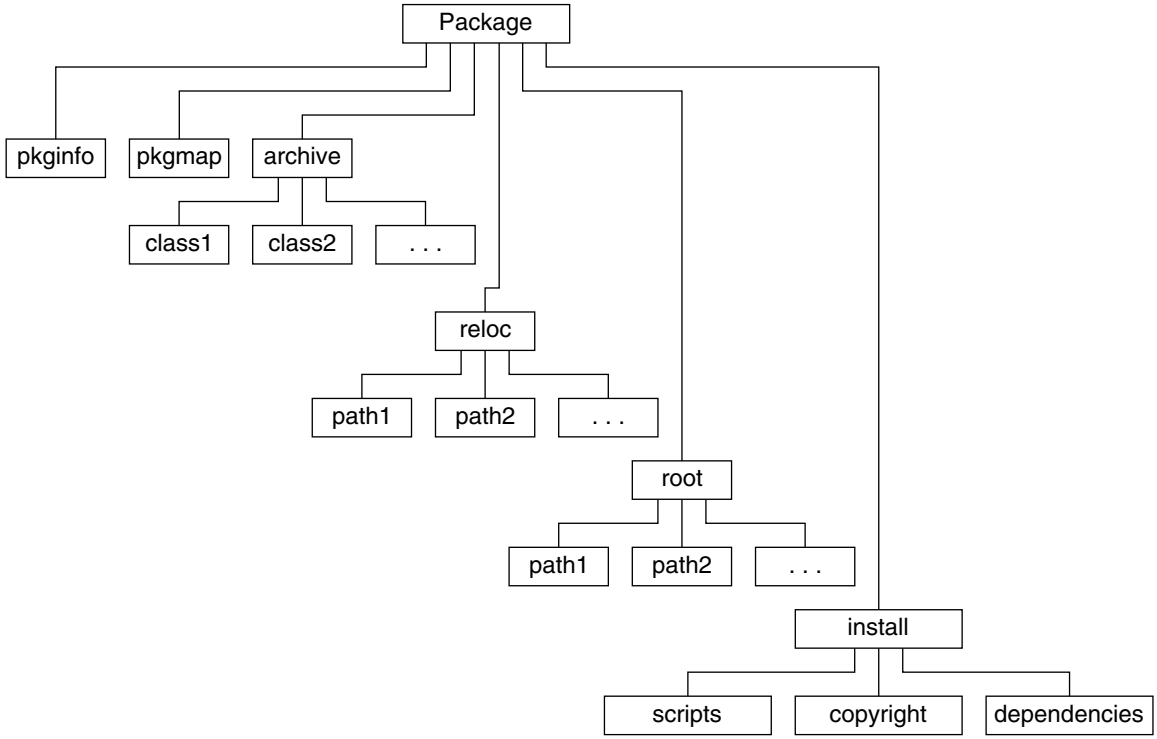


图 6–2 归档软件包目录结构

支持类归档软件包的关键字

为了支持这一新的类归档格式，三个采用关键字形式的新接口在 `pkginfo` 文件内具有特殊含义。这些关键字用于指定需要特殊处理的类。每个关键字语句的格式为：`keyword=class1[class2 ...]`。下表定义了每个关键字值。

关键字	说明
`PKG_SRC_NOVERIFY`	如果所交付的软件包的 `reloc` 或 `root` 目录中的文件属于指定类，则该关键字告诉 `pkgadd` 不要验证这些文件是否存在以及文件属性。所有归档类都需要此关键字，因为这些文件不再位于 `reloc` 或 `root` 目录中。它们是 `archive` 目录中的专用格式文件。

关键字	说明
PKG_DST_QKVERIFY	这些类中的文件在安装后使用一个快速算法进行验证，只有少量或者没有任何文本输出。快速验证首先正确设置每个文件的属性，然后检查以了解该操作是否成功。然后，将根据 pkgmap 测试文件大小和修改时间。不会执行 checksum 验证，该验证与标准验证机制相比错误恢复功能较差。如果在安装期间发生断电或磁盘故障，则目录文件可能与已安装的文件不一致。总能使用 pkgrm 解决这种不一致问题。
PKG_CAS_PASSRELATIVE	通常，安装类操作脚本从 stdin 接收告诉它安装哪些文件的源和目标对列表。指定给 PKG_CAS_PASSRELATIVE 的类不会获得这些源和目标对。相反，这些类会收到单个列表，其中第一个条目是源软件包的位置，其余条目是目标路径。这专门用于简化从归档文件中进行提取的操作。根据源软件包的位置，您可以在 archive 目录中找到归档文件。然后，目标路径被传递给负责提取归档文件内容的函数。所提供的每个目标路径对于基目录而言是绝对的或相对的，具体取决于该路径原来位于 root 还是位于 reloc。如果选择此选项，可能难以将相对路径和绝对路径都组合到单个类中。

对于每个归档类，都需要一个类操作脚本。这是一个包含 Bourne shell 命令的文件，该文件由 pkgadd 执行以便从归档中实际安装文件。如果在软件包的 install 目录中找到一个类操作脚本，pkgadd 会将所有安装职责移交给该脚本。该类操作脚本以超级用户权限运行，并且可以将其文件放置在目标系统中的任何位置。

注 – 实现类归档软件包绝对必须的唯一关键字是 PKG_SRC_NOVERIFY。其他关键字可用于提高安装速度或保存代码。

faspac 实用程序

faspac 实用程序将标准 ABI 软件包转换为随附软件包所使用的类归档格式。此实用程序使用 cpio 进行归档，使用 compress 进行压缩。生成的软件包在顶层目录中有一个名为 archive 的附加目录。在此目录中，将包含按类命名的所有归档文件。install 目录将包含解压缩每个归档文件所需的类操作脚本。绝对路径将不进行归档。

faspac 实用程序具有以下格式：

```
faspac [-m Archive Method] -a -s -q [-d Base Directory] /
[-x Exclude List] [List of Packages]
```

下表描述了每个 faspac 命令选项。

选项	说明
`-m` *Archive Method*	指示一种归档或压缩方法。`bzip2` 是所使用的缺省压缩实用程序。要切换到 zip 压缩或 unzip 解压缩方法，请使用 `-m zip`，或者，对于 cpio 或 compress，请使用 `-m cpio`。
`-a`	修复属性（只有超级用户才能执行此操作）。
`-s`	指示标准 ABI 类型软件包转换。此选项将经过 cpio 或压缩处理的软件包进行打包，并使其具有符合标准 ABI 的软件包格式。
`-q`	指示静默模式。
`-d` *Base Directory*	指示所有软件包所在的目录将由命令行根据需要进行操作。这与 *List of Packages* 条目互斥。
`-x` *Exclude List*	指示要从处理范围中排除的软件包的逗号分隔列表或用引号括起来的空格分隔列表。
List of Packages	指示要处理的软件包的列表。

词汇表

ABI 请参见应用程序二进制接口 (application binary interface, ABI)。

abstract syntax notation 1（抽象语法表示法 1） 一种表示抽象对象的方法。例如，ASN.1 定义公钥证书、组成证书的所有对象以及对象的收集顺序。不过，ASN.1 不指定针对存储或传输序列化对象的方式。

application binary interface（应用程序二进制接口） 对编译的应用程序和运行这些应用程序的操作系统之间的二进制系统接口的定义。

ASN.1 请参见抽象语法表示法 1 (abstract syntax notation 1, ASN.1)。

base directory（基目录） 可重定位对象的安装位置。在 `pkginfo` 文件中使用 `BASEDIR` 参数对基目录进行定义。

build time（生成时） 使用 `pkgmk` 命令生成软件包的时间。

build variable（生成变量） 以小写字母开头并在生成时被计算的变量。

certificate authority（证书颁发机构） 一种颁发证书的机构（例如 Verisign），对软件包进行签名时将使用该证书。

class action script（类操作脚本） 一种文件，用于定义要对一组软件包对象执行的一组操作。

class（类） 用于对软件包对象进行分组的名称。另请参见 class action script（类操作脚本）。

collectively relocatable object（可共同重定位的对象） 相对于通用安装基目录进行定位的软件包对象。另请参见 base directory（基目录）。

common name（通用名称） 带签名的软件包的软件包密钥库中列出的别名。

composite package（复合软件包） 包含可重定位路径名和绝对路径名的软件包。

`compver` **file（compver 文件）** 指定软件包向下兼容性的一种方法。

control file（控制文件） 控制是否安装软件包以及软件包安装方式和位置的文件。请参见 information file（信息文件）和 installation script（安装脚本）。

copyright（版权） 拥有和销售知识产权（例如，软件、源代码或文档）的权利。必须在 CD-ROM 和插入文本上声明拥有权，无论该版权归 SunSoft 还是另一方所有。在 SunSoft 文档中，也对版权拥有权进行了确认。

depend file（depend 文件） 解析基本软件包相关性的一种方法。另请参见 `compver` file（compver 文件）。

DER 请参见 distinguished encoding rules（唯一编码规则）。

digital signature（数字签名） 用于验证软件包完整性和安全性的一种编码消息。

distinguished encoding rules（唯一编码规则） ASN.1 对象的一种二进制表示法，用于定义在计算环境中针对存储或传输序列化 ASN.1 对象的方式。与带签名的软件包一起使用。

incompatible package（不兼容软件包） 与指定软件包不兼容的软件包。另请参见 `depend` file（depend 文件）。

individually relocatable object（可单独重定位的对象） 不限于定位到可共同重定位的对象所在的目录位置的软件包对象。使用 `prototype` 文件中 *path* 字段中的安装变量对其进行定义，安装位置通过 `request` 脚本或 `checkinstall` 脚本确定。

information file（信息文件） 一种文件，可以定义软件包相关性、提供版权信息或在目标系统上保留空间。

install time（安装时） 使用 `pkgadd` 命令安装软件包的时间。

install variable（安装变量） 以大写字母开头并在安装时被计算的变量。

installation script（安装脚本） 使您能够提供软件包自定义安装过程的脚本。

ITU-T Recommendation X.509（ITU-T 推荐标准 X.509） 一种指定广泛采用的 X.509 公钥证书语法的协议。

package abbreviation（软件包缩写） 通过 `pkginfo` 文件中的 `PKG` 参数定义的软件包的简短名称。

package identifier（软件包标识符） 通过 `pkgadd` 命令添加到软件包缩写的数字后缀。

package instance（软件包实例） 软件包的变体，通过组合软件包的 `pkginfo` 文件中的 `PKG`、`ARCH` 和 `VERSION` 参数的定义确定。

package keystore（软件包密钥库） 可通过软件包工具查询的证书和密钥的系统信息库。

package object（软件包对象） 包含在要安装到目标系统的软件包中的应用程序文件的另一种称呼。

package（软件包） 软件应用程序所需文件和目录的集合。

parametric path name（参数化路径名） 包含变量规范的路径名。

patch list（修补程序列表） 影响当前软件包的修补程序的列表。该修补程序列表记录在 `pkginfo` 文件中的已安装软件包中。

PEM 请参见 privacy enhanced message（保密性增强消息）。

PKCS12 请参见 public key cryptography standard #12（公钥密码学标准 #12）。

PKCS7 请参见 public key cryptography standard #7（公钥密码学标准 #7）。

prerequisite package（先决软件包） 依赖于其他软件包是否存在的软件包。另请参见 `depend` file（depend 文件）。

privacy enhanced message（保密性增强消息） 一种使用 base 64 编码和某些可选标头对文件进行编码的方式。广泛用于将证书和私钥编码到文件系统或电子邮件中存在的文件中。

private key（私钥） 仅为交换秘密消息的一方或多方所知的加密/解密密钥。该私钥与公钥结合使用以创建带签名的软件包。

procedure script（过程脚本） 定义软件包安装和删除过程中的特定时刻发生的操作的脚本。

public key cryptography standard #12（公钥密码学标准 #12） 此标准描述将加密对象存储在磁盘上的语法。软件包密钥库会以这种格式进行维护。

public key cryptography standard #7（公钥密码学标准 #7） 此标准描述可能应用了加密算法的数据的通用语法，例如数字签名和数字信封。带签名的软件包包含一个嵌入的 PKCS7 签名。

public key（公钥） 作为加密密钥生成的值，与从该公钥派生的私钥一起，可用于有效地加密消息和数字签名。

relocatable object（可重定位对象） 在目标系统上不需要绝对路径位置的软件包对象。其位置在安装过程中确定。另请参见 collectively relocatable object（可共同重定位的对象）和 individually relocatable object（可单独重定位的对象）。

relocatable（可重定位） 在 `prototype` 文件中使用相对路径名定义的软件包对象。

reverse dependency（反向相关性） 其他软件包依赖于您的软件包是否存在的一种条件。另请参见 `depend` file（depend 文件）。

segmented（分段） 不能存放在单个卷（例如软盘）上的软件包。

signed packages（带签名的软件包）	具有数字签名的常规流格式软件包，用于验证以下内容：软件包是否来自对其签名的实体；实体是否确实对软件包进行了签名；软件包自实体对其进行签名后是否经过修改；对软件包进行签名的实体是否是受信任实体。
`tar`	磁带归档检索。用于在介质中添加或提取文件的 Solaris 命令。
trusted certificate（受信任证书）	包含单个属于其他实体的公钥证书的证书。验证数字签名和启动到安全 (SSL) 服务器的连接时将使用受信任证书。
unsigned package（不带签名的软件包）	未进行任何加密或数字签名的常规 ABI 软件包。
user key（用户密钥）	用于存放敏感的加密密钥信息的密钥。加密密钥信息以受保护的格式存储，可防止未经授权的使用。创建带签名的软件包时将使用用户密钥。
X.509	请参见 ITU-T Recommendation X.509（ITU-T 推荐标准 X.509）。

索引

O

P

R

S