Red Hat Software Collections 3

打包指南

为 Red Hat Enterprise Linux 打包 Software Collections 的指南

Last Updated: 2023-08-30

Red Hat Software Collections 3 打包指南

为 Red Hat Enterprise Linux 打包 Software Collections 的指南

Petr Kovář
Red Hat Customer Content Services
pkovar@redhat.com

法律通告

摘要

打包指南介绍了 Software Collections 的解释以及如何构建和打包它们。熟悉 RPM 软件包的软件和系统管理员（但对 Software Collections 概念有新功能的新知识）可以使用本指南来开始使用 Software Collections。

目录

使开源包含更多 4

第 1 章 SOFTWARE COLLECTIONS 简介 5
1.1. 为什么使用 RPM 的软件包软件？ 5
1.2. 什么是 SOFTWARE COLLECTIONS? 5
1.3. 启用对 SOFTWARE COLLECTIONS 的支持 6
1.4. 安装 SOFTWARE COLLECTION 6
1.5. 列出已安装的 SOFTWARE COLLECTIONS 7
1.6. 启用 SOFTWARE COLLECTION 7
1.7. 列出启用的 SOFTWARE COLLECTIONS 8
1.8. 卸载 SOFTWARE COLLECTION 8

第 2 章 打包 SOFTWARE COLLECTIONS 9
2.1. 创建自己的 SOFTWARE COLLECTIONS 9
2.2. 文件系统层次结构 9
2.3. SOFTWARE COLLECTION ROOT 目录 10
2.4. SOFTWARE COLLECTION PREFIX 10
2.5. SOFTWARE COLLECTION 软件包名称 11
2.6. SOFTWARE COLLECTION SCRIPTLETS 11
2.7. 软件包布局 12
2.8. SOFTWARE COLLECTION MACROS 15
2.9. 常用的路径 REDEFINITIONS 17
2.10. 转换一个一致的 SPEC 文件 20
2.11. 卸载所有 SOFTWARE COLLECTION 目录 26
2.12. 构建 SOFTWARE COLLECTION 27

第 3 章 高级主题 29
3.1. 通过 NFS 使用 SOFTWARE COLLECTIONS 29
3.2. 将 SOFTWARE COLLECTION SCRIPTLET 转换为环境模块 31
3.3. 提供 SYSPATHS SUBPACKAGES 31
3.4. 在 SOFTWARE COLLECTIONS 中管理服务 33
3.5. SOFTWARE COLLECTION LIBRARY 支持 35
3.6. SOFTWARE COLLECTION .PC 文件支持 38
3.7. SOFTWARE COLLECTION MANPATH 支持 40
3.8. SOFTWARE COLLECTION CRONJOB 支持 42
3.9. SOFTWARE COLLECTION 日志文件支持 43
3.10. SOFTWARE COLLECTION LOGROTATE 支持 43
3.11. 软件集合 /VAR/RUN/ 文件支持 44
3.12. SOFTWARE COLLECTION LOCK 文件支持 44
3.13. 软件集合配置文件支持 45
3.14. SOFTWARE COLLECTION 内核模块支持 46
3.15. SOFTWARE COLLECTION SELINUX 支持 46
3.16. RED HAT ENTERPRISE LINUX 6 和 7 之间的区别 47

第 4 章 扩展 RED HAT SOFTWARE COLLECTIONS 49
4.1. 提供 SCLDEVEL 子软件包 49
4.2. 扩展 PYTHON27 和 RH-PYTHON35 SOFTWARE COLLECTIONS 51
4.3. 扩展 RH-RUBY23 软件集合 56
4.4. 扩展 RH-PERL524 SOFTWARE COLLECTION 63

第 5 章 SOFTWARE COLLECTIONS 故障排除 70
5.1. ERROR: LINE XX: UNKNOWN TAG: %SCL_PACKAGE SOFTWARE_COLLECTION_NAME 70

5.2. SCL 命令不存在 70
5.3. 无法打开 /ETC/SCL/PREFIXES/SOFTWARE_COLLECTION_NAME 70
5.4. SCL_SOURCE: 命令未找到 70

附录 A. 获取更多信息 **72**
A.1. RED HAT DEVELOPERS 72
A.2. 安装的文档 72
A.3. 访问红帽文档 72

附录 B. 修订历史记录 **74**
B.1. 致谢 76

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 信息。

第 1 章 SOFTWARE COLLECTIONS 简介

本章介绍了 Software Collections 或 SCLs 的概念和用法。

1.1. 为什么使用 RPM 的软件包软件？

RPM 软件包管理器(RPM)是在 Red Hat Enterprise Linux 上运行的软件包管理系统。RPM 可让您更轻松地分发、管理和更新您为 Red Hat Enterprise Linux 创建的软件。许多软件供应商通过传统的存档文件（如 tarball）发布其软件。但是，将软件打包成 RPM 软件包有几个优点。下面概述了这些优点。

使用 RPM，您可以：

安装、重新安装、删除、升级和验证软件包。

用户可以使用标准软件包管理工具（如 **Yum** 或 **PackageKit**）来安装、重新安装、删除、升级和验证 RPM 软件包。

使用已安装软件包的数据库查询和验证软件包。

由于 RPM 维护已安装软件包及其文件的数据库，因此用户可以轻松地查询和验证其系统上的软件包。

使用元数据描述软件包、安装说明等。

每个 RPM 软件包都包含描述软件包组件、版本、发行版本、大小、项目 URL、安装说明等的元数据。

将软件源打包为源代码和二进制软件包。

RPM 允许您获取软件源并将其打包到源和二进制软件包中。在源软件包中，您有 pristine 源以及所使用的任何补丁，以及完整的构建说明。随着软件的新版本发布，这个设计可简化软件包的维护。

将软件包添加到 Yum 存储库。

您可以将软件包添加到 **Yum** 存储库中，使客户端可以轻松地查找和部署您的软件。

数字签名您的软件包。

使用 GPG 签名密钥，您可以数字签名您的软件包，以便用户可以验证软件包的真实性。

有关 RPM 以及如何使用它的信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南 或 Red Hat Enterprise Linux 6 部署指南。

1.2. 什么是 SOFTWARE COLLECTIONS?

使用 Software Collections，您可以在系统中构建并同时安装同一软件组件的多个版本。Software Collections 对任何传统 RPM 软件包管理工具安装的软件包版本没有影响。

Software Collections:

不要覆盖系统文件

Software Collections 作为一组几个组件发布，它们提供它们的完整功能，而无需覆盖系统文件。

旨在避免与系统文件冲突

Software Collections 使用特殊的文件系统层次结构以避免单个 Software Collection 和基本系统安装之间可能存在冲突。

不需要更改 RPM 软件包管理器

Software Collections 不需要更改主机系统上 RPM 软件包管理器。

只需要对 spec 文件进行小更改

要将传统软件包转换为单个 Software Collection，您只需要对软件包 spec 文件进行小更改。

允许您使用单个 spec 文件构建传统软件包和 Software Collection 软件包

使用单个 spec 文件，您可以同时构建传统软件包和 Software Collection 软件包。

唯一的名称所有包含的软件包

使用 Software Collection 的命名空间时，Software Collection 中包含的所有软件包都被唯一命名。

不要与更新的软件包冲突

Software Collection 的命名空间可确保更新系统中的软件包不会造成任何冲突。

可以依赖于其他 Software Collections

由于一个 Software Collection 可以依赖于另一个软件，所以您可以定义多个级别的依赖项。

1.3. 启用对 SOFTWARE COLLECTIONS 的支持

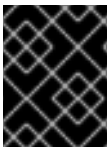
要在系统中启用对 Software Collections 的支持，以便您可以启用和构建 Software Collections，您需要安装的软件包 scl-utils 和 scl-utils-build。

如果系统上还没有安装 scl-utils 和 scl-utils-build 软件包，您可以以 root 用户身份在 shell 提示符后输入以下内容来安装它们：

```
# yum install scl-utils scl-utils-build
```

scl-utils 软件包提供 **scl** 工具，可让您在系统中启用 Software Collections。有关启用 Software Collections 的更多信息，请参阅 第 1.6 节 “启用 Software Collection”。

scl-utils-build 软件包提供了构建 Software Collections 非常重要的宏。有关构建 Software Collections 的更多信息，请参阅 第 2.12 节 “构建 Software Collection”。



重要

根据 Red Hat Enterprise Linux 系统可用的订阅，您可能需要启用 Optional **频道来安装** scl-utils-build 软件包。

1.4. 安装 SOFTWARE COLLECTION

要确保 Software Collection 在您的系统中，请安装 Software Collection 的 so-called metapackage。由于 Software Collections 与 RPM Package Manager 完全兼容，因此您可以使用 **Yum** 或 **PackageKit** 等传统工具进行此任务。

例如，要使用名为 **software_collection_1** 的 metapackage 安装 Software Collection，请运行以下命令：

```
# yum install software_collection_1
```

此命令将自动安装 Software Collection 中的所有软件包，这些软件包对用户使用 Software Collection 执行大多数常见任务至关重要。

Software Collections 只允许安装您要使用的软件包子集。例如，要使用 rh-ruby23 Software Collection 中的 Ruby 解释器，您只需要从该 Software Collection 安装软件包 rh-ruby23-ruby。

如果您安装依赖于 Software Collection 的应用程序，则该 Software Collection 将与应用程序的依赖项的其余部分一起安装。

有关 Software Collection metapackages 的详情，请参考 第 2.7.1 节 “Metapackage”。

有关 **Yum** 和 **PackageKit** 使用的详细信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南 或 Red Hat Enterprise Linux 6 部署指南。

1.5. 列出已安装的 SOFTWARE COLLECTIONS

要获取系统上安装的 Software Collections 列表，请运行以下命令：

```
scl --list
```

要获取指定 Software Collection 中包含的已安装软件包列表，请运行以下命令：

```
scl --list software_collection_1
```

1.6. 启用 SOFTWARE COLLECTION

scl 工具用于启用 Software Collection，并在 Software Collection 环境中运行应用程序。

可使用以下语法描述 **scl** 工具的常规用法：

```
scl action software_collection_1 software_collection_2 command
```

如果您使用多个参数运行命令，请记住将命令及其参数用引号括起来：

```
scl action software_collection_1 software_collection_2 'command --argument'
```

或者，使用 -- 命令分隔符来运行多个参数的命令：

```
scl action software_collection_1 software_collection_2 -- command --argument
```

请记住：

- *运行 scl 工具时，它会创建一个当前 shell 的子进程(subshell)。再次运行命令，然后创建 subshell 的 subshell。*
- *您可以列出为当前子 shell 启用的 Software Collections。请参阅 第 1.7 节 “列出启用的 Software Collections” 了解更多信息。*
- *您必须首先禁用已启用的软件集合，才能再次启用它。要禁用 Software Collection，退出启用 Software Collections 时创建的 subshell。*
- *当使用 scl 工具启用 Software Collection 时，您只能对启用的软件集合执行一个操作。在执行另一个操作前，必须先禁用已启用的软件集合。*

1.6.1. 直接运行应用程序

例如，要直接使用名为 software_collection_1 的 Software Collection 中的 **--version** 选项直接运行 Perl，请执行以下命令：

```
scl enable software_collection_1 'perl --version'
```

或者，您可以提供一个 sypaths 子软件包，以便更轻松地在 Software Collection 环境中运行命令。有关 syspaths 子软件包的详情请参考 第 3.3 节 “提供 syspaths Subpackages”。

1.6.2. 运行启用了多个 Software Collections 的 Shell

要在启用了多个 Software Collections 的环境中运行 Bash shell，请执行以下命令：

```
scl enable software_collection_1 software_collection_2 bash
```

以上命令启用两个 Software Collections，名为 software_collection_1 和 software_collection_2。

1.6.3. 运行存储在文件中的命令

要执行存储在 Software Collection 环境中的多个命令，请运行以下命令：

```
cat cmd | scl enable software_collection_1 -
```

以上命令执行命令，该命令存储在名为 software_collection_1 的软件集合的 **cmd** 文件中。

1.7. 列出启用的 SOFTWARE COLLECTIONS

要获取在当前会话中启用的 Software Collections 列表，请运行以下命令输出 **$X_SCLS** 环境变量：

```
echo $X_SCLS
```

1.8. 卸载 SOFTWARE COLLECTION

在卸载 Software Collection 时，您可以使用 Yum 或 PackageKit 等传统工具，因为 Software Collections 与 RPM Package Manager 完全兼容。例如：要卸载作为名为 **software_collection_1** 的软件集合一部分的所有软件包和子软件包，请运行以下命令：

```
yum remove software_collection_1\*
```

您还可以使用 **yum remove** 命令删除 scl 工具。

有关 Yum 和 PackageKit 使用的详细信息，请参阅 Red Hat Enterprise Linux 7 系统管理员指南 或 Red Hat Enterprise Linux 6 部署指南。

第 2 章 打包 SOFTWARE COLLECTIONS

本章介绍了打包 Software Collections。

2.1. 创建自己的 SOFTWARE COLLECTIONS

通常，您可以使用以下两种方法之一来部署依赖于现有软件集合的应用程序：

- 手动安装所有必需的 Software Collections 和软件包，然后部署应用程序，或者
- 为您的应用程序创建一个新的 Software Collection。

在为您的应用程序创建新软件集合时：

创建 Software Collection metapackage

每个 Software Collection 都会包括一个 metapackage，它安装一个 Software Collection 的软件包子集，这是用户使用 Software Collection 执行最常见任务所必需的。有关创建 metapackages 的更多信息，请参阅 第 2.7.1 节 “Metapackage”。

考虑指定 Software Collection 根目录的位置

建议您通过在 Software Collection spec 文件中设置 **%_scl_prefix** 宏来指定 Software Collection 根目录的位置。如需更多信息，请参阅 第 2.3 节 “Software Collection Root 目录”。

考虑为 Software Collection 软件包添加前缀

建议您使用 vendor 和 Software Collection 的名称作为 Software Collection 软件包的名称添加前缀。如需更多信息，请参阅 第 2.4 节 “Software Collection Prefix”。

指定应用程序所需的所有 Software Collections 和其他软件包作为依赖项

确保应用程序所需的所有 Software Collections 和其他软件包都指定为 Software Collection 的依赖项。如需更多信息，请参阅 第 2.10.8 节 “在另一软件集合上制作 Software Collection Depend”。

转换现有的传统软件包或创建新的软件集合软件包

确保 Software Collection 软件包 spec 文件中的所有宏都使用条件。有关如何转换现有软件包规格文件的更多信息，请参阅 第 2.10 节 “转换一个一致的 Spec 文件”。

构建软件集合

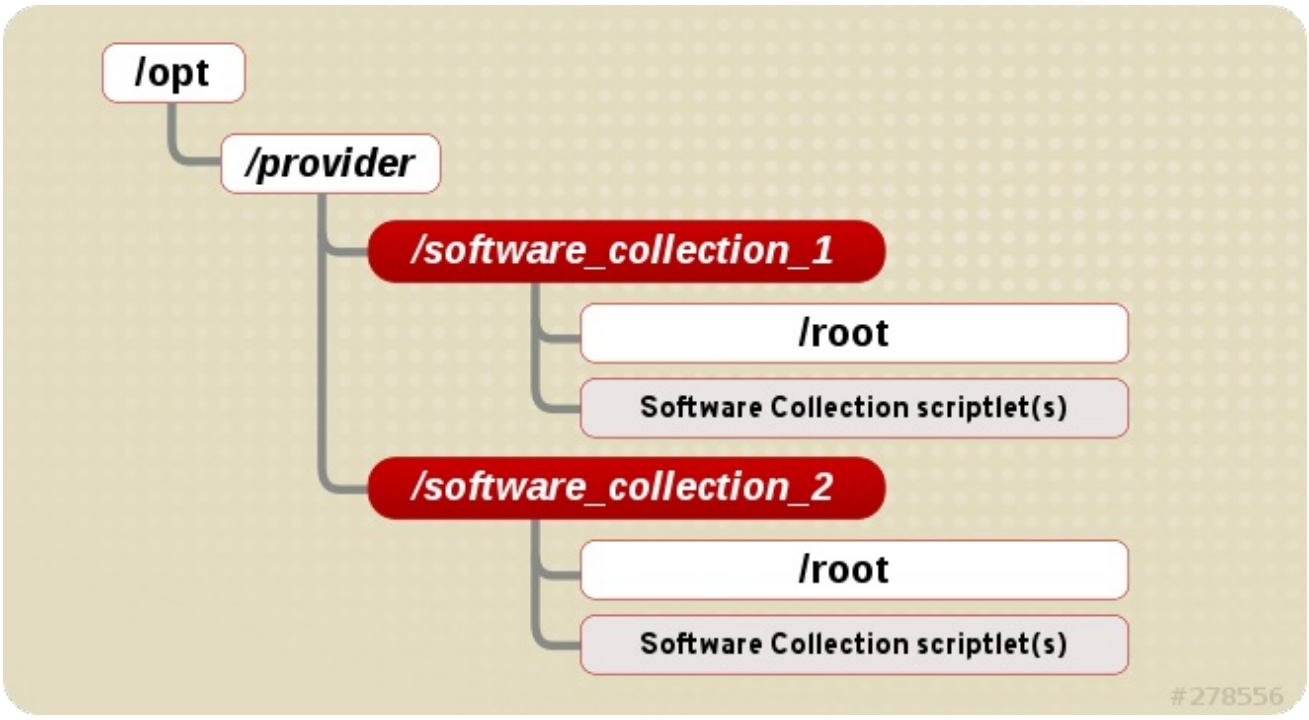
创建 Software Collection metapackage 并为 Software Collection 转换或创建软件包后，您可以使用 rpmbuild 实用程序构建 Software Collection。如需更多信息，请参阅第 2.12 节 “构建 Software Collection”。

2.2. 文件系统层次结构

Software Collections 的根目录通常位于 **/opt/** 目录中，以避免 Software Collections 和基本系统安装之间可能存在冲突。文件系统层次结构标准(FHS)建议使用 **/opt/** 目录。

以下是带有两个 Software Collections、Software_collection_ 1 和 software_collection_ 2 的文件系统层次结构布局的示例：

图 2.1. 软件集合文件系统层次结构



如上所示，每个 Software Collections 目录都包含 Software Collection 根目录，以及一个或多个 Software Collection scriptlets。有关 Software Collection scriptlets 的更多信息，请参阅 第 2.6 节 “Software Collection Scriptlets”。

2.3. SOFTWARE COLLECTION ROOT 目录

您可以通过在 spec 文件中设置 %_scl_prefix 宏来更改根目录的位置，如下例所示：

```
%global _scl_prefix /opt/provider
```

其中 provider 是注册供应商（供应商）名称（如果适用），使用 Linux Foundation 和从属 Linux 分配名称和数字授权机构(LANANA)，符合文件系统层次结构标准。

构建和分发 Software Collections 的每个机构或项目都应该使用自己的供应商名称，它们符合文件系统层次结构标准(FHS)，并避免 Software Collections 和基本系统安装之间可能存在冲突。

建议您使文件系统层次结构符合以下布局：

```
/opt/provider/prefix-application-version/
```

注意

您必须在 spec 文件中的 _scl_prefix 宏上方定义 %scl_package 宏。

有关文件系统层次结构标准的详情，请参考 http://www.pathname.com/fhs/。

有关 Linux 分配名称和编号授权的详情，请参考 http://www.lanana.org/。

2.4. SOFTWARE COLLECTION PREFIX

在命名 Software Collection 时，建议您为 Software Collection 的名称添加前缀，以避免与 Software Collection 一部分的软件包的系统版本冲突。

Software Collection 前缀由两个部分组成：

- 供应商 部分，用于定义提供程序的名称，以及
- Software Collection 本身的名称。

Software Collection 前缀的两个部分用短划线(-)分隔，如下例所示：

```
myorganization-ruby193
```

在本例中，myorganization 是提供程序的名称，ruby193 是 Software Collection 的名称。

虽然最终是一个厂商的决定，还是在前缀中指定供应商的名称，但强烈建议指定它。

一个值得注意的例外是 Red Hat Software Collections 1.x 首次附带的 Software Collections，它们不会在其前缀中指定供应商的名称。在 Red Hat Software Collections 2.0 及之后的版本中添加了较新的 Software Collections 使用 rh 作为提供程序的名称。例如：

```
rh-ruby23
```

2.5. SOFTWARE COLLECTION 软件包名称

Software Collection 软件包名称由两个部分组成：

- 前缀 部分，在 第 2.4 节 “Software Collection Prefix”中讨论，以及
- 作为 Software Collection 一部分的应用程序的名称和版本号。

Software Collection 软件包名称的这两个部分由短划线(-)分开，如下例所示：

```
myorganization-ruby193-foreman-1.1
```

在本例中，my organization-ruby193 是前缀，foreman-1.1 是应用的名称和版本号。

2.6. SOFTWARE COLLECTION SCRIPTLETS

Software Collection scriptlets 是简单的 shell 脚本，用于更改当前系统环境，因此 Software Collection 中的软件包组优先于系统上安装的对应传统软件包组。

要使用 Software Collection scriptlets，请使用作为 scl -utils 软件包一部分的 scl 工具。有关scl 的详情，请参考 第 1.6 节 “启用 Software Collection”。

单个 Software Collection 可以包含多个 Software Collection scriptlets。这些 scriptlets 位于 Software Collection 软件包中的 **/opt/provider/software_collection/** 目录中。如果您只需要在 Software Collection 中分发单个 scriptlet，则强烈建议您使用 **enable** 作为那个 scriptlet 的名称。当用户通过执行 **scl enable software_collection** 命令在 Software Collection 环境中运行命令时，**/opt/provider/software_collection/enable** scriptlet 会被用来更新搜索路径等。

请注意，Software Collection scriptlet 只能在运行 **scl enable** 命令创建的子 shell 中设置系统环境。subshell 仅在执行命令的时间处于活跃状态。

2.7. 软件包布局

每个 Software Collection 的布局都由 metapackage 组成，它安装其它软件包的子集，以及软件集合命名空间中的多个软件包，它们安装在 Software Collection 命名空间中。

2.7.1. Metapackage

每个 Software Collection 都会包括一个 metapackage，它安装一个 Software Collection 的软件包子集，这是用户使用 Software Collection 执行最常见任务所必需的。例如，基本软件包可以提供 Perl 语言解释器，但不能提供 Perl 扩展模块。metapackage 包含基本的文件系统层次结构，并提供多个 Software Collection 的 scriptlets。

metapackage 的目的是确保软件集合中的所有基本软件包都已正确安装，并可启用 Software Collection。

metapackage 会生成以下软件包，它们也是 Software Collection 的一部分：

主软件包：%name

Software Collection 中的主要软件包包含基本软件包的依赖项，它们包含在 Software Collection 中。主软件包不包含任何文件。

当为您的 Software Collection 软件包指定依赖项时，请确保 Software Collection 中没有其它软件包取决于主软件包。主软件包的目的是仅安装那些对用户使用 Software Collection 执行大多数常见任务所需的软件包。

通常，主软件包没有指定任何构建时间依赖项（例如，仅构建另一个 Software Collection 软件包依赖项的软件包）。

例如，如果 Software Collection 的名称是 **myorganization-ruby193**，则主软件包宏被扩展为：

```
myorganization-ruby193
```

运行时子软件包：%name-runtime

Software Collection 中的 runtime 子软件包拥有 Software Collection 的文件系统，并提供 Software Collection 的 scriptlets。需要安装此软件包，以便用户可以使用 Software Collection。

例如，如果 Software Collection 的名称是 **myorganization-ruby193**，则 runtime 子宏被扩展为：

```
myorganization-ruby193-runtime
```

构建子软件包：%name-build

Software Collection 中的 build 子软件包提供 Software Collection 的构建配置。它包含将软件包构建到 Software Collection 所需的 RPM 宏。build 子软件包是可选的，可以在 Software Collection 中排除。

例如，如果 Software Collection 的名称是 **myorganization-ruby193**，则构建子宏被扩展为：

```
myorganization-ruby193-build
```

myorganization-ruby193-build 子软件包的内容如下所示：

```
$ cat /etc/rpm/macros.ruby193-config
%scl myorganization-ruby193
```

syspaths 子软件包：%name-syspaths

Software Collection 中的 syspaths 子软件包提供了一种可选的方式，可将方便的 shell 包装程序和符号链接安装到标准路径，从而更改基本系统安装，但在 Software Collection 软件包中使二进制文件更易于使用。

例如，如果 Software Collection 的名称是 **myorganization-ruby193**，则 syspaths 子宏被扩展为：

> **myorganization-ruby193-syspaths**

有关 syspaths 子软件包的详情请参考 第 3.3 节 “提供 syspaths Subpackages”。

scldevel 子软件包：%name-scldevel

%name Software Collection 中的 scldevel 子软件包包含开发文件，这些文件在开发依赖于 %name Software Collection 的另一个 Software Collection 的软件包时非常有用。scldevel 子软件包是可选的，可以从 %name Software Collection 中排除。

例如，如果 Software Collection 的名称是 **myorganization-ruby193**，scldevel 子软件包宏将扩展为：

> **myorganization-ruby193-scldevel**

有关 scldevel 子软件包的详情请参考 第 4.1 节 “提供 scldevel 子软件包”。

2.7.2. 创建 Metapackage

在创建新 metapackage 时：

- 在 **%scl_package** 宏之上定义 metapackage spec 文件顶部的以下宏：
  - **scl_name_prefix** 指定用作软件集合名称中的前缀的供应商名称，如myorganization-。这与 **_scl_prefix** 不同，它指定了软件集合的根目录，但也使用供应商的名称。请参阅 第 2.4 节 “Software Collection Prefix” 了解更多信息。
  - **scl_name_base** 指定 Software Collection 的基本名称，如ruby。
  - **scl_name_version** 指定 Software Collection 的版本，例如93。
- 建议您定义一个 Software Collection 宏 **nfsmountable**，它更改配置和状态文件的位置，并使软件集合通过 NFS 提供。如需更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。
- 考虑指定 Software Collection 作为 metapackage 的依赖项，它们对 Software Collection 运行非常重要的所有软件包。这样，您可以确保使用 Software Collection metapackage 安装软件包。
- 建议您在 构建 子软件包中添加 **Requires: scl-utils-build**。
- 您不需要在 metapackage 中使用特定于 Software Collection 的宏的条件。
- 在 **enable** scriptlet 中包含 Software Collection 中软件包可能需要的任何路径撤销。

  有关常用路径重新definitions的详情，请参考 第 2.9 节 “常用的路径 Redefinitions”。

- *始终确保 metapackage 在 %prep 部分包含 %setup 宏，否则构建软件集合将失败。如果您不需要在 %setup 宏中使用特定选项，请将 %setup -c -T 命令添加到 %prep 部分。*

  这是因为 %setup 宏定义并创建 %buildsubdir 目录，它通常用于在构建时存储临时文件。如果您没有在 Software Collection 软件包中定义 %setup，则 %buildsubdir 目录中的文件会被覆盖，从而导致构建失败。

- *将您需要使用的任何宏添加到 构建 子软件包中的 macros.%{scl}-config 文件中。*

Metapackage 示例
要了解名为 myorganization-ruby193 的软件集合的典型 metapackage 如下所示，请参阅以下示例：

```
%global scl_name_prefix myorganization-
%global scl_name_base ruby
%global scl_name_version 193

%global scl %{scl_name_prefix}%{scl_name_base}%{scl_name_version}

# Optional but recommended: define nfsmountable
%global nfsmountable 1

%global _scl_prefix /opt/myorganization
%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
Requires: %{scl_prefix}less
BuildRequires: scl-utils-build

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build

%description build
Package shipping essential configuration macros to build %scl Software Collection.

# This is only needed when you want to provide an optional scldevel subpackage
%package scldevel
Summary: Package shipping development files for %scl

%description scldevel
Package shipping development files, especially useful for development of
packages depending on %scl Software Collection.
```

```
%prep
%setup -c -T

%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF

# This is only needed when you want to provide an optional scldevel subpackage
cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel << EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF

# Install the generated man page
mkdir -p %{buildroot}%{_mandir}/man7/
install -p -m 644 %{scl_name}.7 %{buildroot}%{_mandir}/man7/

%files

%files runtime -f filelist
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%files scldevel
%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel

%changelog
* Fri Aug 30 2013 John Doe &lt;jdoe@example.com&gt; 1-1
- Initial package
```

2.8. SOFTWARE COLLECTION MACROS

Software Collection 打包宏 scl 定义在哪里重新定位软件集合的文件结构。重新定位的文件结构是由 Software Collection 只使用的文件系统。

%scl_package 宏定义 Software Collection 的 metapackage 的文件所有权，并提供额外的打包宏以便在 Software Collection 环境中使用。

要能够使用单个 spec 文件构建传统软件包和 Software Collection 软件包，请为 Software Collection 宏添加前缀 %{?scl:macro}，如下例所示：

```
%{?scl:Requires: %scl_runtime}
```

在上例中，%scl_runtime 宏是 Requires 标签的值。宏和标签都使用 %{?scl: 前缀。

2.8.1. 特定于 Software Collection 的宏

下表显示了特定于特定软件集合的所有宏的列表。所有宏都有默认值，在大多数情况下，您不需要更改它们。

表 2.1. Software Collection Specific Macros

Macro	描述	示例值
%scl_name	Software Collection 的名称	**software_collection_1**
%scl_prefix	Software Collection 的名称，在末尾附加一个短划线	**software_collection_1-**
%pkg_name	原始软件包的名称	**perl**
%_scl_prefix	Software Collection 的根（而不是软件包的根）	**/opt/provider/**
%_scl_scripts	Software Collection 的 scriptlet 的位置	**/opt/provider/software_collection_1/**
%_scl_root	软件包的安装根(install-root)	**/opt/provider/software_collection_1/root/**
%scl_require_package software_collection_1 package_2	依赖于特定 Software Collection 中的特定软件包	**software_collection_1-package_2**

2.8.2. 宏不特定于软件集合

下表显示了不特定于特定 Software Collection 的宏列表。由于这些宏没有被重新定位且不指向 Software Collection 文件系统，所以它们允许您指向系统根文件系统。这些宏使用 _root 作为前缀。

所有宏都有默认值，在大多数情况下，您不需要更改它们。

表 2.2. Software Collection Non-Specific Macros

Macro	描述	relocated	示例值
%_root_prefix	Software Collection 的 **%_prefix** 宏	否	**/usr/**
%_root_exec_prefix	Software Collection 的 **%_exec_prefix** 宏	否	**/usr/**
%_root_bindir	Software Collection 的 **%_bindir** 宏	否	**/usr/bin/**
%_root_sbindir	Software Collection 的 **%_sbindir** 宏	否	**/usr/sbin/**

Macro	描述	relocated	示例值
%_root_datadir	Software Collection 的 **%_datadir** 宏	否	**/usr/share/**
%_root_sysconfdir	Software Collection 的 **%_sysconfdir** 宏	否	**/etc/**
%_root_libexecdir	Software Collection 的 **%_libexecdir** 宏	否	**/usr/libexec/**
%_root_sharedstatedir	Software Collection 的 **%_sharedstatedir** 宏	否	**/usr/com/**
%_root_localstatedir	Software Collection 的 **%_localstatedir** 宏	否	**/usr/var/**
%_root_includedir	Software Collection 的 **%_includedir** 宏	否	**/usr/include/**
%_root_infodir	Software Collection 的 **%_infodir** 宏	否	**/usr/share/info/**
%_root_mandir	Software Collection 的 **%_mandir** 宏	否	**/usr/share/man/**
%_root_initddir	Software Collection 的 **%_initddir** 宏	否	**/etc/rc.d/init.d/**
%**_root_libdir**	Software Collection 的 **%_libdir** 宏，如果 Software Collection 的 metapackage 是独立于平台的，则这个宏无法正常工作	否	**/usr/lib/**

2.8.3. nfsmountable Macro

*使用 Software Collection 宏 **nfsmountable** 可让您更改 **_sysconfdir**、**_sharedstatedir** 和 **_localstatedir** 宏的值，以便软件集合可以将其状态文件和配置文件位于 Software Collection 的 **/opt** 文件系统层次结构之外。这使得文件更易于管理，在通过 NFS 使用软件集合时也需要。*

*如果您不需要通过 NFS 支持 Software Collections，请使用 **nfsmountable** 是可选的，但推荐使用。如需更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。*

2.9. 常用的路径 REDEFINITIONS

*本节列出了在 **enable** scriptlet 中重新定义路径用来设置软件集合环境的环境变量。它们还用于在 Software Collection 文件系统层次结构中指定 Software Collection 组件的位置。*

*是否需要在 **enable** scriptlet 中指定路径撤销，取决于您选择包含在 Software Collection 中的软件包。环境变量通常遵循此模式：*

```
$ENV_VAR=$SCL_ENV_VAR:$ENV_VAR
```

2.9.1. 特定于语言的路径 Redefinitions

GEM_PATH

GEM_PATH 环境变量指定 Ruby gems 的位置。因此，它也用于扩展 rh-ruby23 软件集合的 Software Collections。如需更多信息，请参阅 第 4.3 节 "扩展 rh-ruby23 软件集合"。

在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export GEM_PATH="\${GEM_PATH:=%{gem_dir}:\`scl enable %{scl_ruby} -- ruby -e "print Gem.path.join(':')"\`}"
```

GOPATH

GOPATH 环境变量指定 Go 源和二进制文件的位置。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export GOPATH="%{gopath}\${GOPATH:+:\${GOPATH}}"
```

JAVACONFDIRS

JAVACONFDIRS 环境变量用于指定 java.conf 配置文件的位置。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export JAVACONFDIRS="%{_sysconfdir}/java\${JAVACONFDIRS:+:}\${JAVACONFDIRS:-}"
```

PERL5LIB

PERL5LIB 环境变量用于指定自定义 Perl 模块的位置，以便使用 %{?_scl_root} 前缀安装它们。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export PERL5LIB="%{_scl_root}%{perl_vendorlib}\${PERL5LIB:+:\${PERL5LIB}}"
```

PYTHONPATH

PYTHONPATH 环境变量指定自定义 Python 库的位置。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export PYTHONPATH="%{_scl_root}%{python_sitearch}:%{_scl_root}%{python_sitelib}\${PYTHONPATH:+:}\${PYTHONPATH:-}"
```

2.9.2. 其他路径 Redefinitions

CPATH

CPATH 环境变量指定要使用的 GCC 编译器的 include 路径。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export CPATH="%{_includedir}\${CPATH:+:\${CPATH}}"
```

INFOPATH

INFOPATH 环境变量指定包含 Info 文件的目录。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export INFOPATH="%{_infodir}\${INFOPATH:+:\${INFOPATH}}"
```

LD_LIBRARY_PATH

LD_LIBRARY_PATH 环境变量指定库的位置。如需更多信息，请参阅 第 3.5 节 "Software Collection Library 支持"。

在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
```

LIBRARY_PATH

LIBRARY_PATH 环境变量指定要使用的特殊链接器文件的位置或普通库。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export LIBRARY_PATH="%{_libdir}\${LIBRARY_PATH:+:\${LIBRARY_PATH}}"
```

MANPATH

MANPATH 环境变量指定 man page 的位置。如需更多信息，请参阅第 3.7 节 "Software Collection MANPATH 支持"。

在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export MANPATH="%{_mandir}:\${MANPATH:-}"
```

PATH

PATH 环境变量指定二进制文件的位置。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
```

PCP_DIR

PCP_DIR 环境变量指定 PCP 使用的文件和目录的位置。在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export PCP_DIR="%{_scl_root}"
```

PKG_CONFIG_PATH

PKG_CONFIG_PATH 环境变量指定 pkg-config 程序使用的 **.pc** 文件的位置。如需更多信息，请参阅 第 3.6 节 "Software Collection .pc 文件支持"。

在 **enable** scriptlet 中包括以下内容以重新定义环境变量：

```
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
```

XDG_CONFIG_DIRS

XDG_CONFIG_DIRS 环境变量根据 freedesktop.org 规格指定桌面配置文件的位置。在 enable scriptlet 中包括以下内容以重新定义环境变量：

```
export XDG_CONFIG_DIRS="%{_sysconfdir}/xdg:\${XDG_CONFIG_DIRS:-/etc/xdg}"
```

XDG_DATA_DIRS

XDG_DATA_DIRS 环境变量根据 freedesktop.org 规格指定桌面数据文件的位置。它用于某些 Software Collections 来查找特定于 Software Collection 的脚本或启用 bash 完成。

在 enable scriptlet 中包括以下内容以重新定义环境变量：

```
export XDG_DATA_DIRS="%{_datadir}:\${XDG_DATA_DIRS:-/usr/local/share:%{_root_datadir}}"
```

2.10. 转换一个一致的 SPEC 文件

本节讨论将传统 spec 文件转换为 Software Collection spec 文件，以便在传统软件包和 Software Collection 中使用转换的 spec 文件。

2.10.1. 转换的 Spec 文件示例

要查看 diff 文件将传统 spec 文件与转换的 spec 文件进行比较，请参考以下示例：

```
--- a/less.spec
+++ b/less.spec
@@ -1,10 +1,14 @@
+%{?scl:%global _scl_prefix /opt/provider}
+%{?scl:%scl_package less}
+%{!?scl:%global pkg_name %{name}}
+
 Summary: A text file browser similar to more, but better
-Name: less
+Name: %{?scl_prefix}less
 Version: 444
 Release: 7%{?dist}
 License: GPLv3+
 Group: Applications/Text
-Source: http://www.greenwoodsoftware.com/less/%{name}-%{version}.tar.gz
+Source: http://www.greenwoodsoftware.com/less/%{pkg_name}-%{version}.tar.gz
 Source1: lesspipe.sh
 Source2: less.sh
 Source3: less.csh
@@ -19,6 +22,7 @@ URL: http://www.greenwoodsoftware.com/less/
 Requires: groff
 BuildRequires: ncurses-devel
 BuildRequires: autoconf automake libtool
-Obsoletes: lesspipe < 1.0
+Obsoletes: %{?scl_prefix}lesspipe < 1.0
+%{?scl:Requires: %scl_runtime}

 %description
 The less utility is a text file browser that resembles more, but has
```

```
@@ -31,7 +35,7 @@ You should install less because it is a basic utility for viewing text
 files, and you'll use it frequently.

 %prep
-%setup -q
+%setup -q -n %{pkg_name}-%{version}
 %patch1 -p1 -b .Foption
 %patch2 -p1 -b .search
 %patch4 -p1 -b .time
@@ -51,16 +55,16 @@ make CC="gcc $RPM_OPT_FLAGS -D_GNU_SOURCE -
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOU
 %install
 rm -rf $RPM_BUILD_ROOT
 make DESTDIR=$RPM_BUILD_ROOT install
-mkdir -p $RPM_BUILD_ROOT/etc/profile.d
+mkdir -p $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
 install -p -c -m 755 %{SOURCE1} $RPM_BUILD_ROOT/%{_bindir}
-install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT/etc/profile.d
-install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT/etc/profile.d
-ls -la $RPM_BUILD_ROOT/etc/profile.d
+install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+install -p -c -m 644 %{SOURCE3} $RPM_BUILD_ROOT%{_sysconfdir}/profile.d
+ls -la $RPM_BUILD_ROOT%{_sysconfdir}/profile.d

 %files
 %defattr(-,root,root,-)
 %doc LICENSE
-/etc/profile.d/*
+%{_sysconfdir}/profile.d/*
 %{_bindir}/*
 %{_mandir}/man1/*
```

2.10.2. 转换标签和 Macro 定义

以下步骤演示了如何将传统 spec 文件中的标签和宏定义转换为 Software Collection spec 文件。

过程 2.1. 转换标签和宏定义

1. **您可以通过在 %_scl_prefix 之上定义 %scl_package macro 宏来更改根目录的位置：**

   ```
   %{?scl:%global _scl_prefix /opt/provider}
   ```

2. **在 spec 文件中添加 %scl_package 宏。将宏放在 spec 文件前，如下所示：**

   ```
   %{?scl:%scl_package package_name}
   ```

3. **当没有为 Software Collection 构建软件包时，建议您在 spec 文件中定义 %pkg_name 宏：**

   ```
   %{!?scl:%global pkg_name %{name}}
   ```

   因此，您可以使用 %pkg_name 宏定义在 spec 文件中需要的软件包的原始名称，然后可用于构建传统软件包和 Software Collection。

4. **更改 spec 文件中的 Name 标签，如下所示：**

```
Name: %{?scl_prefix}package_name
```

5. 如果您要与其他 Software Collection 软件包构建或链接，请为 **Requires** 和 **BuildRequires** 标签中的软件集合软件包名称添加前缀，如下所示：**%{?scl_prefix}**

```
Requires: %{?scl_prefix}ifconfig
```

当取决于软件包的系统版本时，您应该避免使用版本化的 **Requires** 或 **BuildRequires**。如果您需要依赖于系统更新的软件包，请考虑在 Software Collection 中包含该软件包，或者在系统软件包更新时记住重建您的 Software Collection。

6. 要检查所有基本 Software Collection 的软件包是否为主 metapackage 的依赖项，请在 spec 文件中的 **BuildRequires** 或 **Requires** 标签中添加以下宏：

```
%{?scl:Requires: %scl_runtime}
```

7. 使用 **Obsoletes** 为 **Conflicts**、**BuildConflicts**、**%{?scl_prefix}** 和标签添加前缀。这是为了确保 Software Collection 可用于将新软件包部署到旧的系统中，而无需指定软件包，例如 **Obsolete** 从基本系统安装中删除。例如：

```
Obsoletes: %{?scl_prefix}lesspipe < 1.0
```

8. 为 **Provides** 标签添加前缀 **%{?scl_prefix}**，如下例所示：

```
Provides: %{?scl_prefix}more
```

2.10.3. 转换子软件包

对于使用 **-n** 选项定义它们名称的任何子软件包，使用 **%{?scl_prefix}** 前缀，如下例所示：

```
%package -n %{?scl_prefix}more
```

前缀不仅适用于 **%package** 宏，也适用于 **%description** 和 **%files**。例如：

```
%description -n %{?scl_prefix}rubygems
RubyGems is the Ruby standard for publishing and managing third party
libraries.
```

如果子软件包需要主软件包，请确保也调整该子软件包中的 **Requires** 标签，以便标签使用 **%{?scl_prefix}%{pkg_name}**。例如：

```
Requires: %{?scl_prefix}%{pkg_name} = %{version}-%{release}
```

2.10.4. 转换 RPM 脚本

这部分论述了转换 RPM 脚本的常规规则，它们通常可在传统 spec 文件的 **%prep**、**%build**、**%install**、**%check**、**%pre** 和 **%post** 部分找到。

- 将所有出现的 **%name** 替换为 **%pkg_name**。最重要的是，这包括调整 **%setup** 宏。

- 调整 spec 文件的 %setup 部分中的 %prep 宏，以便宏可以在 Software Collection 环境中处理不同的软件包名称：

  ```
  %setup -q -n %{pkg_name}-%{version}
  ```

  请注意，%setup 宏是必需的，且您必须始终使用带有 -n 选项的宏才能成功构建软件集合。

- 如果您使用任何 %_root_ 宏指向系统文件系统层次结构，则必须使用这些宏的条件，以便您可以使用 spec 文件来构建传统软件包和 Software Collection。编辑宏，如下例所示：

  ```
  mkdir -p %{?scl:%_root_sysconfdir}%{?!scl:%_sysconfdir}
  ```

- 当构建依赖于其他 Software Collection 软件包的软件集合软件包时，务必要确保 scl 启用 功能链接或运行正确的二进制文件等。需要此示例之一是针对 Software Collection 库进行编译，或使用 Software Collection 中的解释器运行解释脚本。

  使用 %{?scl: 前缀嵌套脚本，如下例所示：

  ```
  %{?scl:scl enable %scl - << \EOF}
  set -e
  ruby example.rb
  RUBYOPT="-llib" ruby bar.rb
  # The rest of the script contents goes here.
  %{?scl:EOF}
  ```

  在脚本中指定 set -e 非常重要，无论脚本是否在 rpm shell 还是 scl 环境中执行，脚本的行为都是一致的。

- 请注意 Software Collection 软件包安装过程中执行的任何脚本，例如：

  - %pretrans, %pre,
  - %post, %postun, %posttrans,
  - %triggerin、%triggerun 和 %triggerpostun。

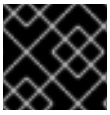
  如果您在这些脚本中使用 scl 启用 功能，建议您从空环境开始，以避免与基本系统安装有任何意外冲突。

  要做到这一点，在启用 Software Collection 前使用 env -i -，如下例所示：

  ```
  %posttrans
  %{?scl:env -i - scl enable %{scl} - << \EOF}
  %vagrant_plugin_register %{vagrant_plugin_name}
  %{?scl:EOF}
  ```

- RPM 脚本中找到的所有硬编码路径都必须替换为正确的宏。例如：将所有出现的 /usr/share 替换为 %{_datadir}。这是必要的，因为 $RPM_BUILD_ROOT 变量和 %{build_root} 宏不会被 scl 宏重新定位。

2.10.5. Software Collection Automatic Provides and Requires and Filtering Support



重要

本节中描述的功能在 Red Hat Enterprise Linux 6 中不提供。

Red Hat Enterprise Linux 7 中的 RPM 支持自动 Provides 和 Requires 和过滤。例如，对于所有 Python 库，RPM 会自动添加以下 Requires ：

```
Requires: python(abi) = (version)
```

如 第 2.10 节 "转换一个一致的 Spec 文件" 所述，您应该在转换传统 RPM 软件包时将此 Requires 与 %{?scl_prefix} 前缀：

```
Requires: %{?scl_prefix}python(abi) = (version))
```

请记住，搜索这些依赖项的脚本有时必须为您的 Software Collection 进行重写，因为原始 RPM 脚本不足可扩展，因此在某些情况下，过滤不可用。例如，要重写自动 Python Provides 和 Requires，请在 macros.%{scl}-config 宏文件中添加以下行：

```
%__python_provides /usr/lib/rpm/pythondeps-scl.sh --provides %{_scl_root} %{scl_prefix}
%__python_requires /usr/lib/rpm/pythondeps-scl.sh --requires %{_scl_root} %{scl_prefix}
```

/usr/lib/rpm/pythondeps-scl.sh 文件基于传统软件包中的 pythondeps.sh 文件并调整搜索路径。

如果您有需要调整的 Provides 或 Requires，例如 pkg_config Provides，则有两种方法：

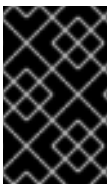
- *在 macros.%{scl}-config 宏中添加以下几行，使其适用于 Software Collection 中的所有软件包：*

```
%_use_internal_dependency_generator 0
%__deploop() while read FILE; do /usr/lib/rpm/rpmdeps -%{1} ${FILE}; done | /bin/sort -u
%__find_provides /bin/sh -c "%{?__filter_prov_cmd} %{__deploop P} %{?__filter_from_prov}"
%__find_requires /bin/sh -c "%{?__filter_req_cmd}  %{__deploop R} %{?__filter_from_req}"

# Handle pkgconfig's virtual Provides and Requires
%__filter_from_req | %{__sed} -e 's|pkgconfig|%{?scl_prefix}pkgconfig|g'
%__filter_from_prov | %{__sed} -e 's|pkgconfig|%{?scl_prefix}pkgconfig|g'
```

- *或者，在您要过滤 Provides 或 Requires 的每个 spec 文件中的标签定义后添加以下行：*

```
%{?scl:%filter_from_provides s|pkgconfig|%{?scl_prefix}pkgconfig|g}
%{?scl:%filter_from_requires s|pkgconfig|%{?scl_prefix}pkgconfig|g}
%{?scl:%filter_setup}
```



重要

在使用过滤器时，您需要注意您更改的自动依赖项。例如，如果传统软件包包含 Requires: pkgconfig(package_1) 和 Requires: pkgconfig(package_2)，且软件集合中仅包含 package_2，请确保不要为 package_1 过滤 Requires 标签。

2.10.6. Software Collection Macro 文件支持

在某些情况下，您可能需要通过 Software Collection 软件包提供宏文件。它们位于 %{?scl:%{_root_sysconfdir}}%{!?scl:%{_sysconfdir}}/rpm/ 目录中，对应于传统软件包的 /etc/rpm/ 目录。在提供宏文件时，请确保：

- 通过将 .%{scl} 附加到其名称来重命名宏文件，以便它们不会与基本系统安装中的文件冲突。
- 宏文件中的宏不能扩展，或者它们正在使用条件，如下例所示：

```
%__python2 %{_bindir}/python
%python2_sitelib %(%{?scl:scl enable %scl '}%{__python2} -c "from
distutils.sysconfig import get_python_lib; print(get_python_lib())"%{?scl:'})
```

再如，在有些情况下，您可能需要创建依赖于 Software Collection python26 的 Software Collection mypython。python26 Software Collection 定义 %{__python2} 宏，如上例中所示。此宏将评估为 /opt/provider/mypython/root/usr/bin/python2，但 python2 二进制文件仅在 python26 Software Collection (/opt/provider/python26/root/usr/bin/python2) 中提供。

要在 mypython Software Collection 环境中构建软件，请确保：

- macros.python.python26 宏文件，它是 python26-python-devel 软件包的一部分，包含以下行：

```
%__python26_python2 /opt/provider/python26/root/usr/bin/python2
```

- 以及 python26-build 子软件包中的宏文件，以及任何基于 Software Collection 的 build 子软件包包含以下行：

```
%scl_package_override() {%global __python2 %__python26_python2}
```

只有在存在来自相应 Software Collection 的构建子软件包时才重新定义 %{__python2} 宏，这通常意味着您要为该 Software Collection 构建软件。

2.10.7. Software Collection Shebang 支持

shebang 是脚本开头的一系列字符，用作解释器指令。shebang 由自动依赖项生成器处理，指向某个位置，可能在系统根文件系统中。

当自动依赖关系生成器处理 shebang 时，它会根据它们所指向的解释器添加依赖项。在 Software Collection 的角度来看，有两种 shebangs 类型：

#!/usr/bin/env example

这个 shebang 指示 /usr/bin/env 程序运行解释器。

自动依赖关系生成器将如预期对 /usr/bin/env 程序创建一个依赖项。

如果在 enable scriptlet 中正确定义了 $PATH 环境变量，则在 Software Collection 文件系统层次结构中找到 示例 解释器，如预期。

建议您重写 Software Collection 软件包中的 shebang，以便 shebang 指定软件集合文件系统层次结构中的解释器的完整路径。

#!/usr/bin/example

这个 shebang 指定解释器的直接路径。

自动依赖关系生成器将对位于 Software Collection 文件系统层次结构之外的 /usr/bin/example 解释器创建一个依赖项。但是，在为 Software Collection 构建软件包时，您通常希望对位于 Software Collection 文件系统层次结构中的 %{?_scl_root}/usr/bin/example 解释器创建依赖项。

请记住，即使您正确重新定义 $PATH 环境变量，这不会影响使用哪个解释器。始终使用软件集合文件系统层次结构之外的解释器的系统版本。在大多数情况下，这并不是必需的。

如果您使用这种类型的 shebang，并且您希望 shebang 在构建软件集合软件包时指向 Software Collection 文件系统层次结构，请使用类似如下的命令：

```
find %{buildroot} -type f | \
  xargs sed -i -e '1 s"^#!/usr/bin/example"#!%{?_scl_root}/usr/bin/example"'
```

其中 /usr/bin/example 是您要使用的解释器。

2.10.8. 在另一软件集合上制作 Software Collection Depend

要使一个 Software Collection 依赖于另一个 Software Collection 中的软件包，您需要调整依赖 Software Collection 的 spec 文件中的 **BuildRequires** 和 **Requires** 标签，以便这些标签正确定义依赖项。

例如，要定义两个名为 software_collection_1 和 software_collection_2 的 Software Collections 的依赖项，请将以下三行添加到应用程序的 spec 文件中：

```
BuildRequires: scl-utils-build
Requires: %scl_require software_collection_1
Requires: %scl_require software_collection_2
```

确保 spec 文件还包含 spec 文件前的 **%scl_package** 宏，例如：

```
%{?scl:%scl_package less}
```

请注意，**%scl_package** 宏必须包含在您的软件集合的每个 spec 文件中。

您还可以使用 **%scl_require_package** 宏定义特定软件集合中特定软件包的依赖项，如下例所示：

```
BuildRequires: scl-utils-build
Requires: %scl_require_package software_collection_1 package_name
```

2.11. 卸载所有 SOFTWARE COLLECTION 目录

请记住，**yum remove** 命令不会卸载删除 Software Collectionruntime 子软件包后删除的软件集合软件包和子软件包提供的目录。

为确保卸载所有目录，请使这些软件包和子软件包依赖于 runtime 子软件包。要做到这一点，请在每个软件包和子软件包的 spec 文件中添加以下行 **%scl_runtime** 宏：

```
%{?scl:Requires: %scl_runtime}
```

添加上面的行可确保这些软件包和子软件包提供的所有目录都会被正确地删除，只要运行时 子软件包不依赖于任何这些软件包和子软件包。

2.12. 构建 SOFTWARE COLLECTION

如果您正确地转换了软件集合的传统 spec 文件，如 第 2.10 节 “转换一个一致的 Spec 文件” 所述，您将能够在 Software Collection 和传统的构建根目录中构建生成的软件包。在传统构建 root 中构建转换的软件包将生成传统的基础系统 RPM 软件包，而构建包含 %{scl}-build 的 Software Collection 构建 root 将生成一个 Software Collection 软件包。

要在您的系统中构建 Software Collection，请运行以下命令：

```
rpmbuild -ba package.spec --define 'scl name'
```

上面显示的命令和用于构建传统软件包的标准命令(**rpmbuild -ba package.spec**)的区别在于，在构建 Software Collection 时必须将 **--define** 选项附加到 **rpmbuild** 命令中。

--define 选项定义 **scl** 宏，它使用 Software Collection spec 文件中配置的软件集合(软件包.**spec**)。

另外，要使用标准命令 **rpmbuild -ba package.spec** 来构建 Software Collection，请在**package.spec** 文件中指定以下内容：

```
BuildRequires: software_collection-build
```

其中 software_collection 是 Software Collection 的名称。

2.12.1. 在没有构建子软件包的情况下重建 Software Collection

当您要重建没有构建子软件包(software_collection-build)的软件集合时，您可以通过重建 Software Collection metapackage 来创建构建子软件包，从而避免使用 **rpmbuild -ba package.spec --define 'scl name'** 命令。

请注意，您需要在您的系统上安装了 scl-utils-build 软件包，否则使用 **rpmbuild** 命令重建 Software Collection metapackage 将失败。

有关 scl-utils-build 软件包的更多信息，请参阅 第 1.3 节 “启用对 Software Collections 的支持”。

2.12.2. 避免 debuginfo 文件冲突

当您构建指定了同一 **Source** 标签的两个 Software Collection 软件包（或传统的 RPM 软件包和 Software Collection 软件包），因此将源文件解包到 **%_builddir** 目录下的同一目录中，它们的 **debuginfo** 软件包将存在文件冲突。由于这些冲突，用户无法同时在同一系统中安装这两个软件包。

为避免这些文件冲突，必须更改其中一个软件包的 spec 文件，将其上游源解包到唯一的命名的根目录中。这会在 **%_builddir** 目录下的构建树中添加更多目录级别。这样，**debuginfo** 软件包生成脚本会生成 **debuginfo** 文件，该文件与其他 **debuginfo** 软件包中的文件不冲突。

要查看 diff 文件将原始 spec 文件与更改的 spec 文件进行比较，请参考以下示例：

```
--- a/tbb.spec
+++ b/tbb.spec
@@ -66,11 +66,13 @@ PDF documentation for the user of the Threading Building Block (TBB)
 C++ library.

 %prep
-%setup -q -n %{sourcebasename}
+%setup -q -c -n %{name}
+cd %{sourcebasename}
```

```
 %patch1 -p1
 %patch2 -p1

 %build
+cd %{sourcebasename}
 %{?scl:scl enable %{scl} - << \EOF}
 make %{?_smp_mflags} CXXFLAGS="$RPM_OPT_FLAGS" tbb_build_prefix=obj
 %{?scl:EOF}
@@ -81,6 +83,7 @@ done

 %install
 rm -rf $RPM_BUILD_ROOT
+cd %{sourcebasename}
 mkdir -p $RPM_BUILD_ROOT/%{_libdir}
 mkdir -p $RPM_BUILD_ROOT/%{_includedir}

@@ -108,20 +111,20 @@ done

 %files
 %defattr(-,root,root,-)
-%doc COPYING doc/Release_Notes.txt
+%doc %{sourcebasename}/COPYING %{sourcebasename}/doc/Release_Notes.txt
 %{_libdir}/*.so.2

 %files devel
 %defattr(-,root,root,-)
-%doc CHANGES
+%doc %{sourcebasename}/CHANGES
 %{_includedir}/tbb
 %{_libdir}/*.so
 %{_libdir}/pkgconfig/*.pc

 %files doc
 %defattr(-,root,root,-)
-%doc doc/Release_Notes.txt
-%doc doc/html
+%doc %{sourcebasename}/doc/Release_Notes.txt
+%doc %{sourcebasename}/doc/html

 %changelog
 * Wed Nov 13 2013 John Doe <jdoe@example.com> - 4.1-5.20130314
```

第 3 章 高级主题

本章讨论打包 Software Collections 的高级主题。

3.1. 通过 NFS 使用 SOFTWARE COLLECTIONS

在某些环境中，要求通常是以集中模型来分布应用程序和工具，而不是允许用户安装他们喜欢的应用程序或工具版本。这样，NFS 是挂载集中管理软件的通用方法。

您需要定义 Software Collection 宏 **nfsmountable** 以通过 NFS 使用 Software Collection。如果在构建 Software Collection 时定义了宏，则生成的 Software Collection 具有软件集合的 **/opt** 文件系统层次结构之外的状态文件和配置文件。这可让您以只读形式通过 NFS 挂载 **/opt** 文件系统层次结构。它还使状态文件和配置文件更易于管理。

如果您不需要通过 NFS 支持 Software Collections，请使用 **nfsmountable** 是可选的，但推荐使用。

要定义 **nfsmountable** 宏，请确保 Software Collection metapackage spec 文件包含以下行：

```
%global nfsmountable 1

%scl_package %scl
```

如上所示，必须在定义 **nfsmountable** 宏前定义 **%scl_package** 宏。这是因为 **%scl_package** 宏重新定义 **_sysconfdir**、**_sharedstatedir** 和 **_localstatedir** 宏，具体取决于是否定义了 **nfsmountable** 宏。下表中详细介绍了 **nfsmountable** 更改的值。

表 3.1. 更改 Software Collection Macros 的值

Macro	原始定义	原始定义的扩展值	更改了定义	为更改后的定义展开值
_sysconfdir	%{_scl_root}/etc	/opt/provider/%{scl}/root/etc	%{_root_sysconfdir}%{_scl_prefix}/%{scl}	/etc/opt/provider/%{scl}
_sharedstatedir	%{_scl_root}/var/lib	/opt/provider/%{scl}/root/var/lib	%{_root_localstatedir}%{_scl_prefix}/%{scl}/lib	/var/opt/provider/%{scl}/lib
_localstatedir	%{_scl_root}/var	/opt/provider/%{scl}/root/var	%{_root_localstatedir}%{_scl_prefix}/%{scl}	/var/opt/provider/%{scl}

3.1.1. 更改了目录结构和文件所有权

nfsmountable 宏还会影响 **scl_install** 和 **scl_files** 宏创建目录结构，并在运行 **rpmbuild** 命令时设置文件所有权。

例如，名为 software_collection 的软件集合的目录结构，其定义的 **nfsmountable** 宏如下所示：

```
$ rpmbuild -ba software_collection.spec --define 'scl software_collection'
```

```
...
$ rpm -qlp software_collection-runtime-1-1.el6.x86_64
/etc/opt/provider/software_collection
/etc/opt/provider/software_collection/X11
/etc/opt/provider/software_collection/X11/applnk
/etc/opt/provider/software_collection/X11/fontpath.d
...
/opt/provider/software_collection/root/usr/src
/opt/provider/software_collection/root/usr/src/debug
/opt/provider/software_collection/root/usr/src/kernels
/opt/provider/software_collection/root/usr/tmp
/var/opt/provider/software_collection
/var/opt/provider/software_collection/cache
/var/opt/provider/software_collection/db
/var/opt/provider/software_collection/empty
...
```

3.1.2. 注册和取消注册 Software Collections

如果 Software Collection 通过 NFS 共享但没有在本地安装，则需要通过注册该 Software Collection 来了解它。

运行 **scl register** 命令注册 Software Collection：

```
$ scl register /opt/provider/software_collection
```

其中 /opt/provider/software_collection 是您要注册的 Software Collection 的文件系统层次结构的绝对路径。该路径的目录必须包含 enable scriptlet 和 root/ 目录，才能被视为有效的 Software Collection 文件系统层次结构。

取消注册 Software Collection 是一个反向操作，当您不再希望 scl 工具了解注册的软件集合时，您可以执行的操作。

在运行 scl 命令时调用 deregister scriptet 来取消注册软件集合：

```
$ scl deregister software_collection
```

其中 software_collection 是您要取消注册的软件集合的名称。

3.1.2.1. 在 Software Collection Metapackage 中使用(de) register Scriptlets

您可以在 Software Collection metapackage 中指定(de) register scriptlets 与如何指定 enable scriptlets 类似。当指定 scriptets 时，请记住将它们显式包括在 metapackage spec 文件的 %file 部分。

有关指定(de) register scriptets 的示例，请参见以下示例代码：

```
%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
# Contents of the enable scriptlet goes here
...
EOF
```

```
cat >> %{buildroot}%{_scl_scripts}/register << EOF
# Contents of the register scriptlet goes here
...
EOF

cat >> %{buildroot}%{_scl_scripts}/deregister << EOF
# Contents of the deregister scriptlet goes here
...
EOF
...
%files runtime -f filelist
%scl_files
%{_scl_scripts}/register
%{_scl_scripts}/deregister
```

在 register scriptlet 中，您可以选择性地指定注册 Software Collection 时要运行的命令，例如：要在 /etc/opt/ 或 /var/opt/ 中创建文件的命令。

3.2. 将 SOFTWARE COLLECTION SCRIPTLET 转换为环境模块

环境模块允许您通过动态修改 shell 环境来管理应用程序的不同版本。要将 Software Collection 与环境模块系统搭配使用，请将 Software Collection 的 **enable** scriptlet 转换为带有脚本 **/usr/share/Modules/bin/createmodule.sh** 的环境模块。

过程 3.1. 将启用 scriptlet 转换为环境模块

1. 确定在您的系统中安装了 environment-modules 软件包：

   ```
   # yum install environment-modules
   ```

2. 运行 **/usr/share/Modules/bin/createmodule.sh** 脚本，将 Software Collection 的 **enable** scriptlet 转换为环境模块：

   ```
   /usr/share/Modules/bin/createmodule.sh /path/to/enable/scriptlet
   ```

   使用您要转换的 **enable** scriptlet 的文件路径替换 /path/to/enable/scriptlet。

3. 在 Software Collection metapackage 的 **%pre** 部分添加相同的命令 **/usr/share/Modules/bin/createmodule.sh /path/to/enable/scriptlet**，在生成您的 **enable** scriptlet 的代码下。

   如果您在其中一个 Software Collection 软件包中将 **enable** scriptlet 打包为一个文件，请在 **%post** 部分添加命令 **/usr/share/Modules/bin/createmodule.sh /path/to/enable/scriptlet**。

有关环境模块的更多信息，请参阅 module(1) 手册页。

3.3. 提供 SYSPATHS SUBPACKAGES

要使用 Software Collection 的软件包，用户需要执行与使用传统 RPM 软件包时不同的某些任务。例如，它们需要使用 **scl enable** 调用，它更改环境变量，如 **PATH** 或 **LD_LIBRARY_PATH**，以便可以找到在其他位置安装的二进制文件。用户还需要将替代名称用于 systemd 服务。有些脚本也可以使用完整路径调用二进制文件，如 **/usr/bin/mysql**，因此这些脚本可能无法用于 Software Collection。

解决上述问题的建议解决方案是使用 sypaths 子软件包。基本的概念是允许用户在不影响基本系统安装的情况下消耗同一软件包的不同版本，但可选择使用 Software Collection 软件包，就像它们是传统 RPM 软件包一样，使软件集合更易于使用。

可选的 sypaths 子软件包（如 rh-mariadb102-syspaths）提供安装到标准路径中的 shell 打包程序和符号链接（通常为 /usr/bin/）。这意味着，通过选择安装 sypaths 子软件包，用户会意外地更改基本系统安装，从而使 sypaths 子软件包通常适合不需要安装和运行同一软件包的多个版本。这在使用数据库时尤其如此。

使用 sypaths 子软件包可避免调整 Software Collection 软件包中脚本的需求，以便这些脚本更易于使用。请记住，sypaths 子软件包与基本系统安装中的软件包有冲突，因此无法与 sypaths 子软件包一起安装传统的软件包。如果这是问题，请考虑使用基于容器的技术将 sypaths 子软件包与基本系统安装隔离。

3.3.1. 命名 sypaths Subpackages

对于使用 sypaths 子软件包概念的每个软件集合，通常会提供多个 sypaths 子软件包。每个软件包都提供 sypaths 子软件包，该文件可以通过打包程序或符号链接提供。

在这里，有一个名为 software_collection_1-sypaths 的 Software Collection metapackage 子软件包的子软件包，其中 software_collection_1 是 Software Collection 的名称。software_collection_1-sypaths 子软件包需要其他 sypaths 子软件包包含 Software Collection。安装 software_collection_1-sypaths 子软件包会导致安装所有其他 sypaths 软件包。

例如，如果要为 software_collection_1-package_1 软件包中包含的二进制文件 **binary_1** 包含，以及一个包含在 software_collection_1-package_2 软件包中的二进制文件 **binary_2**，然后在 software_collection_1 Software Collection 中创建以下三个 sypaths 子软件包：

```
software_collection_1-sypaths
software_collection_1-package_1-sypaths
software_collection_1-package_2-sypaths
```

3.3.2. sypaths Subpackages 中包含的文件

适用于 sypaths 子软件包中的文件是用户与之交互的二进制文件的可执行 shell 打包程序。

以下是 software_collection_1 中包含的二进制文件 **binary_1** 的示例，位于 **/opt/rh/software_collection_1/root/usr/bin/binary_1** 中：

```
#!/bin/bash
source scl_source enable software_collection_1
exec "/opt/rh/software_collection_1/root/usr/bin/binary_1" "$@"
```

当您在 **/usr/bin/binary_1** 中安装此打包程序并使它可执行时，用户只需运行 **binary_1** 命令，而无需将其前缀为 **scl enable software_collection_1**。安装在 **/usr/bin/** 中的打包程序设置正确的环境，并执行位于 **/opt/provider/%{scl}** 文件系统层次结构的目标二进制文件。

3.3.3. sypaths Wrappers 的限制

sypaths 包装程序是 shell 脚本，意味着用户无法像目标二进制文件一样使用打包程序执行每个可能的任务。例如，当使用 gdb 调试二进制文件时，指向 **/opt/provider/%{scl}** 文件系统层次结构的完整路径，因为 gdb 无法使用打包程序 shell 脚本。

3.3.4. sypaths 子软件包中的符号链接

除了二进制文件的打包程序外，还有一些适合在 /opt、/etc/opt/ 或 /var/opt/ 目录之外安装的文件，因此可由 syspaths 子软件包提供。例如，您可以使数据库文件的路径（通常位于 /var/opt/provider/%{scl} 下）更轻松地发现位于 /var/lib/ 中的符号链接。但是，对于某些符号链接，最好不要在其原始名称下将其安装到 /var/lib/ 中，因为它们可能与基础系统安装的传统 RPM 软件包的名称冲突。

最好将符号链接 /var/lib/software_collection_1-original_name 命名为类似。对于日志文件，建议名称为 /var/log/software_collection_1-original_name 或类似。请记住，名称本身并不重要，此处的设计目标是使这些文件在 /var/lib/ 或 /var/log/ 目录下轻松找到。

同样适用于配置文件，目标是使符号链接在 /etc 目录下轻松发现。

3.3.5. 没有前缀的服务

systemd 和 SysV init 服务是用户与守护进程服务交互的示例。通常，用户在启动服务时不需要在命令中包含 scl enable，因为服务是由设计在干净的环境中启动的。但是，用户仍需要使用正确的服务名称，通常以 Software Collection 名称作为前缀（如 rh-mariadb102-mariadb）。

syspaths 子软件包允许用户使用服务的传统名称，如 mariadb、mongod 或 postgresql，如果安装了适当的 syspaths 子软件包。为达到此目的，请创建一个符号链接，而不在符号链接名称中包含软件集合名称，指向传统服务文件。

例如，由 /etc/rc.d/init.d/software_collection_1- service_1 文件提供的 software_collection_1 Software Collection 中的服务 service_1 可以作为 service_1 访问：

```
/etc/rc.d/init.d/service_1 -> /etc/rc.d/init.d/software_collection_1-service_1
```

或者，在使用 systemd 单元文件时：

```
/usr/lib/systemd/system/service_1 -> /usr/lib/systemd/system/software_collection_1-service_1
```

3.4. 在 SOFTWARE COLLECTIONS 中管理服务

在打包软件集合时，请确保用户可以直接管理 Software Collection 提供的任何服务（守护进程），或使用系统默认工具（如 service 或 chkconfig on Red Hat Enterprise Linux 6）或 systemctl on Red Hat Enterprise Linux 7。

对于 Red Hat Enterprise Linux 6 中的 Software Collections，请确保调整 spec 文件的 %install 部分，以避免与 Software Collection 一部分的服务的系统版本冲突：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/rc.d/init.d/%{?scl_prefix}service_name
```

使用服务的实际名称替换 service_name。

对于 Red Hat Enterprise Linux 7 中的 Software Collections，请按如下所示调整 spec 文件的 %install 部分：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{_unitdir}/%{?
scl_prefix}service_name.service
```

使用这个配置，您可以引用 Software Collection 中包含的服务版本，如下所示：

-

```
%{?scl_prefix}service_name
```

请记住，没有环境变量从用户的环境传播到 SysV init 脚本（或 Red Hat Enterprise Linux 7 上的 systemd 服务文件）。这是预期的，并确保服务始终在干净环境中启动。但是，这需要您为由 SysV 初始化脚本（或 systemd 服务文件）运行的进程正确设置 Software Collection 环境。

3.4.1. 为服务配置环境

建议您使您要为服务启用的软件集合进行配置。本节中的指示演示了如何使软件集合配置名为 software_collection。

过程 3.2. 为 Red Hat Enterprise Linux 6 上的服务配置环境

1. 在 /opt/provider/software_collection/service-environment 中创建包含以下内容的配置文件：

   ```
   [SCLNAME]_SCLS_ENABLED="software_collection"
   ```

   将 SCLNAME 替换为软件集合的唯一标识符，例如，您的 Software Collection 使用大写字母编写的名称。

   将 software_collection 替换为 %scl_name 宏定义的 Software Collection 的名称。

2. 在 SysV init 脚本的开头添加以下行：

   ```
   source /opt/provider/software_collection/service-environment
   ```

3. 在 SysV init 脚本中，确定运行位于 /opt/provider/ 文件系统层次结构中的二进制文件的命令。使用 scl enable $[SCLNAME]_SCLS_ENABLED 为这些命令添加前缀，类似于在 Software Collection 环境中运行命令。

   例如，替换以下行：

   ```
   /usr/bin/daemon_binary --argument-1 --argument-2
   ```

   使用：

   ```
   scl enable $[SCLNAME]_SCLS_ENABLED -- /usr/bin/daemon_binary --argument-1 --argument-2
   ```

4. 有些命令（如 su 或 runuser ）也清除环境变量。因此，如果在 SysV init 脚本中使用这些命令，请在运行这些命令后再次启用您的 Software Collection。

   例如，替换以下行：

   ```
   su - user_name -c '/usr/bin/daemon_binary --argument-1 --argument-2'
   ```

   使用：

   ```
   su - user_name -c '\
     source /opt/provider/software_collection/service-environment \
     scl enable $SCLNAME_SCLS_ENABLED -- /usr/bin/daemon_binary --argument-1 --argument-2'
   ```

过程 3.3. 为 Red Hat Enterprise Linux 7 中的服务配置环境

1. 在 /opt/provider/software_collection/service-environment 中创建包含以下内容的配置文件：

   ```
   [SCLNAME]_SCLS_ENABLED="software_collection"
   ```

   将 SCLNAME 替换为软件集合的唯一标识符，例如，您的 Software Collection 使用大写字母编写的名称。

   将 software_collection 替换为 %scl_name 宏定义的 Software Collection 的名称。

2. 在 systemd 服务文件中添加以下行以载入配置文件：

   ```
   EnvironmentFile=/opt/provider/software_collection/service-environment
   ```

3. 在 systemd 服务文件中，使用 scl enable $[SCLNAME]_SCLS_ENABLED，与在 Software Collection 环境中运行命令的 ExecStartPre、ExecStart 和类似的指令中加上前缀：

   ```
   ExecStartPre=/usr/bin/scl enable $[SCLNAME]_SCLS_ENABLED --
   /opt/provider/software_collection/root/usr/bin/daemon_helper_binary --argument-1 --
   argument-2
   ExecStart=/usr/bin/scl enable $[SCLNAME]_SCLS_ENABLED --
   /opt/provider/software_collection/root/usr/bin/daemon_binary --argument-1 --
   argument-2
   ```

警告

前缀带有 scl enable ... 的 scl 的命令将阻止服务转换到 SELinux 策略所表示的目标 SELinux 上下文无效。如果使用 SELinux 限制该服务，systemd 服务文件需要直接执行二进制文件。如果二进制文件与作为 Software Collection 一部分的共享库相关联，DT_RUNPATH 属性可帮助在运行时访问这些共享库，而无需使用 scl enable ... wrapper。请参阅 第 3.15.1 节 “Red Hat Enterprise Linux 7 中的 SELinux 支持” 了解更多信息。

3.5. SOFTWARE COLLECTION LIBRARY 支持

如果您只分发要在 Software Collection 环境中使用的库，或者除了系统中可用的库外，更新 enable scriptlet 中的 LD_LIBRARY_PATH 环境变量，如下所示：

```
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
```

配置可确保，如果启用了 Software Collection，则 Software Collection 中的库版本优先于系统上可用的库版本。

注意

如果您在 Software Collection 中发布私有共享库，请考虑使用 DT_RUNPATH 属性而不是 LD_LIBRARY_PATH 环境变量，以便在 Software Collection 环境中访问私有共享库。

3.5.1. 使用 Software Collection 之外的库

如果您要在 Software Collection 环境之外分发要使用的库，您可以使用目录 /etc/ld.so.conf.d/ 来实现这一目的。

警告

对于系统上已可用的库，不要使用 /etc/ld.so.conf.d/。只有在系统中不可用的库才建议使用 /etc/ld.so.conf.d/，否则 Software Collection 中的库版本可能会优先于库的系统版本。这可能会导致应用程序的系统版本不必要的行为，包括意外终止和数据丢失。

过程 3.4. 将 /etc/ld.so.conf.d/ 用于 Software Collection 中的库

1. 创建名为 %{?scl_prefix}libs.conf 的文件并相应地调整 spec 文件配置：

```
SOURCE2: %{?scl_prefix}libs.conf
```

2. 在 %{?scl_prefix}libs.conf 文件中，包括一个与 Software Collection 关联的库版本所在的目录列表。例如：

```
/opt/provider/software_collection_1/root/usr/lib64/
```

在上例中，作为 Software Collection software_collection_1 一部分的 /usr/lib64/ 目录包含在列表中。

3. 编辑 spec 文件的 %install 部分，因此 %{?scl_prefix}libs.conf 文件已安装，如下所示：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/ld.so.conf.d/
```

3.5.2. 使用 Software Collection Name 为 Library Major soname 前缀

当使用 Software Collection 中包含的库时，请始终记住，具有相同主要 soname 的库可作为基本系统安装的一部分在系统中可用。因此，不要忘记在根据 Software Collection 中包含的库构建应用程序时使用 scl enable 命令。如果不这样做可能会导致应用程序在不正确的环境中执行，并与库的系统版本相关联。

警告

请记住，在不正确的环境中（例如在系统环境中而不是 Software Collection 环境中）中执行应用程序，并将应用程序链接到不正确的库可能会导致应用程序的行为（包括意外终止和数据丢失）。

要确保您的应用程序没有与不正确的库相关联，即使 LD_LIBRARY_PATH 环境变量没有正确设置，请更改 Software Collection 中包含的库的主要 soname。更改主 soname 的建议方法是使用 Software Collection 名称为主 soname 版本号添加前缀。

以下是带有 mysql55- 前缀的 MySQL 客户端库示例：

```
$ rpm -ql mysql55-mysql-libs | grep 'lib.*so'
/opt/provider/mysql55/root/usr/lib64/mysql/libmysqlclient.so.mysql55-18
/opt/provider/mysql55/root/usr/lib64/mysql/libmysqlclient.so.mysql55-18.0.0
```

在同一系统中，以下列出了 MySQL 客户端库的系统版本：

```
$ rpm -ql mysql-libs | grep 'lib.*so'
/usr/lib64/mysql/libmysqlclient.so.18
/usr/lib64/mysql/libmysqlclient.so.18.0.0
```

rpmbuild 工具为包含版本共享库的软件包生成自动 Provides 标签。如果您没有按照上述描述为 soname 前缀，那么如果 mysql 软件包为 libmysqlclient.so.18()(64bit)，则为 Provides 的示例。在这个 Provides 中，RPM 可以选择不正确的 RPM 软件包，从而导致应用程序缺少要求。

如果您按照上述描述为 soname 加上前缀，那么如果 mysql 为 Provides，则生成的 libmysqlclient.so.mysql55-18()(64bit) 示例是。在这个 Provides 中，RPM 选择正确的 RPM 依赖项，并满足应用程序的要求。

通常，除非绝对需要，否则 Software Collection 软件包不应该提供已经由基本系统安装的软件包提供的任何符号。规则的一个例外是在使用基本系统安装的软件包中的符号时。

3.5.3. Red Hat Enterprise Linux 7 中的 Software Collection Library 支持

为 Red Hat Enterprise Linux 7 构建 Software Collection 时，请使用 %__provides_exclude_from 宏以防止扫描某些文件以自动生成 RPM 符号。

例如，为了避免扫描 %{_libdir} 目录中的 .so 文件，请在 Software Collection spec 文件中的 BuildRequires 或 Requires 标签前面添加以下行：

```
%if %{?scl:1}%{!?scl:0}
  # Do not scan .so files in %{_libdir}
  %global __provides_exclude_from ^%{_libdir}/.*.so.*$
%endif
```

功能是自动 Provides 和 Requires 的 RPM 支持的一部分，如需更多信息，请参阅 第 2.10.5 节 “Software Collection Automatic Provides and Requires and Filtering Support”。

3.6. SOFTWARE COLLECTION .PC 文件支持

.pc 文件是 pkg-config 程序使用的特殊元数据文件，用于存储系统上可用库的信息。

如果您发布只在 Software Collection 环境中使用的 .pc 文件，或者除了系统上安装的 .pc 文件外，更新 PKG_CONFIG_PATH 环境变量。根据 .pc 文件中定义的内容，更新 PKG_CONFIG_PATH 宏的 %{_libdir} 环境变量（该宏扩展至库目录，通常为 /usr/lib/ 或 /usr/lib64/），或 %{_datadir} 宏（扩展至共享目录，通常为 /usr/share/）。

如果在 .pc 文件中定义库目录，调整 Software Collection spec 文件的 PKG_CONFIG_PATH 部分来更新 %install 环境变量，如下所示：

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PKG_CONFIG_PATH="%
```

```
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF
```

如果在 .pc 文件中定义共享目录，请通过调整 Software Collection spec 文件的 PKG_CONFIG_PATH 部分来更新 %install 环境变量，如下所示：

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PKG_CONFIG_PATH="%
{_datadir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
EOF
```

以上两个示例都配置 enable scriptlet，以便在启用了 Software Collection 时，它确保 Software Collection 中的 .pc 文件优先于系统上可用的 .pc 文件。

Software Collection 可以提供一个打包程序脚本，它可以被系统可见来启用 Software Collection，例如在 /usr/bin/ 目录中。在这种情况下，请确保 .pc 文件对系统可见，即使禁用了 Software Collection。

要允许您的系统使用禁用的软件集合中的 .pc 文件，请使用与 Software Collection 关联的 .pc 文件的路径更新 PKG_CONFIG_PATH 环境变量。根据 .pc 文件中定义的内容，更新 PKG_CONFIG_PATH 宏的 %{_libdir} 环境变量（扩展至库目录），或 %{_datadir} 宏（扩展至共享目录）。

过程 3.5. 为 %{_libdir} 更新 PKG_CONFIG_PATH 环境变量

1. 要为 PKG_CONFIG_PATH 宏更新 %{_libdir} 环境变量，请创建一个自定义脚本 /etc/profile.d/name.sh。当系统上启动 shell 时，脚本会被预加载。

   例如，创建以下文件：

   ```
   %{?scl_prefix}pc-libdir.sh
   ```

2. 使用 pc-libdir.sh 简短脚本修改 PKG_CONFIG_PATH 变量以引用您的 .pc 文件：

   ```
   export PKG_CONFIG_PATH="%
   {_libdir}/pkgconfig:/opt/provider/software_collection/path/to/your/pc_files"
   ```

3. 将文件添加到 Software Collection 软件包的 spec 文件中：

```
SOURCE2: %{?scl_prefix}pc-libdir.sh
```

4. 通过调整 Software Collection 软件包 spec 文件的 %install 部分将此文件安装到系统 /etc/profile.d/ 目录中：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

过程 3.6. 为 %{_datadir} 更新 PKG_CONFIG_PATH 环境变量

1. 要为 PKG_CONFIG_PATH 宏更新 %{_datadir} 环境变量，请创建一个自定义脚本 /etc/profile.d/name.sh。当系统上启动 shell 时，脚本会被预加载。

例如，创建以下文件：

```
%{?scl_prefix}pc-datadir.sh
```

2. 使用 pc-datadir.sh 简短脚本修改 PKG_CONFIG_PATH 变量以引用您的 .pc 文件：

```
export PKG_CONFIG_PATH="%
{_datadir}/pkgconfig:/opt/provider/software_collection/path/to/your/pc_files"
```

3. 将文件添加到 Software Collection 软件包的 spec 文件中：

```
SOURCE2: %{?scl_prefix}pc-datadir.sh
```

4. 通过调整 Software Collection 软件包 spec 文件的 %install 部分将此文件安装到系统 /etc/profile.d/ 目录中：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

3.7. SOFTWARE COLLECTION MANPATH 支持

要允许系统中的 man 命令显示启用的软件集合中的 man page，使用与 Software Collection 关联的 man page 的路径更新 MANPATH 环境变量。

要更新 MANPATH 环境变量，请将以下内容添加到 Software Collection spec 文件的 %install 部分：

```
%install
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export MANPATH="%{_mandir}:\${MANPATH:-}"
EOF
```

这将配置 enable scriptlet 以更新 MANPATH 环境变量。只要没有启用 Software Collection，与 Software Collection 关联的 man page 将无法看到。

Software Collection 可以提供一个打包程序脚本，它可以被系统可见来启用 Software Collection，例如在 /usr/bin/ 目录中。在这种情况下，请确保 man page 对系统可见，即使 Software Collection 被禁用。

要允许系统中的 man 命令显示禁用的软件集合中的 man page，请使用与 Software Collection 关联的 man page 的路径更新 MANPATH 环境变量。

过程 3.7. 更新禁用的软件集合的 MANPATH 环境变量

1. 要更新 MANPATH 环境变量，请创建一个自定义脚本 /etc/profile.d/name.sh。当系统上启动 shell 时，脚本会被预加载。

   例如，创建以下文件：

   ```
   %{?scl_prefix}manpage.sh
   ```

2. 使用修改 MANPATH 变量的 manpage.sh 简短脚本来引用您的 man 路径目录：

   ```
   export MANPATH="/opt/provider/software_collection/path/to/your/man_pages:${MANPATH}"
   ```

3. 将文件添加到 Software Collection 软件包的 spec 文件中：

   ```
   SOURCE2: %{?scl_prefix}manpage.sh
   ```

4. 通过调整 Software Collection 软件包 spec 文件的 %install 部分将此文件安装到系统 /etc/profile.d/ 目录中：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

3.8. SOFTWARE COLLECTION CRONJOB 支持

使用 Software Collection，您可以使用专用服务或 cronjobs 在系统上运行定期的任务。如果要使用专用服务，请参阅 第 3.4 节 “在 Software Collections 中管理服务” 如何在 Software Collection 环境中使用 initscripts。

过程 3.8. 使用 cronjobs 运行定期任务

1. 要使用 cronjobs 运行定期任务，请将 Software Collection 的 crontab 文件放在具有 Software Collection 的名称的 /etc/cron.d/ 目录中。

   例如，创建以下文件：

   ```
   %{?scl_prefix}crontab
   ```

2. 确保 crontab 文件的内容遵循标准 crontab 文件格式，如下例所示：

   ```
   0 1 * * Sun root scl enable software_collection
   '/opt/provider/software_collection/root/usr/bin/cron_job_name'
   ```

   其中 software_collection 是 Software Collection 的名称，/opt/provider/software_collection/root/usr/bin/cron_job_name 是您要定期运行的命令。

3. 将文件添加到 Software Collection 软件包的 spec 文件中：

   ```
   SOURCE2: %{?scl_prefix}crontab
   ```

4. 通过调整 Software Collection 软件包 spec 文件的 %install 部分，将文件安装到系统目录 /etc/cron.d/ 中：

   ```
   %install
   install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
   scl:%_sysconfdir}/cron.d/
   ```

3.9. SOFTWARE COLLECTION 日志文件支持

默认情况下，在 Software Collection 中的程序会在 /opt/provider/%{scl}/root/var/log/ 目录中创建日志文件。

要使日志文件更易访问且更易于管理，建议您使用重新定义 nfsmountable 宏的 _localstatedir 宏。这会导致在 /var/opt/供应商/%{scl}/log/ 目录下创建日志文件，位于 /opt/供应商/%{scl} 文件系统层次结构之外。

例如，服务 mydaemon 通常将其日志文件存储在基本系统安装的 /var/log/mydaemon/mydaemond.log 中。当 mydaemon 打包为 software_collection Software Collection 并定义了 nfsmountable 宏时，在 software_collection 中日志文件的路径如下：

```
/var/opt/provider/software_collection/log/mydaemon/mydaemond.log
```

有关使用 nfsmountable 宏的更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。

3.10. SOFTWARE COLLECTION LOGROTATE 支持

使用 Software Collection 或与 Software Collection 关联的应用程序，您可以使用 logrotate 程序管理日志文件。

过程 3.9. 使用 logrotate 管理日志文件

1. 要使用 logrotate 管理日志文件，请将 Software Collection 的自定义 logrotate 文件放在 logrotate 作业 /etc/ logrotate.d/ 的系统目录中。

   例如，创建以下文件：

   ```
   %{?scl_prefix}logrotate
   ```

2. 确保 logrotate 文件的内容采用标准的 logrotate 文件格式，如下所示：

   ```
   /opt/provider/software_collection/var/log/your_application_name.log {
       missingok
       notifempty
       size 30k
   ```

```
    yearly
    create 0600 root root
}
```

3. 将文件添加到 Software Collection 软件包的 spec 文件中：

```
SOURCE2: %{?scl_prefix}logrotate
```

4. 通过调整 Software Collection 软件包 spec 文件的 %install 部分，将文件安装到系统目录 /etc/logrotate.d/ 中：

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:%_sysconfdir}/logrotate.d/
```

3.11. 软件集合 /VAR/RUN/ 文件支持

PID 文件是通常位于 /var/run/package_name/ 目录下的文件示例。将 PID 文件打包到 Software Collection 中时，建议您使用 nfsmountable 宏并将 PID 文件存储在以下目录中：

```
/var/run/software_collection-package_name/
```

其中 software_collection 是 Software Collection 的名称，package_name 是 Software Collection 中包含的软件包的名称。

按照这种命名约定避免文件与基本系统安装冲突，而您的 Software Collection 可以使用 /var/run/ 功能，例如 PID 文件的 tmpfs 文件系统。

有关使用 nfsmountable 宏的更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。

3.12. SOFTWARE COLLECTION LOCK 文件支持

默认情况下，打包到 Software Collection 的程序会在 /opt/provider/%{scl}/root/var/lock/ 目录中创建锁定文件。

要使锁定文件更易于访问且更易于管理，建议您使用重新定义 nfsmountable 宏的 _localstatedir 宏。这会导致在 /var/opt/供应商/%{scl}/lock/ 目录下创建的文件，位于 /opt/provider/%{scl} 文件系统层次

结构之外。

如果软件集合中的应用程序或服务在 /var/opt/供应商/%{scl}/lock/ 目录下写入锁定，那么这些应用程序和服务可以与系统版本同时运行（当您 Software Collection 应用程序和服务的资源不会与系统版本冲突时）。

例如，在基本系统安装的 /var/lock/ 目录中通常会创建一个锁定文件 mylockfile.lock。如果锁定文件是 software_collection Software Collection 的一部分，并且定义了 nfsmountable 宏，则 software_collection 中的锁定文件的路径如下：

```
/var/opt/provider/software_collection/lock/mylockfile.lock
```

有关使用 nfsmountable 宏的更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。

防止程序当前运行

如果要防止软件集合的应用程序或服务在运行相应应用程序或服务的系统版本运行时运行，请确保您的应用程序或服务需要锁定，请将锁定写入系统目录 /var/lock/。这样，您的应用程序或服务的锁定文件不会被覆盖。锁定文件不会被重命名，名称与系统版本相同。

3.12.1. Software Collection SysV init 锁定文件支持

当一个服务由初始化脚本启动时，会在 /var/lock/subsys/ 目录中与初始化脚本的名称相同。如 第 3.4 节 “在 Software Collections 中管理服务” 所述，服务名称包含一个 Software Collection 前缀。对 /var/lock/subsys/ 中的文件使用相同的命名约定，以确保锁定文件名与基本系统安装不冲突。

3.13. 软件集合配置文件支持

默认情况下，Software Collection 中的配置文件存储在 /opt/provider/%{scl} 文件系统中。

要使配置文件更易于访问且更易于管理，建议您使用重新定义 nfsmountable 宏的 _sysconfdir 宏。这会导致在 /etc/opt/供应商/%{scl}/ 目录下创建配置文件，位于 /opt/provider/%{scl} 文件系统层次结构之外。

例如，配置文件 example.conf 通常存储在基本系统安装的 /etc 目录中。如果配置文件是 software_collection Software Collection 的一部分，并且定义了 nfsmountable 宏，则 software_collection 中的配置文件的路径如下：

```
/etc/opt/provider/software_collection/example.conf
```

有关使用 nfsmountable 宏的更多信息，请参阅 第 3.1 节 “通过 NFS 使用 Software Collections”。

3.14. SOFTWARE COLLECTION 内核模块支持

由于 Linux 内核模块通常与 Linux 内核的特定版本关联，所以当您将内核模块打包到 Software Collection 中时，您必须小心。这是因为，如果安装了 Linux 内核的更新版本，Red Hat Enterprise Linux 中的软件包管理系统不会自动更新或安装内核模块的更新版本。要将内核模块打包到 Software Collection 中，请查看以下建议。确保：

1. 内核模块软件包的名称包括内核版本，
2. 标签 Requires （可在您的内核模块 spec 文件中找到）包括内核版本和修订版本（格式为 kernel-version-revision）。

3.15. SOFTWARE COLLECTION SELINUX 支持

由于 Software Collections 设计为在备用目录中安装 Software Collection 软件包，所以请设置必要的 SELinux 标签，以便 SELinux 了解备用目录。

如果 Software Collection 软件包的文件系统层次结构代表相应传统软件包的文件系统层次结构，您可以运行 semanage fcontext 和 restorecon 命令来设置 SELinux 标签。

例如，如果 Software Collection 软件包中的 /opt/provider/software_collection_1/root/usr / 目录代表您的传统软件包的 /usr/ 目录，请设置 SELinux 标签，如下所示：

```
semanage fcontext -a -e /usr /opt/provider/software_collection_1/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/root/usr
```

以上命令确保 /opt/provider/software_collection_1/root/usr/ 目录中的所有目录和文件都被 SELinux 标记，就像它们位于 /usr/ 目录中一样。

3.15.1. Red Hat Enterprise Linux 7 中的 SELinux 支持

当为 Red Hat Enterprise Linux 7 打包 Software Collection 时，将以下命令添加到 Software Collection metapackage 中的 %post 部分来设置 SELinux 标签：

```
semanage fcontext -a -e /usr /opt/provider/software_collection_1/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/root/usr
```

```
selinuxenabled && load_policy || :
```

最后一个命令可确保正确加载新创建的 SELinux 策略，并且软件集合中软件包安装的文件也会使用正确的 SELinux 上下文创建。通过在 metapackage 中使用此命令，您不需要在 Software Collection 中的所有软件包中包含 restorecon 命令。

请注意，semanage fcontext 命令由 policycoreutils-python 软件包提供，因此您必须在 Software Collection metapackage 的 policycoreutils-python 中包含 Requires 非常重要。

注意

在 Red Hat Enterprise Linux 7 中，启动服务的 SELinux 方面发生了很大变化。最重要的是，在 systemd 服务文件中使用 scl enable ... wrapper 会导致服务作为 unconfined _service_t 上下文以不受限制的进程运行。由于此上下文没有按照设计过渡规则，因此如果该服务应该使用 SELinux 策略限制，服务将无法过渡到 SELinux 策略表示的目标 SELinux 上下文，这意味着，如果服务启动，则无法在 Red Hat Enterprise Linux 7 上使用 scl enable ...。

3.16. RED HAT ENTERPRISE LINUX 6 和 7 之间的区别

Red Hat Enterprise Linux 7 中的 RPM Package Manager 提供了很多功能更改，这些更改在 Red Hat Enterprise Linux 6 附带的 RPM Package Manager 旧版本中不提供。

本节详细介绍了为两个系统构建 Software Collection 软件包时可能会影响您的更改。

库支持的不同在 第 3.5.3 节 “Red Hat Enterprise Linux 7 中的 Software Collection Library 支持” 中详细介绍。SELinux 支持的不同信息包括在 第 3.15.1 节 “Red Hat Enterprise Linux 7 中的 SELinux 支持”。

3.16.1. %license Macro

%license 宏允许您指定软件包要安装的许可证文件。宏只在 Red Hat Enterprise Linux 7 中的 RPM

Package Manager 支持。在 Red Hat Enterprise Linux 6 和 7 上构建 Software Collection 软件包时，声明 Red Hat Enterprise Linux 6 的 %license 宏，如下所示：

```
%{!?_licensedir:%global license %%doc}
```

3.16.2. 缺少运行时子软件包依赖项

在 Red Hat Enterprise Linux 7 中，scl 工具会在 Software Collection runtime 子软件包中自动生成所需的 Requires。这在 Red Hat Enterprise Linux 6 中无法正常工作。为该系统构建 Software Collection 时，您需要在每个 Software Collection 软件包中的 runtime 子软件包上明确指定依赖项：

```
Requires: %{?scl_prefix}runtime
```

3.16.3. scl-package （）provides

按照设计，构建 Software Collection 软件包会生成多个 Provide: scl-package() 标签。它们的目的是在内部识别属于特定 Software Collection 的构建软件包。下表中详述了标签。

表 3.2. 在 Red Hat Enterprise Linux 7 中提供

Software Collection 软件包	provides
${software_collection_1}	scl-package(software_collection_1)
${software_collection_1}-build	scl-package(software_collection_1)
${software_collection_1}-runtime	scl-package(software_collection_1)

Red Hat Enterprise Linux 6 附带 RPM Package Manager 的旧版本，因此例外，在 Red Hat Enterprise Linux 6 中构建相同的软件包只生成一个 Provide: scl-package() 标签，如下表所述。这是一个预期的行为，差异由 scl 工具在内部处理。

表 3.3. 在 Red Hat Enterprise Linux 6 中提供

Software Collection 软件包	provides
${software_collection_1}	scl-package(software_collection_1)

不要使用这些内部生成的依赖项来列出属于特定 Software Collection 的软件包。有关如何正确列出 Software Collection 软件包的详情，请参考 第 1.5 节 “列出已安装的 Software Collections”。

第 4 章 扩展 RED HAT SOFTWARE COLLECTIONS

本章论述了扩展作为 Red Hat Software Collections 产品一部分的一些 Software Collections。

4.1. 提供 SCLDEVEL 子软件包

scldevel 子软件包的目的是通过提供多个通用宏文件来简化创建依赖 Software Collections 的过程。软件包器然后在扩展现有 Software Collections 时使用这些宏文件。scldevel 作为软件集合的 metapackage 的子软件包提供。

4.1.1. 创建 scldevel 子软件包

下面的部分描述了为 Ruby Software Collections、ruby193 和 ruby200 的两个示例创建一个 scldevel 子软件包。

过程 4.1. 提供您自己的 scldevel 子软件包

1. 在 Software Collection 的 metapackage 中，通过定义其名称、概述和描述来添加 scldevel 子软件包：

```
%package scldevel
Summary: Package shipping development files for %scl
Provides: scldevel(%{scl_name_base})

%description scldevel
Package shipping development files, especially useful for development of
packages depending on %scl Software Collection.
```

建议您在依赖 Software Collections 的软件包构建过程中使用虚拟 Provides: scldevel(%{scl_name_base})。这将确保 %{scl_name_base} Software Collection 及其宏的版本可用，如以下步骤中指定的。

2. 在 Software Collection 的 metapackage 的 %install 部分，创建 宏.%{scl_name_base}-scldevel 文件，该文件是 scldevel 子软件包的一部分，其中包含：

```
cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel <<
EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF
```

请注意，在所有共享相同 %{scl_name_base} 名称的 Software Collections 之间，提供的 macros.%{scl_name_base}-scldevel 文件必须冲突。这是禁止安装多个 %{scl_name_base} Software Collections 版本。例如，当安装了 ruby200-scldevel 子软件包时，无法安装 ruby193-scldevel 子软件包。

4.1.2. 在依赖软件集合中使用 scldevel 子软件包

要在依赖于 ruby200 Software Collection 的软件集合中使用 scldevel 子软件包，请更新依赖 Software Collection 的 metapackage，如下所述。

过程 4.2. 在依赖软件集合中使用您自己的 scldevel 子软件包

1. 考虑在 metapackage 的 spec 文件的开头添加以下内容：

```
%{!?scl_ruby:%global scl_ruby ruby200}
%{!?scl_prefix_ruby:%global scl_prefix_ruby %{scl_ruby}-}
```

这两个行是可选的。它们仅被称为一个可视化提示，依赖 Software Collection 设计为依赖于 ruby200 Software Collection。如果构建 root 中没有其他 scldevel 子软件包，则 ruby200-scldevel 子软件包将用作构建要求。

您可以使用以下行替换这些行：

```
%{?scl_prefix_ruby}
```

2. 在 metapackage 中添加以下构建要求：

```
BuildRequires: %{scl_prefix_ruby}scldevel
```

通过指定此构建要求，您可以确保 scldevel 子软件包位于构建根目录中，并且默认值不在使用中。省略此软件包可能会导致后续软件包构建时出现问题。

3. 确保 metapackage 的 spec 文件的 %package runtime 部分包含以下行：

```
%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_ruby}runtime
```

4.

考虑在 metapackage 的 spec 文件的 %package build 部分包括以下行：

```
%package build
Summary: Package shipping basic build configuration
Requires: %{scl_prefix_ruby}scldevel
```

指定 Requires: %{scl_prefix_ruby}scldevel 可确保宏在 Software Collection 的所有软件包中可用。

请注意，添加这个 Requires 仅在特定用例中有意义，比如依赖 Software Collection 中的软件包使用 scldevel 子软件包提供的宏。

4.2. 扩展 PYTHON27 和 RH-PYTHON35 SOFTWARE COLLECTIONS

本节论述了通过创建依赖 Software Collection 来扩展 python27 和 rh-python35 Software Collections。

在 Red Hat Software Collections 3.8 中，scl 工具被扩展来支持宏 %scl_package_override （），这有助于打包您自己的依赖 Software Collection。

4.2.1. vt191 Software Collection

以下是构建依赖 Software Collection 的未注释示例。Software Collection 的名称为 vt191，包含 版本tools Python 软件包 1.9.1。

请注意 vt191 Software Collection metapackage 中的以下内容：

- vt191 Software Collection metapackage 设置了以下构建依赖项：

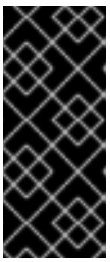
```
BuildRequires: %{scl_prefix_python}scldevel
```

例如，这扩展至 python27-scldevel。

python27-scldevel 子软件包提供了两个重要的宏 %scl_python 和 %scl_prefix_python。请注意，这些宏在 metapackage spec 文件的顶部定义。虽然定义不是必需的，但它们提供了一

个可视化提示，但 vt191 软件集合已设计为基于 python27 Software Collection 构建。它们也充当回退值。

- 要正确设置 site-packages 目录，请使用 %python27python_sitelib 宏的值，并将 python27 替换为 vt191。请注意，如果您使用其他提供程序构建 Software Collection （例如 /opt/myorganization/ 而不是 /opt/rh/），则需要更改它们。



重要

由于 /opt/rh/ 供应商用于安装红帽提供的 Software Collections，因此强烈建议使用不同的供应商来避免可能的冲突。请参阅 第 2.3 节 “Software Collection Root 目录” 了解更多信息。

- vt191-build 子软件包设置了以下依赖项：

```
Requires: %{scl_prefix_python}scldevel
```

例如，这扩展至 python27-scldevel。这种依赖项的目的是确保在为 vt191 软件集合构建软件包时始终存在宏。

- vt191 Software Collection 的 enable scriptlet 使用以下行：

```
. scl_source enable %{scl_python}
```

请注意行开头的点。当 vt191 Software Collection 启动时，此行使 Python Software Collection 会隐式启动，以便用户只能键入 scl enable vt191 命令而不是 scl enable python27 vt191 命令在 Software Collection 环境中运行命令。

- 宏文件 macros.vt191-config 调用 %scl_package_override 功能来正确覆盖 %__os_install_post、、Python 依赖项生成器，以及其它软件包规格文件中使用的特定于 Python 的宏。

```
# define name of the scl
%global scl vt191
%scl_package %scl

# Defaults for the values for the python27/rh-python35 Software Collection. These
# will be used when python27-scldevel (or rh-python35-scldevel) is not in the
# build root
```

```
%{!?scl_python:%global scl_python python27}
%{!?scl_no_vendor:%global scl_no_vendor python27}
%{!?scl_prefix_python:%global scl_prefix_python %{scl_python}-}

# Only for this build, you need to override default __os_install_post,
# because the default one would find /opt/.../lib/python2.7/ and try
# to bytecompile with the system /usr/bin/python2.7
%global __os_install_post %{%{scl_no_vendor}_os_install_post}
# Similarly, override __python_requires for automatic dependency generator
%global __python_requires %{%{scl_no_vendor}_python_requires}

# The directory for site packages for this Software Collection
%global vt191_sitelib %(echo %{python27python_sitelib} | sed 's|%{scl_python}|%{scl}|')

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
BuildRequires: scl-utils-build
# Always make sure that there is the python27-sclbuild (or rh-python35-sclbuild)
# package in the build root
BuildRequires: %{scl_prefix_python}scldevel
# Require python27-python-devel, you will need macros from that package
BuildRequires: %{scl_prefix_python}python-devel
Requires: %{scl_prefix}python-versiontools

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_python}runtime

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build
# Require python27-scldevel (or rh-python35-scldevel) so that there is always access
# to the %%scl_python and %%scl_prefix_python macros in builds for this Software
# Collection
Requires: %{scl_prefix_python}scldevel

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%prep
%setup -c -T

%install
%scl_install

# Create the enable scriptlet that:
```

```
# - Adds an additional load path for the Python interpreter.
# - Runs scl_source so that you can run:
#     scl enable vt191 "bash"
#   instead of:
#     scl enable python27 vt191 "bash"

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
. scl_source enable %{scl_python}
export PYTHONPATH="%{vt191_sitelib}\${PYTHONPATH:+:\${PYTHONPATH}}"
EOF

mkdir -p %{buildroot}%{vt191_sitelib}

# - Enable Software Collection-specific bytecompilation macros from
#   the python27-python-devel package.
# - Also override the %%python_sitelib macro to point to the vt191 Software
#   Collection.
# - If you have architecture-dependent packages, you will also need to override
#   the %%python_sitearch macro.

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl}-config << EOF
%%scl_package_override() %%{expand:%{?python27_os_install_post:%%global
__os_install_post %%python27_os_install_post}
%%global __python_requires %%python27_python_requires
%%global __python_provides %%python27_python_provides
%%global __python %python27__python
%%global python_sitelib %vt191_sitelib
%%global python2_sitelib %vt191_sitelib
}
EOF

%files

%files runtime -f filelist
%scl_files
%vt191_sitelib

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Wed Jan 22 2014 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

4.2.2. python-versiontools 软件包

以下是 python-versiontools 软件包 spec 文件的注释示例。在 spec 文件中记录以下内容：

- BuildRequires 标签前缀为 %{?scl_prefix_python} 而不是 %{scl_prefix}。

- %install 部分的过期指定 --install-purelib。

```
%{?scl:%scl_package python-versiontools}
%{!?scl:%global pkg_name %{name}}

%global pypi_name versiontools

Name:          %{?scl_prefix}python-versiontools
Version:       1.9.1
Release:       1%{?dist}
Summary:       Smart replacement for plain tuple used in __version__

License:       LGPLv3
URL:           https://launchpad.net/versiontools
Source0:       http://pypi.python.org/packages/source/v/versiontools/versiontools-1.9.1.tar.gz

BuildArch:     noarch
BuildRequires:  %{?scl_prefix_python}python-devel
BuildRequires:  %{?scl_prefix_python}python-setuptools
%{?scl:BuildRequires: %{scl}-build %{scl}-runtime}
%{?scl:Requires: %{scl}-runtime}

%description
Smart replacement for plain tuple used in __version__

%prep
%setup -q -n %{pypi_name}-%{version}

%build
%{?scl:scl enable %{scl} "}
%{__python} setup.py build
%{?scl:"}

%install
# Explicitly specify --install-purelib %{python_sitelib}, which is now overriden
# to point to vt191, otherwise Python will try to install into the python27
# Software Collection site-packages directory
%{?scl:scl enable %{scl} "}
%{__python} setup.py install -O1 --skip-build --root %{buildroot} --install-purelib %
{python_sitelib}
%{?scl:"}

%files
%{python_sitelib}/%{pypi_name}*

%changelog
* Wed Jan 22 2014 John Doe <jdoe@example.com> - 1.9.1-1
- Built for vt191 SCL.
```

4.2.3. 构建 vt191 Software Collection

构建 vt191 Software Collection：

1. *安装属于 python27 Software Collection 的 python27-scldevel 和 python27-python-devel 子软件包。*
2. *构建 vt191.spec 并安装 vt191-runtime 和 vt191-build 软件包。*
3. *安装 python27-python-setuptools 软件包，这是 版本tools 的构建要求。*
4. *构建 python-versiontools.spec。*

4.2.4. 测试 vt191 Software Collection

测试 vt191 Software Collection：

1. *安装 vt191-python-versiontools 软件包。*
2. *运行以下命令：*

```
$ scl enable vt191 "python -c 'import versiontools; print(versiontools.__file__)'"
```

3. *验证输出是否包含以下行：*

```
/opt/rh/vt191/root/usr/lib/python2.7/site-packages/versiontools/__init__.pyc
```

请注意，路径中的 provider rh 会根据 %_scl_prefix 宏的撤销而有所不同。请参阅 第 2.3 节 “Software Collection Root 目录” 了解更多信息。

4.3. 扩展 RH-RUBY23 软件集合

在 Red Hat Software Collections 3.8 中，可以通过添加依赖软件包来扩展 rh-ruby23 Software Collection。Ruby on Rails 4.2 (rh-ror42) Software Collection 基于 rh-ruby23 Software Collection 提供的 Ruby 2.3 构建，是这样的扩展的一个示例。

本节提供有关 rh-ror42 metapackage 和 rh-ror42-rubygem-bcrypt 软件包的详细信息，它们是 rh-ror42 软件集合的一部分。

4.3.1. rh-ror42 Software Collection

本节包含 rh-ror42 软件集合的 Ruby on Rails 4.2 metapackage 的注释示例。rh-ror42 Software Collection 依赖于 rh-ruby23 Software Collection。

注意 rh-ror42 Software Collection metapackage 示例中的以下内容：

- rh-ror42 Software Collection spec 文件设置了以下构建依赖项：

```
BuildRequires: %{scl_prefix_ruby}scldevel
BuildRequires: %{scl_prefix_ruby}rubygems-devel
```

例如，这扩展至 rh-ruby23-scldevel 和 rh-ruby23-rubygems-devel。

rh-ruby23-scldevel 子软件包包含两个重要的宏 %scl_ruby 和 %scl_prefix_ruby。rh-ruby23-scldevel 子软件包应该在构建 root 中提供。如果有多个 Ruby Software Collections 可用，rh-ruby23-scldevel 决定应使用哪个可用 Software Collections。

请注意，%scl_ruby 和 %scl_prefix_ruby 宏也定义在 spec 文件的顶部。虽然定义不是必需的，但它们提供了一个可视化提示，但 rh-ror42 Software Collection 已设计为基于 rh-ruby23 Software Collection 构建。它们也充当回退值。

- rh-ror42-runtime 子软件包必须依赖于它依赖的软件集合的 runtime 子软件包。这个依赖项指定如下：

```
%package runtime
Requires: %{scl_prefix_ruby}runtime
```

当软件包针对 rh-ruby23 软件集合构建时，这扩展至 rh-ruby23-runtime。

- rh-ror42-build 子软件包必须依赖于它依赖的软件集合的 scldevel 子软件包。这是为了确保这个 Software Collection 的所有其他软件包都定义相同的宏，因此针对同一 Ruby 版本构建

它。

```
%package build
Requires: %{scl_prefix_ruby}scldevel
```

如果是 rh-ruby23 Software Collection，这扩展至 rh-ruby23-scldevel。

- rh-ror42 Software Collection 的 enable scriptlet 包含以下行：

```
. scl_source enable %{scl_ruby}
```

请注意行开头的点。当 rh-ror42 Software Collection 启动时，此行使 Ruby Software Collection 会隐式启动，以便用户只能键入 scl enable rh-ror42 命令而不是 scl enable rh-ruby23 rh-ror42 命令，以便在 Software Collection 环境中运行命令。

- 提供了 rh-ror42-scldevel 子软件包，以便在需要它来构建软件集合来扩展 rh-ror42 Software Collection 时可用。软件包提供 %{scl_ror} 和 %{scl_prefix_ror} 宏，可用于扩展 rh-ror42 软件集合。

- 由于 rh-ror42 Software Collection 的 gem 安装在单独的 root 目录结构中，您需要确保设置了 rubygems 目录的正确所有权。这可以通过使用代码片段生成文件列表 rubygems_filesystem.list 来完成。

建议您将 runtime 软件包设置为拥有所有目录（如果位于 root 文件系统中）归另一个软件包所有。例如，当 rh-ror42 Software Collection 是 Rubygem 目录结构时，此类目录的一个示例。

```
%global scl_name_prefix rh-
%global scl_name_base ror
%global scl_name_version 41

%global scl %{scl_name_prefix}%{scl_name_base}%{scl_name_version}

# Fallback to rh-ruby23. rh-ruby23-scldevel is unlikely to be available in
# the build root.
%{!?scl_ruby:%global scl_ruby rh-ruby23}
%{!?scl_prefix_ruby:%global scl_prefix_ruby %{scl_ruby}-}

# Do not produce empty debuginfo package.
%global debug_package %{nil}

# Support SCL over NFS.
```

```
%global nfsmountable 1

%{!?install_scl: %global install_scl 1}

%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 2.0
Release: 5%{?dist}
License: GPLv2+

%if 0%{?install_scl}
Requires: %{scl_prefix}rubygem-therubyracer
Requires: %{scl_prefix}rubygem-sqlite3
Requires: %{scl_prefix}rubygem-rails
Requires: %{scl_prefix}rubygem-sass-rails
Requires: %{scl_prefix}rubygem-coffee-rails
Requires: %{scl_prefix}rubygem-jquery-rails
Requires: %{scl_prefix}rubygem-sdoc
Requires: %{scl_prefix}rubygem-turbolinks
Requires: %{scl_prefix}rubygem-bcrypt
Requires: %{scl_prefix}rubygem-uglifier
Requires: %{scl_prefix}rubygem-jbuilder
Requires: %{scl_prefix}rubygem-spring
%endif
BuildRequires: help2man
BuildRequires: scl-utils-build
BuildRequires: %{scl_prefix_ruby}scldevel
BuildRequires: %{scl_prefix_ruby}rubygems-devel

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
# The enable scriptlet depends on the ruby executable.
Requires: %{scl_prefix_ruby}ruby

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
Requires: scl-utils-build
Requires: %{scl_runtime}
Requires: %{scl_prefix_ruby}scldevel

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%package scldevel
Summary: Package shipping development files for %scl
Provides: scldevel(%{scl_name_base})
```

```
%description scldevel
Package shipping development files, especially usefull for development of
packages depending on %scl Software Collection.

%prep
%setup -c -T

%install
%scl_install

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export LD_LIBRARY_PATH="%{_libdir}\${LD_LIBRARY_PATH:+:\${LD_LIBRARY_PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
export PKG_CONFIG_PATH="%
{_libdir}/pkgconfig\${PKG_CONFIG_PATH:+:\${PKG_CONFIG_PATH}}"
export GEM_PATH="\${GEM_PATH:=%{gem_dir}:\`scl enable %{scl_ruby} -- ruby -e "print
Gem.path.join(':')"\`}"

. scl_source enable %{scl_ruby}
EOF

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel << EOF
%%scl_%{scl_name_base} %{scl}
%%scl_prefix_%{scl_name_base} %{scl_prefix}
EOF


scl enable %{scl_ruby} - << \EOF
set -e

# Fake rh-ror42 Software Collection environment.
GEM_PATH=%{gem_dir}:`ruby -e "print Gem.path.join(':')"` \
X_SCLS=%{scl} \
ruby -rfileutils > rubygems_filesystem.list << \EOR
  # Create the RubyGems file system.
  Gem.ensure_gem_subdirectories '%{buildroot}%{gem_dir}'
  FileUtils.mkdir_p File.join '%{buildroot}', Gem.default_ext_dir_for('%{gem_dir}')

  # Output the relevant directories.
  Gem.default_dirs['%{scl}_system'.to_sym].each { |k, p| puts p }
EOR
EOF

%files

%files runtime -f rubygems_filesystem.list
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%files scldevel
%{_root_sysconfdir}/rpm/macros.%{scl_name_base}-scldevel
```

```
%changelog
* Thu Jan 16 2015 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

4.3.2. rh-ror42-rubygem-bcrypt 软件包

以下是 rh-ror42-rubygem-bcrypt 软件包 spec 文件的注释示例。这个软件包提供 bcrypt Ruby gem。有关 bcrypt 的更多信息，请参阅以下网站：

- http://rubygems.org/gems/bcrypt-ruby

请注意，rh-ror42-rubygem-bcrypt 软件包 spec 文件和普通 Software Collection 软件包 spec 文件之间的唯一显著区别如下：

- BuildRequires 标签前缀为 %{?scl_prefix_ruby} 而不是 %{scl_prefix}。

```
%{?scl:%scl_package rubygem-%{gem_name}}
%{!?scl:%global pkg_name %{name}}

%global gem_name bcrypt

Summary: Wrapper around bcrypt() password hashing algorithm
Name: %{?scl_prefix}rubygem-%{gem_name}
Version: 3.1.9
Release: 2%{?dist}
Group: Development/Languages
# ext/* - Public Domain
# spec/TestBCrypt.java - ISC
License: MIT and Public Domain and ISC
URL: https://github.com/codahale/bcrypt-ruby
Source0: http://rubygems.org/downloads/%{gem_name}-%{version}.gem
Requires: %{?scl_prefix_ruby}ruby(release)
Requires: %{?scl_prefix_ruby}ruby(rubygems)
BuildRequires: %{?scl_prefix_ruby}rubygems-devel
BuildRequires: %{?scl_prefix_ruby}ruby-devel
BuildRequires: %{?scl_prefix}rubygem(rspec)
Provides: %{?scl_prefix}rubygem(bcrypt) = %{version}

%description
bcrypt() is a sophisticated and secure hash algorithm designed by The
OpenBSD project for hashing passwords. bcrypt provides a simple,
humane wrapper for safely handling passwords.

%package doc
Summary: Documentation for %{pkg_name}
Group: Documentation
Requires: %{?scl_prefix}%{pkg_name} = %{version}-%{release}
```

```
%description doc
Documentation for %{pkg_name}.

%prep
%setup -n %{pkg_name}-%{version} -q -c -T
%{?scl:scl enable %{scl} - << \EOF}
%gem_install -n %{SOURCE0}
%{?scl:EOF}

%build

%install
mkdir -p %{buildroot}%{gem_dir}
cp -pa .%{gem_dir}/* \
        %{buildroot}%{gem_dir}/

mkdir -p %{buildroot}%{gem_extdir_mri}
cp -pa .%{gem_extdir_mri}/* %{buildroot}%{gem_extdir_mri}/

# Prevent a symlink with an invalid target in -debuginfo (BZ#878863).
rm -rf %{buildroot}%{gem_instdir}/ext/

%check
%{?scl:scl enable %{scl} - << \EOF}
pushd .%{gem_instdir}
# 2 failutes due to old RSpec
# https://github.com/rspec/rspec-expectations/pull/284
rspec -I$(dirs +1)%{gem_extdir_mri} spec |grep '34 examples, 2 failures' || exit 1
popd
%{?scl:EOF}

%files
%dir %{gem_instdir}
%exclude %{gem_instdir}/.*
%{gem_libdir}
%{gem_extdir_mri}
%exclude %{gem_cache}
%{gem_spec}
%doc %{gem_instdir}/COPYING

%files doc
%doc %{gem_docdir}
%doc %{gem_instdir}/README.md
%doc %{gem_instdir}/CHANGELOG
%{gem_instdir}/Rakefile
%{gem_instdir}/Gemfile*
%{gem_instdir}/%{gem_name}.gemspec
%{gem_instdir}/spec


%changelog
* Fri Mar 21 2015 John Doe <jdoe@example.com> - 3.1.2-4
- Initial package.
```

4.3.3. 构建 rh-ror42 Software Collection

构建 rh-ror42 Software Collection：

1. 安装 rh-ruby23-scldevel 子软件包，它是 rh-ruby23 软件集合的一部分。
2. 构建 rh-ror42.spec 并安装 ror42-runtime 和 ror42-build 软件包。
3. 构建 rubygem-bcrypt.spec。

4.3.4. 测试 rh-ror42 Software Collection

测试 rh-ror42 Software Collection：

1. 安装 rh-ror42-rubygem-bcrypt 软件包。
2. 运行以下命令：

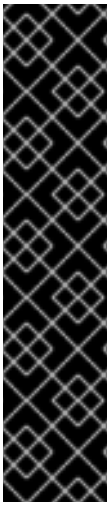
```
$ scl enable rh-ror42 -- ruby -r bcrypt -e "puts BCrypt::Password.create('my password')"
```

3. 验证输出是否包含以下行：

```
$2a$10$s./ReniLY.wXPHVBQ9npoeyZf5KzywfpvI5lhjG6Ams3u0hKqwVbW
```

4.4. 扩展 RH-PERL524 SOFTWARE COLLECTION

本节论述了通过构建您自己的依赖软件集合来扩展 rh-perl524 Software Collection。



重要

本节中描述的示例只适用于使用以下软件包扩展 rh-perl524 Software Collection 时按预期工作：

- 不要提供任何 Perl 模块，以及
- 仅依赖于 rh-perl524 Software Collection 提供的 Perl 模块。

4.4.1. h2m144 Software Collection

本节包含依赖 Software Collection 的 metapackage 的注释示例。依赖软件包名为 h2m144，包含 help2man Perl 软件包版本 1.44.1。h2m144 Software Collection 依赖于 rh-perl524 Software Collection。

请注意，h2m144 Software Collection metapackage 中的以下内容：

- h2m144 Software Collection metapackage 设置了以下构建依赖项：

```
BuildRequires: %{scl_prefix_perl}scldevel
```

这会扩展到 rh-perl524-scldevel。

rh-perl524-scldevel 子软件包包含两个重要的宏 %scl_perl 和 %scl_prefix_perl，并提供 Perl 依赖项生成器。请注意，宏在 metapackage spec 文件的顶部定义。虽然定义不是必需的，但它们提供了一个可视化提示，但 h2m144 Software Collection 已设计为基于 rh-perl524 Software Collection 构建。它们也充当回退值。

- h2m144-build 子软件包设置了以下依赖项：

```
Requires: %{scl_prefix_perl}scldevel
```

这会扩展到 rh-perl524-scldevel。此依赖项的目的是确保在为 h2m144 软件集合构建软件包时始终存在宏和依赖项生成器。

- *h2m144 Software Collection 的 enable scriptlet 包含以下行：*

```
. scl_source enable %{scl_perl}
```

请注意行开头的点。当 h2m144 Software Collection 启动时，此行使 Perl Software Collection 会隐式启动，以便用户只能键入 scl enable h2m144 命令而不是 scl enable rh-perl524 h2m144 命令在 Software Collection 环境中运行命令。

- *宏文件 macros.h2m144-config 调用 Perl 依赖项生成器，以及其它软件包规格文件中使用的特定于 Perl 的宏。*

```
%global scl h2m144
%scl_package %scl

# Default values for the rh-perl524 Software Collection. These
# will be used when rh-perl524-scldevel is not in the build root.
%{!?scl_perl:%global scl_perl rh-perl524}
%{!?scl_prefix_perl:%global scl_prefix_perl %{scl_perl}-}

# Only for this build, override __perl_requires for the automatic dependency
# generator.
%global __perl_requires /usr/lib/rpm/perl.req.stack

Summary: Package that installs %scl
Name:    %scl_name
Version: 1
Release: 1%{?dist}
License: GPLv2+
BuildRequires: scl-utils-build
# Always make sure that there is the rh-perl524-scldevel
# package in the build root.
BuildRequires: %{scl_prefix_perl}scldevel
# Require rh-perl524-perl-macros; you will need macros from that package.
BuildRequires: %{scl_prefix_perl}perl-macros
Requires: %{scl_prefix}help2man

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils
Requires: %{scl_prefix_perl}runtime

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration
```

```
Requires: scl-utils-build
# Require rh-perl524-scldevel so that there is always access to the %%scl_perl
# and %%scl_prefix_perl macros in builds for this Software Collection.
Requires: %{scl_prefix_perl}scldevel

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%prep
%setup -c -T

%build

%install
%scl_install

# Create the enable scriptlet that:
# - Adds an additional load path for the Perl interpreter.
# - Runs scl_source so that you can run:
#     scl enable h2m144 'bash'
#   instead of:
#     scl enable rh-perl524 h2m144 'bash'

cat >> %{buildroot}%{_scl_scripts}/enable << EOF
. scl_source enable %{scl_perl}
export PATH="%{_bindir}:%{_sbindir}\${PATH:+:\${PATH}}"
export MANPATH="%{_mandir}:\${MANPATH:-}"
EOF

cat >> %{buildroot}%{_root_sysconfdir}/rpm/macros.%{scl}-config << EOF
%%scl_package_override() %%{expand:%%global __perl_requires /usr/lib/rpm/perl.req.stack
%%global __perl_provides /usr/lib/rpm/perl.prov.stack
%%global __perl %{_scl_prefix}/%{scl_perl}/root/usr/bin/perl
}
EOF

%files

%files runtime -f filelist
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Tue Apr 22 2014 John Doe <jdoe@example.com> - 1-1
- Initial package.
```

4.4.2. help2man 软件包

以下是 help2man 软件包 spec 文件的注释示例。在 spec 文件中记录以下内容：

- *BuildRequires 标签前缀为 %{?scl_prefix_perl} 而不是 %{scl_prefix}。*

```
%{?scl:%scl_package help2man}
%{!?scl:%global pkg_name %{name}}

# Supported build option:
#
# --with nls ... build this package with --enable-nls
%bcond_with nls

Name:          %{?scl_prefix}help2man
Summary:       Create simple man pages from --help output
Version:       1.44.1
Release:       1%{?dist}
Group:         Development/Tools
License:       GPLv3+
URL:           http://www.gnu.org/software/help2man
Source:        ftp://ftp.gnu.org/gnu/help2man/help2man-%{version}.tar.xz
%{!?with_nls:BuildArch: noarch}

BuildRequires:  %{?scl_prefix_perl}perl(Getopt::Long)
BuildRequires:  %{?scl_prefix_perl}perl(POSIX)
BuildRequires:  %{?scl_prefix_perl}perl(Text::ParseWords)
BuildRequires:  %{?scl_prefix_perl}perl(Text::Tabs)
BuildRequires:  %{?scl_prefix_perl}perl(strict)
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(Locale::gettext) /usr/bin/msgfmt}
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(Encode)}
%{?with_nls:BuildRequires: %{?scl_prefix_perl}perl(I18N::Langinfo)}
Requires:   %{?scl_prefix_perl}perl(:MODULE_COMPAT_%(%{?scl:scl enable %{scl_perl}
'}eval "`perl -V:version`"; echo $version%{?scl:'}))

Requires(post): /sbin/install-info
Requires(preun): /sbin/install-info

%description
help2man is a script to create simple man pages from the --help and
--version output of programs.

Since most GNU documentation is now in info format, this provides a
way to generate a placeholder man page pointing to that resource while
still providing some useful information.

%prep
%setup -q -n help2man-%{version}

%build
%configure --%{!?with_nls:disable}%{?with_nls:enable}-nls --libdir=%{_libdir}/help2man
%{?scl:scl enable %{scl} "}
make %{?_smp_mflags}
%{?scl:"}

%install
%{?scl:scl enable %{scl} "}
make install_l10n DESTDIR=$RPM_BUILD_ROOT
```

```
%{?scl:"}
%{?scl:scl enable %{scl} "}
make install DESTDIR=$RPM_BUILD_ROOT
%{?scl:"}
%find_lang %pkg_name --with-man

%post
/sbin/install-info %{_infodir}/help2man.info %{_infodir}/dir 2>/dev/null || :

%preun
if [ $1 -eq 0 ]; then
  /sbin/install-info --delete %{_infodir}/help2man.info \
    %{_infodir}/dir 2>/dev/null || :
fi

%files -f %pkg_name.lang
%doc README NEWS THANKS COPYING
%{_bindir}/help2man
%{_infodir}/*
%{_mandir}/man1/*

%if %{with nls}
%{_libdir}/help2man
%endif

%changelog
* Tue Apr 22 2014 John Doe <jdoe@example.com> - 1.44.1-1
- Built for h2m144 SCL.
```

4.4.3. 构建 h2m144 Software Collection

构建 h2m144 Software Collection：

1. 安装作为 perl524 Software Collection 一部分的 rh-perl524-scldevel 和 rh-perl524-perl-macros 软件包。

2. 构建 h2m144.spec 并安装 h2m144-runtime 和 h2m144-build 软件包。

3. 安装 rh-perl524-perl,rh-perl524-perl-Text-ParseWords 和 rh-perl524-perl-Getopt-Long 软件包，这是 help2man 的构建要求。

4. 构建 help2man.spec。

4.4.4. 测试 h2m144 Software Collection

测试 h2m144 Software Collection：

1. ***安装 h2m144-help2man 软件包。***

2. ***运行以下命令：***

```
$ scl enable h2m144 'help2man bash'
```

3. ***验证输出是否类似于以下行：***

```
.\" DO NOT MODIFY THIS FILE!  It was generated by help2man 1.44.1.
.TH BASH, "1" "April 2014" "bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)" "User
Commands"
.SH NAME
bash, \- manual page for bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)
.SH SYNOPSIS
.B bash
[\fIGNU long option\fR] [\fIoption\fR] ...
.SH DESCRIPTION
GNU bash, version 4.1.2(1)\-release\-(x86_64\-redhat\-linux\-gnu)
.IP
bash [GNU long option] [option] script\-file ...
.SS "GNU long options:"
.HP
\fB\-\-debug\fR
```

第 5 章 SOFTWARE COLLECTIONS 故障排除

本章可帮助您对构建 Software Collections 时可能会遇到的一些常见问题进行故障排除。

5.1. ERROR: LINE XX: UNKNOWN TAG: %SCL_PACKAGE SOFTWARE_COLLECTION_NAME

构建 Software Collection 软件包时可能会遇到此错误消息。它通常由缺少软件包 scl-utils-build 造成的。要安装 scl-utils-build 软件包，请运行以下命令：

```
# yum install scl-utils-build
```

如需更多信息，请参阅 第 1.3 节 “启用对 Software Collections 的支持”。

5.2. SCL 命令不存在

这个错误消息通常由缺少软件包 scl-utils 造成的。要安装 scl-utils 软件包，请运行以下命令：

```
# yum install scl-utils
```

如需更多信息，请参阅 第 1.3 节 “启用对 Software Collections 的支持”。

5.3. 无法打开 /ETC/SCL/PREFIXES/SOFTWARE_COLLECTION_NAME

通过将不正确的参数与您要调用的 scl 命令搭配使用，可以导致此错误消息。检查 scl 命令是否正确，并且您没有误输入任何参数。

同样的错误消息也可以通过缺少的 Software Collection 造成的。确保系统上已正确安装了 software_collection_name Software Collection。如需更多信息，请参阅 第 1.5 节 “列出已安装的 Software Collections”。

5.4. SCL_SOURCE: 命令未找到

这个错误消息通常由安装有旧版本 scl-utils 软件包造成的。要更新 scl-utils 软件包，请运行以下命令：

```
# yum update scl-utils
```

-

附录 A. 获取更多信息

有关 Software Collection 打包、Red Hat Developers、Red Hat Software Collections 和 Red Hat Developer Toolset 产品以及 Red Hat Enterprise Linux 的更多信息，请参阅以下列出的资源。

A.1. RED HAT DEVELOPERS

- Red Hat Developers 上的 Red Hat Software Collections 概述 - 红帽开发人员门户提供了一些教程，供您开始使用不同的开发技术来开始开发代码。这包括 Node.js、Perl、PHP、Python 和 Ruby Software Collection。

- Red Hat Developer Blog - Red Hat Developer Blog 包含最新的信息、最佳实践、建议、产品和程序公告，以及针对基于红帽技术设计和开发应用程序的示例代码和其他资源的指针。

A.2. 安装的文档

- SCL (1)- scl 工具的手册页，用于在 Software Collection 的环境中运行程序。

- SCL --help - scl 工具的常规使用信息，用于在 Software Collection 环境中运行程序。

- rpmbuild(8)- rpmbuild 实用程序的手册页，用于构建二进制和源代码软件包。

A.3. 访问红帽文档

位于 的 红帽产品文档 https://access.redhat.com/documentation/ 充当中央信息来源。它在 HTML、PDF 和 EPUB 格式中提供了不同类型的从发行和技术说明到安装、用户和参考指南。

以下是一个与本书直接或间接相关的简要的文档列表：

- Red Hat Software Collections 3.8 发行注记 - Red Hat Software Collections 3.8 发行注记介绍了主要功能，并包含有关 Red Hat Software Collections 的其他信息，它提供一组动态编程语言、数据库服务器和各种相关软件包。

- Red Hat Developer Toolset 12.1 用户指南 - Red Hat Developer Toolset 12.1 的 用户指南包含有关 Red Hat Developer Toolset (Red Hat Developer Toolset)的信息，这是 Red Hat

Enterprise Linux 平台上的开发人员的产品。使用 Software Collections，Red Hat Developer Toolset 提供了 GCC 编译器、GDB 调试器和其他二进制工具的当前版本。

- *使用 Red Hat Software Collections 3.8 容器镜像 - 本指南提供了有关如何基于 Red Hat Software Collections 使用容器镜像的信息。可用的容器镜像包括应用程序、守护进程和数据库。镜像可以在 Red Hat Enterprise Linux 7 服务器和 Red Hat Enterprise Linux Atomic Host 上运行。*

- *Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 的开发人员指南提供了 Red Hat Developer Toolset 功能的详细信息，以及 Red Hat Software Collections 简介，以及库和运行时支持、编译和构建、调试和分析的信息。*

- *Red Hat Enterprise Linux 7 系统管理员指南 - Red Hat Enterprise Linux 7 系统管理员指南介绍了与 Red Hat Enterprise Linux 7 部署、配置和管理相关的信息。*

- *Red Hat Enterprise Linux 6 开发人员指南 - Red Hat Enterprise Linux 6 的开发人员指南提供了 Red Hat Developer Toolset 功能的详细信息，以及 Red Hat Software Collections 简介，以及库和运行时支持、编译和构建、调试和分析的信息。*

- *Red Hat Enterprise Linux 6 部署指南 - Red Hat Enterprise Linux 6 的部署指南介绍了与 Red Hat Enterprise Linux 6 部署、配置和管理相关的信息。*

附录 B. 修订历史记录

修订 4.2-0 Fri Jun 09 2023 Lenka Špačková
改进了 第 2.10.1 节 “转换的 Spec 文件示例”和 第 2.10.2 节 “转换标签和 Macro 定义”。

修订 4.1-9 Tue May 23 2023 Lenka Špačková
更新了 Red Hat Developer Toolset 12.1 发行版本的引用。

修订 4.1-8 Mon Nov 15 2021 Lenka Špačková
打包指南的 Red Hat Software Collections 3.8 发行版本。

修订 4.1-7 Mon Oct 11 2021 Lenka Špačková
打包指南的 Red Hat Software Collections 3.8 Beta 发行版。

修订 4.1-6 Thu Jun 03 2021 Lenka Špačková
打包指南的 Red Hat Software Collections 3.7 发行版本。

修订 4.1-5 Mon May 03 2021 Lenka Špačková
打包指南的 Red Hat Software Collections 3.7 Beta 发行版本。

修订 4.1-4 Tue Dec 01 2020 Lenka Špačková
打包指南的 Red Hat Software Collections 3.6 发行版本。

修订 4.1-3 Tue Oct 29 2020 Lenka Špačková
Red Hat Software Collections 3.6 Beta 版本打包指南。

修订 4.1-2 Tue May 26 2020 Lenka Špačková
打包指南的 Red Hat Software Collections 3.5 发行版本。

修订 4.1-1 Tue Apr 21 2020 Lenka Špačková
Red Hat Software Collections 3.5 Beta 版本打包指南。

修订 4.1-0 Tue Dec 10 2019 Lenka Špačková
打包指南的 Red Hat Software Collections 3.4 发行版本。

修订 4.0-9 Mon Oct 28 2019 Lenka Špačková
打包指南的 Red Hat Software Collections 3.4 Beta 发行版。

修订 4.0-8 Tue Jun 04 2019 Petr Kovář
打包指南的 Red Hat Software Collections 3.3 发行版本。

修订 4.0-7 Wed Apr 10 2019 Petr Kovář
打包指南的 Red Hat Software Collections 3.3 Beta 发行版。

修订 4.0-6 Thu Nov 01 2018 Petr Kovář
打包指南的 Red Hat Software Collections 3.2 发行版本。

修订 4.0-5 Wed Oct 17 2018 Petr Kovář
打包指南的 Red Hat Software Collections 3.2 Beta 发行版本。

修订 4.0-4 Thu Apr 12 2018 Petr Kovář
打包指南的 Red Hat Software Collections 3.1 发行版本。

修订 4.0-3 Fri Mar 16 2018 Petr Kovář

打包指南的 Red Hat Software Collections 3.1 Beta 发行版。

修订 4.0-2 **Tue Oct 17 2017** **Petr Kovář**
打包指南的 Red Hat Software Collections 3.0 发行版本。

修订 4.0-1 **Thu Aug 31 2017** **Petr Kovář**
Red Hat Software Collections 3.0 Beta 版本打包指南。

修订 3.10-0 **Mon Jun 5 2017** **Petr Kovář**
重新发布以修复 BZ11458821。

修订 3.9-0 **Thu Apr 20 2017** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.4 发行版本。

修订 3.8-0 **Wed Apr 05 2017** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.4 Beta 发行版本。

修订 3.7-0 **Wed Jan 25 2017** **Petr Kovář**
重新发布以修复 BZ39）63733。

修订 3.6-0 **Wed Nov 02 2016** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.3 发行版本。

修订 3.5-0 **Wed Oct 12 2016** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.3 Beta 发行版。

修订 3.4-0 **Mon May 23 2016** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.2 发行版本。

修订 3.3-0 **Tue Apr 26 2016** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.2 Beta 发行版。

修订 3.2-0 **Wed Nov 04 2015** **Petr Kovář**
Red Hat Software Collections 2.1 发行版本打包指南。

修订 3.1-0 **Tue Oct 06 2015** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.1 Beta 发行版本。

修订 3.0-2 **Tue May 19 2015** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.0 发行版本。

修订 3.0-1 **Wed Apr 22 2015** **Petr Kovář**
打包指南的 Red Hat Software Collections 2.0 Beta 发行版。

修订 2.2-4 **Fri Nov 21 2014** **Petr Kovář**
重新发布以修复 BZ39）50573、BZ39）1022023 和 BZ39）49650。

修订 2.2-2 **Thu Oct 30 2014** **Petr Kovář**
打包指南的 Red Hat Software Collections 1.2 发行版本。

修订 2.2-1 **Tue Oct 07 2014** **Petr Kovář**
Red Hat Software Collections 1.2 Beta 刷新版本打包指南。

修订 2.2-0 **Tue Sep 09 2014** **Petr Kovář**
Software Collections Guide 重命名为 Packaging Guide。
打包指南的 Red Hat Software Collections 1.2 Beta 发行版。

修订 2.1-29 Wed Jun 04 2014 Petr Kovář
Software Collections Guide 的 Red Hat Software Collections 1.1 发行版本。

修订 2.1-21 Thu Mar 20 2014 Petr Kovář
Software Collections 1.1 Beta 的 Red Hat Software Collections 1.1 Beta 版本。

修订 2.1-18 Tue Mar 11 2014 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 2.1 发行版本。

修订 2.1-8 Tue Feb 11 2014 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 2.1 Beta 发行版本。

修订 2.0-12 Tue Sep 10 2013 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 2.0 发行版本。

修订 2.0-8 Tue Aug 06 2013 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 2.0 Beta-2 发行版本。

修订 2.0-3 Tue May 28 2013 Petr Kovář
Software Collections 指南的 Red Hat Developer Toolset 2.0 Beta-1 发行版本。

修订 1.0-2 Tue Apr 23 2013 Petr Kovář
重新发布以修复 BZ945949000。

修订 1.0-1 Tue Jan 22 2013 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 1.1 发行版本。

修订 1.0-2 Thu Nov 08 2012 Petr Kovář
Software Collections 指南的 Red Hat Developer Toolset 1.1 Beta-2 发行版本。

修订 1.0-1 Wed Oct 10 2012 Petr Kovář
Software Collections 指南的 Red Hat Developer Toolset 1.1 Beta-1 发行版本。

修订 1.0-0 Tue Jun 26 2012 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 1.0 发行版本。

修订 0.0-2 Tue Apr 10 2012 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 1.0945-2 发行版本。

修订 0.0-1 Tue Mar 06 2012 Petr Kovář
Software Collections Guide 的 Red Hat Developer Toolset 1.0945-1 发行版本。

B.1. 致谢

本书的作者借此感谢以下人员对他们的宝贵贡献： Jindřich 4.18.0-, Marcela Maüláláová, Bohuslav Kabrda, Honza Horák, Jan Zelenř, Martin üermák, Jitka Plesnř, Langdon White, Florian Nadge, Florian Nadge, Stephen Wadeley, Douglas Silas, Tomáü HQapek, 和 V\":\"t Ondruch）等等。