Sun Cluster 数据服务开发者指南（适用于 Solaris OS）

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

文件号码 819–2070–10
2005 年 8 月，修订版 A

050817@12762

目录

前言

《Sun Cluster 数据服务开发者指南（适用于 Solaris OS）》包含有关使用资源管理 API 在基于 SPARC® 和 x86 的系统上开发 Sun™ Cluster 数据服务的信息。

注 – 在本文档中，术语“x86”指 Intel 32 位微处理器芯片系列和 AMD 制造的兼容微处理器芯片。

注 – Sun Cluster 软件可以在 SPARC 和 x86 两种平台上运行。除非在特定的章、节、说明、标有项目符号的项、图、表或示例中指出，否则本文档中的信息均适用于两种平台。

本书的读者

本文档面向具有丰富的 Sun 软硬件知识的有经验的开发者。本书中的信息均假定读者具有 Solaris 操作系统方面的知识。

本书结构

《Sun Cluster 数据服务开发者指南（适用于 Solaris OS）》包括以下章节和附录：

第 1 章概述了开发数据服务所需的概念。

第 2 章提供了关于开发数据服务的详细信息。

第 3 章提供了构成资源管理 API (RMAPI) 的访问函数和回调方法的参考。

第 4 章讨论了修改资源类型需要了解的问题。还介绍了关于群集管理员升级资源时可用的方法的信息。

第 5 章提供了 `in.named` 应用程序的 Sun Cluster 数据服务样例。

第 6 章概述了构成数据服务开发库 (DSDL) 的应用程序编程接口。

第 7 章说明了 DSDL 在设计和实现资源类型中的典型应用。

第 8 章介绍了使用 DSDL 实现的资源类型样例。

第 9 章介绍了 SunPlex™ Agent Builder。

第 10 章介绍了如何创建普通数据服务。

第 11 章介绍了 DSDL API 函数。

第 12 章提供了有关群集重新配置通知协议 (CRNP) 的信息。CRNP 使故障转移和可伸缩应用程序成为“群集可识别”的应用程序。

附录 A 介绍了标准资源类型、资源和资源组属性。

附录 B 提供了数据服务样例中每个方法的完整代码。

附录 C 列出了 `SUNW.xfnts` 资源类型中每个方法的完整代码。

附录 D 列出了对资源组管理器 (RGM) 的名称和值的合法字符的要求。

附录 E 列出了对要成为具有高可用性的候选应用程序的非群集可识别的普通应用程序的要求。

附录 F 列出了 CRNP 的文档类型定义。

附录 G 列出了第 12 章中讨论的完整的 `CrnpClient.java` 应用程序。

相关文档

有关相关 Sun Cluster 主题的信息，可从下表列出的文档中获得。Sun Cluster 文档可在 `http://docs.sun.com` 获得。

主题	文档
概述	《Sun Cluster 概述（适用于 Solaris OS）》

主题	文档
概念	《Sun Cluster 概念指南（适用于 Solaris OS）》
硬件安装和管理	《Sun Cluster 3.0-3.1 Hardware Administration Manual for Solaris OS》
	单个硬件管理指南
软件安装	《Sun Cluster 软件安装指南（适用于 Solaris OS）》
数据服务安装和管理	《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》
	单个数据服务指南
数据服务开发	《Sun Cluster 数据服务开发者指南（适用于 Solaris OS）》
系统管理	《Sun Cluster 系统管理指南（适用于 Solaris OS）》
错误消息	《Sun Cluster Error Messages Guide for Solaris OS》
命令和功能参考	《Sun Cluster Reference Manual for Solaris OS》

有关 Sun Cluster 文档的完整列表，请参见适用于您的 Sun Cluster 软件版本的发行说明，网址为：http://docs.sun.com。

获得帮助

如果您在安装或使用 Sun Cluster 软件时遇到任何问题，请与您的服务提供商联系并提供以下信息：

- 您的姓名和电子邮件地址
- 您的公司名称、地址和电话号码
- 系统的型号和序列号
- 操作系统的发行版本号（例如，Solaris 10）
- Sun Cluster 的发行版本号（例如，Sun Cluster 3.1）

使用以下命令可为服务提供商收集您系统上的信息。

命令	功能
`prtconf -v`	显示系统内存的大小并报告有关外围设备的信息
`psrinfo -v`	显示有关处理器的信息

命令	功能
`showrev -p`	报告已安装了哪些修补程序
SPARC：`prtdiag -v`	显示系统诊断信息
`/usr/cluster/bin/scinstall -pv`	显示 Sun Cluster 发行版本和软件包版本信息

还请提供 `/var/adm/messages` 文件的内容。

文档、支持和培训

Sun 职能	URL	说明
文档	`http://www.sun.com/documentation/`	下载 PDF 和 HTML 文档，以及订购印刷文档
支持和培训	`http://www.sun.com/supportraining/`	获得技术支持、下载修补程序，以及了解 Sun 课程

印刷约定

下表描述了本书中使用的印刷约定。

表 P–1 印刷约定

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

表 P–1 印刷约定　（续）

字体*	含义	示例
AaBbCc123	保留未译的新词或术语以及要强调的词。要使用实名或值替换的命令行变量。	要删除文件，请键入 `rm` *filename*。 （注：在联机状态下，有些需要强调的词以黑体显示。）
新词术语强调	新词或术语以及要强调的词。	执行**修补程序分析**。 请**勿**保存文件。
《书名》	书名	阅读《用户指南》的第 6 章。

* 浏览器的设置可能会与这些设置有所不同。

命令示例中的 shell 提示符

下表显示了
C shell、Bourne shell 和 Korn shell 的缺省系统提示符和超级用户提示符。

表 P–2 Shell 提示符

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

第 1 章

资源管理概述

本书提供了为 Oracle®、Sun Java™ System Web Server（以前的 Sun ONE Web Server）或 DNS 等软件应用程序创建资源类型的指导信息。因此本书面向的是资源类型的开发者。

本章概述了开发数据服务需要了解的概念。本章包括以下主题：

注 – 本书中使用的术语**资源类型**和**数据服务**是可以互换的。术语**代理**在本书中很少使用，但是该术语与**资源类型**和**数据服务**等效。

Sun Cluster 应用程序环境

使用 Sun Cluster 系统，可以使应用程序作为具有高可用性和高可伸缩性资源运行和管理。资源组管理器 (RGM) 提供了高可用性和高可伸缩性机制。以下元素构成此工具的编程接口：

- 为使 RGM 控制群集中的应用程序编写的一组回调方法。
- 资源管理 API (RMAPI) 是一组低级 API 命令和函数，可用于编写回调方法。这些 API 是在 `libscha.so` 库中实现的。
- 进程监视器工具 (PMF) 用于监视和重新启动群集中的进程。
- 数据服务开发库 (DSDL) 是一组库函数，它在较高的级别上封装了低级 API 和进程管理功能。DSDL 添加了一些附加功能以便于编写回调方法。这些函数是在 `libdsdev.so` 库中实现的。

下图说明了这些元素之间的相互关系。

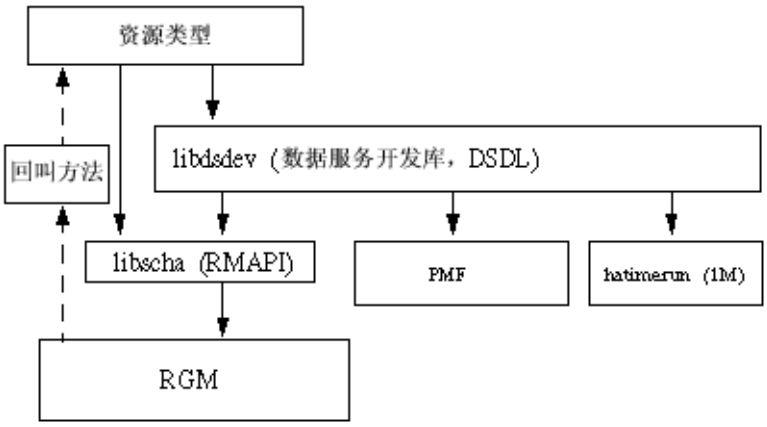


图 1–1 Sun Cluster 应用环境的编程体系结构

第 9 章中介绍的 SunPlex Agent Builder 是 Sun Cluster 软件包中的一个工具，它用于自动执行创建数据服务的进程。Agent Builder 将生成用 C 语言（通过使用 DSDL 函数编写回调方法）或用 Korn (ksh) shell 命令语言（通过使用低级 API 命令编写回调方法）编写的数据服务代码。

RGM 将作为守护进程在每个群集节点上运行，并根据预配置的策略自动启动和停止选定节点上的资源。在节点出现故障或重新引导的情况下，RGM 通过停止受影响节点上的资源再在其他节点上启动该资源，使其具有高可用性。RGM 还会自动启动和停止资源特定的监视器。这些监视器用于检测资源故障并将故障资源重定位到其他节点上，或者监视其他方面的资源性能。

RGM 既支持故障转移资源（一次只能在一个节点上联机），又支持可伸缩资源（可以同时在多个节点上联机）。

资源组管理器模型

本节介绍了一些基本术语，并对 RGM 及其相关接口进行了详细说明。

RGM 可以处理三大类相关对象：资源类型、资源和资源组。下面通过一个示例来介绍这些对象。

您实现了资源类型 `ha-oracle`，该资源类型使现有的 Oracle DBMS 应用程序具有高可用性。最终用户定义了单独的销售、工程和财务数据库，每个数据库都是 `ha-oracle` 类型的资源。群集管理员将这些资源放入单独的资源组中，这样，这些资源即可在不同的节点上运行并单独进行故障转移。再创建一个资源类型 `ha-calendar` 以实现需要 Oracle 数据库的高可用性的日历服务器。群集管理员将财务日历资源放在财务数据库资源所在的资源组中，以便两个资源在同一节点上运行并同时进行故障转移。

资源类型

资源类型由以下几个部分组成：

- 要在群集中运行的软件应用程序
- 控制程序，被 RGM 用作回调方法以管理作为群集资源的应用程序
- 形成群集的静态配置的一组属性

RGM 使用资源类型属性来管理特定类型的资源。

注 – 除了软件应用程序以外，资源类型还可以表示其他系统资源（例如网络地址）。

可以指定资源类型的属性，还可以在资源类型注册 (RTR) 文件中设置属性值。RTR 文件采用第 31 页中的 “设置资源和资源类型属性”和 `rt_reg`(4) 手册页中介绍的格式。有关样例 RTR 文件的说明，另请参见第 72 页中的 “定义资源类型注册文件”。

第 213 页中的 “资源类型属性”提供了资源类型属性的列表。

群集管理员在群集上安装和注册资源类型实现和基础应用程序。注册过程将把来自 RTR 文件的信息输入到群集配置中。《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》介绍了注册数据服务的过程。

资源

资源将继承其资源类型的属性和值。另外，您还可以在 RTR 文件中声明资源属性。第 220 页中的 “资源属性”包含资源属性的列表。

群集管理员可以根据在 RTR 文件中指定属性的方式更改特定属性的值。例如，属性定义可以指定允许值的范围。属性定义还可以指定属性何时可调：永不、随时、创建时（在将资源添加到群集时）或禁用资源时。在这些指定范围内，群集管理员可以使用管理命令对属性进行更改。

群集管理员可以创建多个类型相同的资源，每个资源都可以有自己的名称和一组属性值，这样，底层应用程序的多个实例就能够在群集中运行。每个实例都要求在群集内具有唯一的名称。

资源组

每个资源都必须在资源组中进行配置。RGM 使组中的所有资源在同一节点上同时联机或脱机。当 RGM 使资源组联机或脱机时，它将对组中的各个资源运行回调方法。

资源组当前处于联机状态所在的节点称为该资源组的**主要节点**或**主节点**。资源组由它的每个主要节点**控制**。每个资源组都有一个关联的 `Nodelist` 属性，该属性用于标识资源组的所有**潜在主要节点**或**主控节点**。群集管理员将设置 `Nodelist` 属性。

资源组还具有一组属性。这些属性包括配置属性（可以由群集管理员进行设置）和动态属性（由 RGM 进行设置，可以反映资源组的活动状态）。

RGM 定义了两类资源组：故障转移资源组和可伸缩资源组。故障转移资源组一次只能在一个节点上联机，而可伸缩资源组可以同时在多个节点上联机。RGM 提供了一组支持资源组每个类型创建的属性。有关这些属性的详细信息，请参见第 31 页中的 “将数据服务传送到群集”和第 39 页中的 “实现回调方法”。

第 233 页中的 “资源组属性”包含资源组属性的列表。

资源组管理器

资源组管理器 (RGM) 将作为在群集的每个成员节点上运行的守护进程 `rgmd` 实现。所有的 `rgmd` 进程可以相互通信，并用作在一个单独的群集范围内使用的工具。

RGM 支持以下功能：

- 无论节点何时引导或崩溃，RGM 都会尝试通过在正确的主控节点上使受管理资源组自动联机，来维护所有受管理资源组的可用性。
- 如果特定资源失效，其监视程序可以请求资源组在同一主控节点上重启或切换到新的主控节点。
- 群集管理员可以发出管理命令以请求执行以下任一操作：
  - 更改资源组的控制权。
  - 启用或禁用资源组内的特定资源。
  - 创建、删除或修改资源类型、资源或资源组。

RGM 激活配置更改后，它将在群集的所有成员节点间协调其操作。此类活动称为**重新配置**。为了使单个资源的状态更改生效，RGM 将对该资源运行资源类型特定的回调方法。

回调方法

Sun Cluster 框架使用回调机制来提供数据服务与 RGM 之间的通信。此框架定义了一组回调方法，这包括它们的参数和返回值，以及 RGM 调用各个方法时的情况。

通过对一组单个回调方法编码并将每个方法作为 RGM 可以调用的控制程序来实现，可以创建数据服务。也就是说，数据服务不是由一个可执行脚本或二进制文件 (C) 组成的，而是由多个可执行脚本 (ksh) 或二进制文件 (C) 组成的，RGM 可以直接调用其中任何一个对象。

回调方法是通过 RTR 文件向 RGM 注册的。在 RTR 文件中，可以标识已经为数据服务实现的每个方法的程序。当群集管理员在群集上注册数据服务时，RGM 将读取 RTR 文件，该文件提供了回调程序的标识以及其他信息。

资源类型必需的回调方法仅包括启动方法（Start 或 Prenet_start）和停止方法（Stop 或 Postnet_stop）。

回调方法可以分为以下几类：

- 控制和初始化方法
  - Start 和 Stop 方法用于启动和停止正被联机或脱机的组中的资源。
  - Init、Fini 和 Boot 方法将执行对资源的初始化和终止代码。
- 管理支持方法
  - Validate 方法用于验证由管理操作设置的属性。
  - Update 方法用于更新联机资源的属性设置。
- 与网络相关的方法

  在同一资源组中的网络地址被配置为“启用”之前或“关闭”之后，Prenet_start 和 Postnet_stop 将执行特殊的启动或关闭操作。
- 监视器控制方法
  - Monitor_start 和 Monitor_stop 用于启动或停止资源的监视器。
  - Monitor_check 用于在将资源组移至节点之前评估该节点的可靠性。

有关回调方法的更多信息，请参见第 3 章和 rt_callbacks(1HA) 手册页。有关样例数据服务中的回调方法，另请参见第 5 章和第 8 章。

编程接口

资源管理体系结构为编写数据服务代码提供了一个低级别 API（即基本 API）、一个构建在基本 API 之上的较高级别的库和 SunPlex Agent Builder 工具，该工具可以根据您提供的基本输入内容自动生成数据服务。

Resource Management API（资源管理 API）

资源管理 API (RMAPI) 提供了一组低级函数，这些函数可以启用数据服务，以访问有关系统中的资源类型、资源和资源组的信息，请求本地重新启动或故障转移，以及设置资源状态。通过 `libscha.so` 库可以访问这些函数。RMAPI 以 shell 命令和 C 函数的形式提供这些回调方法。有关 RMAPI 函数的更多信息，请参见 `scha_calls`(3HA) 手册页和第 3 章。有关如何在样例数据服务回调方法中使用这些函数的示例，另请参见第 5 章。

数据服务开发库

构建在 RMAPI 之上的是数据服务开发库 (DSDL)，它提供了一个较高级别的集成框架，同时保留了 RGM 的底层**方法回调模型**。`libdsdev.so` 库包含 DSDL 函数。DSDL 集成了各种数据服务开发工具，包括：

- `libscha.so`。低级资源管理 API。
- **PMF**。进程监视器工具 (PMF)，它提供了监视进程及其子进程和在进程及其子进程中止时将其重新启动的方法。请参见 `pmfadm`(1M) 和 `rpc.pmfd`(1M) 手册页。
- `hatimerun`。在超时的情况下运行程序的工具。请参见 `hatimerun`(1M) 手册页。

对于大多数应用程序，DSDL 提供了生成数据服务所需的多数或所有功能。但请注意，DSDL 不会替代低级 API，而是封装并扩展它。事实上，许多 DSDL 函数都调用 `libscha.so` 函数。同样地，当使用 DSDL 为数据服务的主体编码时，您可以直接调用 `libscha.so` 函数。

有关 DSDL 的更多信息，请参见第 6 章和 `scha_calls`(3HA) 手册页。

SunPlex Agent Builder

Agent Builder 是自动创建数据服务的工具。您输入有关目标应用程序和要创建的数据服务的基本信息。Agent Builder 将生成一个数据服务，其中包括源代码和可执行代码（C 或 Korn shell）、自定义的 RTR 文件和 Solaris 软件包。

对于多数应用程序，都可以使用 Agent Builder 生成完整的数据服务，且只需您进行很少的手动更改。具有较复杂要求（例如，对附加属性添加验证检查）的应用程序可能需要执行 Agent Builder 无法完成的操作。但是，即使在这些情况下，您也可以使用 Agent Builder 生成代码的主体，然后再手动编写剩余代码。至少，您可以使用 Agent Builder 生成 Solaris 软件包。

资源组管理器管理接口

Sun Cluster 提供了图形用户界面 (GUI) 和一组命令来管理群集。

SunPlex Manager

SunPlex Manager 是基于 Web 的工具，您可以通过它执行以下任务：

- 安装群集。
- 管理群集。
- 创建和配置资源和资源组。
- 使用 Sun Cluster 软件配置数据服务。

有关如何安装 SunPlex Manager 以及如何使用 SunPlex Manager 安装群集软件的说明，请参见《Sun Cluster 软件安装指南（适用于 Solaris OS）》。SunPlex Manager 为多数唯一的管理任务提供了联机帮助。

`scsetup` 实用程序

大多数 Sun Cluster 管理任务都可与 `scsetup`(1M) 实用程序交互执行。

可以使用 `scsetup` 实用程序管理以下 Sun Cluster 元素：

- 定额
- 资源组
- 数据服务
- 群集互连
- 设备组和卷
- 专用主机名
- 新节点
- 其他群集属性

还可以使用 `scsetup` 实用程序执行以下操作：

- 创建资源组
- 将网络资源添加至资源组
- 将数据服务资源添加至资源组
- 注册资源类型
- 使资源组联机或脱机
- 将资源组切换转移
- 启用或禁用资源
- 更改资源组属性
- 更改资源属性
- 从资源组中删除资源

- 删除资源组
- 清除资源的 `Stop_failed` 错误标志

管理命令

管理 RGM 对象的 Sun Cluster 命令包括 `scrgadm`、`scswitch` 和 `scstat -g`。

使用 `scrgadm` 命令，使您可以查看、创建、配置和删除 RGM 所使用的资源类型、资源组和资源对象。此命令属于群集的管理接口，但是不能作为本章剩余部分中介绍的应用程序接口用于同一编程环境中。但是，`scrgadm` 是构造 API 进行操作的群集配置的工具。理解管理接口有助于理解应用程序接口。有关此命令可以执行的管理任务的详细信息，请参见 `scrgadm`(1M) 手册页。

`scswitch` 命令用于将资源组在指定节点上切换为联机和脱机。此命令还可以启用或禁用资源或其监视器。有关此命令可以执行的管理任务的详细信息，请参见 `scswitch`(1M) 手册页。

`scstat -g` 命令用于显示所有资源组和资源的当前动态状态。有关此命令可以执行的管理任务的详细信息，请参见 `scstat`(1M) 手册页。

第 2 章

开发数据服务

本章向您介绍如何使应用程序具有高可用性和可伸缩性，并提供有关开发数据服务的详细信息。

本章包括以下主题：

分析应用程序的适用性

创建数据服务的第一步是确定目标应用程序是否满足具有高可用性或可伸缩性的要求。如果应用程序没有满足所有要求，您可以修改应用程序源代码，使其具有高可用性或可伸缩性。

下表列出了要具有高可用性或可伸缩性的应用程序需要满足的要求。如果需要详细信息或者需要修改应用程序源代码，请参见附录 B。

注 – 为了具有高可用性，可伸缩服务必须满足以下所有条件，同时还需满足下列项之后的一些附加条件。

- 在 Sun Cluster 环境中，网络可识别（客户机-服务器模型）和非网络可识别（无客户机）应用程序都有可能具有高可用性或可伸缩性。但是，在分时环境中，Sun Cluster 不能提供增强型可用性，在该环境下，应用程序在通过 `telnet` 或 `rlogin` 访问的服务器上运行。
- 该应用程序必须具有崩溃容限能力。也就是说，如果在节点发生意外毁坏的情况后启动应用程序，该应用程序必须能够在启动时恢复磁盘数据（如果有必要）。而且，崩溃后的恢复时间必须在限定范围内。崩溃容限是使应用程序具有高可用性的前提条件，因为恢复磁盘和重启应用程序的功能实质上是为了保持数据的完整性。不要求数据服务具有恢复连接的功能。
- 该应用程序不能依赖于运行时所在节点的物理主机名。有关附加信息，请参见第 299 页中的 “主机名”。
- 应用程序必须能够在多个 IP 地址被配置为“启用”的环境中正常运行。例如，节点位于多个公共网络中的多宿主 (multihomed) 主机环境，以及在一个硬件接口上多个逻辑接口被配置为“启用”的节点环境。
- 要具有高可用性，应用程序数据必须位于群集文件系统中。请参见第 297 页中的 “多主机数据”。

  如果应用程序针对数据位置使用硬连线路径名，您可以将该路径更改为指向群集文件系统中某一位置的符号链接，而无需更改应用程序源代码。有关附加信息，请参见第 298 页中的 “将符号链接用于多主机数据放置”。
- 应用程序二进制数据和库都可以位于各个本地节点或本地群集文件系统中。位于群集文件系统中的优点在于进行单个安装就可以了。缺点在于进行滚动升级时，应用程序在 RGM 的控制下运行的同时，也会使用二进制数据。
- 客户机应该具有在首次查询尝试超时后自动重试的功能。如果应用程序和协议已经处理了单个服务器崩溃和重新引导的问题，则可以处理有关资源组进行故障转移或切换的问题。有关附加信息，请参见第 300 页中的 “客户机重试”。
- 应用程序在群集文件系统中不能具有 UNIX® 域套接字或命名管道。

此外，可伸缩服务必须满足以下要求：

- 应用程序必须具有运行多个实例的能力，所有实例都在群集文件系统中相同的应用程序数据上进行操作。
- 应用程序必须保持数据一致性，以便从多个节点同时进行访问。
- 应用程序必须通过全局可视机制（例如群集文件系统）实现充分锁定。

对于可伸缩服务，应用程序的特征也可以确定负载平衡策略。例如，允许任一实例响应客户机请求的负载平衡策略 `Lb_weighted` 不能用于使用服务器内存中的高速缓存进行客户机连接的应用程序。在这种情况下，您应该指定一个负载平衡策略，以限制指定客户机到应用程序的一个实例的通信。负载平衡策略 `Lb_sticky` 和 `Lb_sticky_wild` 将反复地把客户机发出的所有请求发送到同一应用程序实例，在该实例中请求可以使用内存中的高速缓存。 请注意，如果传入的多个客户机请求来自不同的客户机，RGM 将在服务的实例中分配这些请求。有关设置可伸缩数据服务的负载平衡策略的更多信息，请参见第 45 页中的 “实现故障切换资源”。

确定要使用的接口

Sun Cluster 开发者支持软件包 (SUNWscdev) 提供了两组用来对数据服务方法进行编码的接口：

- 资源管理 API (RMAPI)，一组低层次函数（在 libscha.so 库中）
- 数据服务开发库 (DSDL)，一组较高层次函数（在 libdsdev.so 库中）可以封装 RMAPI 的功能并提供一些附加功能

Sun Cluster 开发者支持软件包中还提供了一个用来自动创建数据服务的工具 SunPlex Agent Builder。

以下是开发数据服务的建议方法：

1. 决定是在 C shell 中还是在 Korn shell 中进行编码。如果决定使用 Korn shell，则不能使用仅提供 C 接口的 DSDL。
2. 运行 Agent Builder，指定所需的信息并生成数据服务，其中包括源代码和可执行代码、一个 RTR 文件和一个软件包。
3. 如果需要定制生成的数据服务，您可以向生成的源文件添加 DSDL 代码。Agent Builder 将通过注释指明源文件中可添加用户自己的代码的特定位置。
4. 如果需要进一步定制代码以支持目标应用程序，您可以向现有源代码添加 RMAPI 函数。

在实际情况中，您可以采用多种方法来创建数据服务。例如，除了向 Agent Builder 生成的代码中的特定位置添加自己的代码，您还可以用使用 DSDL 或 RMAPI 函数临时编写的程序完全替换生成的某一方法或生成的监视程序。但是，无论采取哪种方法，在绝大多数情况下从 Agent Builder 出发都很有效，原因如下：

- Agent Builder 生成的代码（在本质上是通用的）已在多个数据服务中进行了测试。
- Agent Builder 可生成 RTR 文件 makefile、资源软件包和数据服务的其他支持文件。即使您不使用任何数据服务代码，使用上述其他文件也能省去相当多的工作量。
- 您可以修改生成的代码。

注 – 与 RMAPI 提供一组用于脚本的 C 函数和命令不同，DSDL 仅提供了一个 C 函数接口。因此，如果在 Agent Builder 中指定 Korn shell (ksh) 输出，生成的源代码将调用 RMAPI，因为没有 DSDL ksh 命令。

设置用来编写数据服务的开发环境

开始进行数据服务开发之前，您必须安装 Sun Cluster 开发软件包 (SUNWscdev)，以便可以访问 Sun Cluster 头文件和库文件。虽然该软件包已安装在所有群集节点上，但是您通常是在一个独立的、非群集开发计算机上进行数据服务开发，而不是在群集节点上进行。在此情况下，您必须使用 pkgadd 命令在开发计算机上安装 SUNWscdev 软件包。

当编译和链接代码时，您必须设置特定选项来标识头文件和库文件。

注 – 不可在 Solaris 操作系统和 Sun Cluster 产品中同时使用兼容模式编译的 C++ 代码和标准模式编译的 C++ 代码。因此，如果要创建在 Sun Cluster 上使用的基于 C++ 的数据服务，您必须对该数据服务进行编译，如下所示：

- 对于 Sun Cluster 3.0 和早期版本，请使用兼容模式。
- 对于 Sun Cluster 3.1 和更高版本，请使用标准模式。

在非群集节点上完成开发后，可以将已完成的数据服务传送到群集以进行测试。

注 – 请确保使用的是 Solaris 8 操作系统（或更高版本）的 Developer 或 Entire Distribution 软件组。

本节中的过程将介绍如何完成以下任务：

- 安装 Sun Cluster 开发软件包 (SUNWscdev) 以及设置正确的编译器和链接程序选项。
- 将数据服务传送到群集。

▼ 如何设置开发环境

此步骤介绍了如何安装 SUNWscdev 软件包和设置用于数据服务开发的编译器和链接程序选项。

步骤 **1. 成为超级用户或作为等效角色。**

2. 将目录更改为所需的 CD-ROM 目录。

```
# cd cd-rom-directory
```

3. 在当前目录下安装 SUNWscdev 软件包。

- **针对区环境中的 Solaris 10 操作系统，在全局区域中以全局管理员身份键入以下命令：**

  ```
  # pkgadd -G -d . SUNWscdev
  ```

  如果 SUNWscdev 的内容对全局区域中与非全局区域共享的区域没有任何影响，则将 SUNWscdev 软件包添加到全局区域。

- **针对非区环境中的 Solaris 操作系统的其他任何版本或 Solaris 10 操作系统，键入以下命令：**

  ```
  # pkgadd -d . SUNWscdev
  ```

4. **在 `makefile` 中，指定用于标识数据服务代码的库文件和其他文件的编译器和链接程序选项。**

   指定 -I 选项用于标识 Sun Cluster 头文件、-L 选项用于指定开发系统上的编译时库搜索路径、-R 选项用于指定群集中运行时链接程序的库搜索路径。

   ```
   # Makefile for sample data service
   ...

   -I /usr/cluster/include

   -L /usr/cluster/lib

   -R /usr/cluster/lib
   ...
   ```

将数据服务传送到群集

在开发计算机上完成数据服务后，您必须将其传送到群集以进行测试。为了降低在传送过程中发生错误的可能性，请将数据服务代码和 RTR 文件合并成软件包，然后将该软件包安装在群集的所有节点上。

注 – 无论是使用 pkgadd 还是使用其它方法来安装数据服务，您必须将数据服务安装在所有群集节点上。请注意，Agent Builder 将自动创建此软件包。

设置资源和资源类型属性

Sun Cluster 提供了一组用来定义数据服务静态配置的资源类型属性和资源属性。资源类型属性可指定资源的类型及其版本、API 版本和每个回调方法的路径。第 213 页中的 “资源类型属性”中列出了所有资源类型属性。

资源属性（例如 Failover_mode、Thorough_probe_interval 和方法超时）还定义资源的静态配置。动态资源属性（例如 Resource_state 和 Status）反映受管理资源的活动状态。第 220 页中的 “资源属性”中介绍了资源属性。

资源类型和资源属性在资源类型注册 (RTR) 文件中声明，该文件是数据服务的基本组件。RTR 文件用于在群集管理员使用 Sun Cluster 注册数据服务时定义数据服务的初始配置。

请使用 Agent Builder 生成数据服务的 RTR 文件，这是因为 Agent Builder 声明了一组对任何数据服务既有用又必需的属性。例如，某些特殊属性（如 `Resource_type`）必须在 RTR 文件中声明。否则，数据服务注册将失败。其他属性尽管没有此方面的要求，但是如果不在 RTR 文件中声明它们，群集管理员将无法使用。而某些属性无论是否进行了声明都可以使用，这是因为 RGM 对其进行了定义并为其提供默认值。为了避免这种复杂的情况，请使用 Agent Builder 以确保生成正确的 RTR 文件。如果需要，您可以在以后对该 RTR 文件进行编辑，以更改特定值。

本节的其余部分介绍了由 Agent Builder 创建的 RTR 文件样例。

声明资源类型属性

群集管理员无法配置您在 RTR 文件中声明的资源类型属性。这些属性将成为永久资源类型配置中的一部分。

注 – 资源类型属性 `Installed_nodes` 只能由群集管理员进行配置。您无法在 RTR 文件中声明 `Installed_nodes`。

资源类型声明的语法如下所示：

property-name = *value*;

注 – 资源组、资源和资源类型的属性名称**不**区分大小写。在指定属性名称时，您可以使用大小写字母的任意组合。

以下是 RTR 文件中用于样例 (`smpl`) 数据服务的资源类型声明：

```
# Sun Cluster Data Services Builder template version 1.0
# Registration information and resources for smpl
#
#NOTE: Keywords are case insensitive, i.e., you can use
#any capitalization style you prefer.
#
Resource_type = "smpl";
Vendor_id = SUNW;
RT_description = "Sample Service on Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;
```

```
Init_nodes = RG_PRIMARIES;

RT_basedir=/opt/SUNWsmpl/bin;

Start           =    smpl_svc_start;
Stop            =    smpl_svc_stop;

Validate        =    smpl_validate;
Update          =    smpl_update;

Monitor_start   =    smpl_monitor_start;
Monitor_stop    =    smpl_monitor_stop;
Monitor_check   =    smpl_monitor_check;
```

提示 – 您必须将 `Resource_type` 属性声明为 RTR 文件中的第一项。否则，资源类型注册将失败。

第一组资源类型声明提供了有关资源类型的基本信息。

`Resource_type` 和 `Vendor_id`
提供资源类型的名称。您可以仅使用 `Resource_type` 属性指定资源类型名称 (`smpl`)，或使用 `Vendor_id` 属性作为前缀，并用“.”将该属性与资源类型分隔开 (`SUNW.smpl`)，如样例中所示。如果您使用 `Vendor_id`，请使其成为公司用于定义资源类型的股市符号。在群集中资源类型的名称必须唯一。

注 – 按照约定，资源类型名称 (*vendoridApplicationname*) 用作软件包名称。从 Solaris 9 操作系统开始，供应商 ID 和应用程序名称的组合可以超过九个字符。但是，如果您使用的是 Solaris 操作系统的早期版本，则供应商 ID 和应用程序名称的组合不能超过九个字符（尽管 RGM 并没有强制此限制）。

另外，在所有情况下，Agent Builder 都明确地根据资源类型名称生成软件包名称，因此它将强制执行九个字符的限制。

`RT_description`
简要地介绍资源类型。

`RT_version`
用于标识数据服务样例的版本。

`API_version`
用于标识 API 的版本。例如，`API_version = 2` 表明数据服务可以安装在从 Sun Cluster 3.0 开始的 Sun Cluster 的所有版本上；`API_version = 5` 表明数据服务可以安装在从 Sun Cluster 3.1 9/04 开始的 Sun Cluster 的所有版本上。但是，`API_version = 5` 还表明数据服务无法安装在 Sun Cluster 3.1 9/04 发行之前的 Sun Cluster 的所有版本上。有关该属性的详细信息，请参见第 213 页中的 “资源类型属性”中有关 `API_version` 条目下的内容。

`Failover = TRUE`
表明数据服务无法在可同时在多个节点上联机的资源组中运行。换句话说，该声明指定了一个故障转移数据服务。有关该属性的详细信息，请参见第 213 页中的 “资源类型属性”中有关 `Failover` 条目下的内容。

`Start`、`Stop` 和 `Validate`
提供由 RGM 调用的各个回调方法程序的路径。这些路径是 `RT_basedir` 所指定的目录的相对路径。

其余的资源类型声明提供了配置信息。

`Init_nodes = RG_PRIMARIES`
指定 RGM 仅在可以主控数据服务的节点上调用 `Init`、`Boot`、`Fini` 和 `Validate` 方法。`RG_PRIMARIES` 所指定的节点是安装有数据服务的所有节点的一个子集。将值设置为 `RT_INSTALLED_NODES` 可指定 RGM 在安装有数据服务的所有节点上调用这些方法。

`RT_basedir`
指向 `/opt/SUNWsample/bin`，作为指向完整相对路径（例如回调方法路径）的目录路径。

`Start`、`Stop` 和 `Validate`
提供指向由 RGM 调用的各个回调方法程序的路径。这些路径是 `RT_basedir` 所指定的目录的相对路径。

声明资源属性

与资源类型属性一样，资源属性也在 RTR 文件中声明。按照惯例，在 RTR 文件中资源属性声明位于资源类型声明之后。资源声明的语法是一组用花括号括起来的属性值对：

```
{
    attribute = value;
    attribute = value;
                .
                .
                .
    attribute = value;
}
```

对于 Sun Cluster 提供的资源属性（称为**系统定义的**属性），您可以在 RTR 文件中更改特定属性。例如，Sun Cluster 为每个回调方法的方法超时属性提供默认值。在 RTR 文件中，您可以指定各种缺省值。

您还可以使用一组由 Sun Cluster 提供的属性的属性，在 RTR 文件中定义新的资源属性（称为**扩展**属性）。第 240 页中的 “资源特性属性”中列出了用于更改和定义资源属性的属性。在 RTR 文件中，扩展属性声明位于系统定义的属性声明之后。

第一组系统定义的资源属性指定了回调方法的超时值。

```
...

# Resource property declarations appear as a list of bracketed
```

```
# entries after the resource type declarations. The property
# name declaration must be the first attribute after the open
# curly bracket of a resource property entry.
#
# Set minimum and default for method timeouts.
{
        PROPERTY = Start_timeout;
        MIN=60;
        DEFAULT=300;
}

{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Update_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Stop_timeout;
        MIN=60;
        DEFAULT=300;
{
        PROPERTY = Monitor_Check_timeout;
        MIN=60;
        DEFAULT=300;
}
```

属性名称 (`PROPERTY = value`) 必须是每个资源属性声明的第一个属性。您可以在由 RTR 文件中属性的属性定义的限制内配置资源属性。例如，样例中每个方法超时的缺省值都是 300 秒。群集管理员可以更改此值。但是，由 `MIN` 属性指定的最小允许值是 60 秒。第 240 页中的 “资源特性属性”中包含资源属性的属性列表。

下一组资源属性用于定义在数据服务中有特定用法的属性。

```
{
        PROPERTY = Failover_mode;
        DEFAULT=SOFT;
        TUNABLE = ANYTIME;
}
{
        PROPERTY = Thorough_Probe_Interval;
```

```
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Set Retry_interval as a multiple of 60 since it is converted from seconds
# to minutes, rounding up. For example, a value of 50 (seconds)
# is converted to 1 minute. Use this property to time the number of
# retries (Retry_count).
{
        PROPERTY = Retry_interval;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

{
        PROPERTY = Network_resources_used;
        TUNABLE = WHEN_DISABLED;
        DEFAULT = "";
}
{
        PROPERTY = Scalable;
        DEFAULT = FALSE;
        TUNABLE = AT_CREATION;
}
{
        PROPERTY = Load_balancing_policy;
        DEFAULT = LB_WEIGHTED;
        TUNABLE = AT_CREATION;
}
{
        PROPERTY = Load_balancing_weights;
        DEFAULT = "";
        TUNABLE = ANYTIME;
}
{
        PROPERTY = Port_list;
        TUNABLE = ANYTIME;
        DEFAULT = ;
}
```

这些资源属性声明包含 TUNABLE 属性。该属性将限制在哪种情况下群集管理员可以更改与该属性相关的属性值。例如，值 AT_CREATION 表明群集管理员仅能在创建资源时指定值，并且以后无法更改此值。

对于这些属性中的大多数，您可以接受由 Agent Builder 生成的默认值，除非您有理由更改它们。以下是有关这些属性的信息。有关附加信息，请参见第 220 页中的 “资源属性”或 r_properties(5) 手册页。

Failover_mode
: 表明在 Start 或 Stop 方法失败的情况下，RGM 是应重新查找资源组还是应异常中止节点。

Thorough_probe_interval、Retry_count 和 Retry_interval
: 用于故障监视器。Tunable 等于 ANYTIME，因此如果故障监视器不能发挥最佳工作效果，群集管理员可以对它们进行调整。

Network_resources_used
: 数据服务使用的逻辑主机名或共享地址资源的列表。Agent Builder 将声明该属性，以便群集管理员对数据服务进行配置时，可以指定资源列表（如果有）。

Scalable
: 设置为 FALSE 以表明该资源没有使用群集网络（共享地址）设备。如果将该属性设置为 FALSE，资源类型属性 Failover 必须设置为 TRUE 以表明故障转移服务。有关如何使用该属性的附加信息，请参见第 31 页中的 “将数据服务传送到群集”和第 39 页中的 “实现回调方法”。

Load_balancing_policy 和 Load_balancing_weights
: 自动声明这些属性。但是，这些属性在故障转移资源类型中不起作用。

Port_list
: 标识服务器侦听的端口的列表。Agent Builder 将声明该属性，以便群集管理员对数据服务进行配置时，可以指定端口列表。

声明扩展属性

扩展属性位于样例 RTR 文件的结尾处。

```
# Extension Properties
#

# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, smpl, specify the path of the configuration file on
# PXFS (typically named.conf).
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "The Configuration Directory Path(s)";
}

# The following two properties control restart of the fault monitor.
{
        PROPERTY = Monitor_retry_count;
        EXTENSION;
```

```
        INT;
        DEFAULT = 4;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Number of PMF restarts allowed for fault monitor.";
}
{
        PROPERTY = Monitor_retry_interval;
        EXTENSION;
        INT;
        DEFAULT = 2;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time window (minutes) for fault monitor restarts.";
}
# Time out value in seconds for the probe.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 120;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time out value for the probe (seconds)";
}

# Child process monitoring level for PMF (-C option of pmfadm).
# Default of -1 means to not use the -C option of pmfadm.
# A value of 0 or greater indicates the desired level of child-process.
# monitoring.
{
        PROPERTY = Child_mon_level;
        EXTENSION;
        INT;
        DEFAULT = -1;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Child monitoring level for PMF";
}
# User added code -- BEGIN VVVVVVVVVVVV
# User added code -- END   ^^^^^^^^^^^^
```

Agent Builder 将创建以下扩展属性，这些属性对于大多数数据服务来说都有用。

`Confdir_list`
指定应用程序配置目录的路径，这对于很多应用程序是有用的信息。当群集管理员对数据服务进行配置时，可以提供该目录的位置。

`Monitor_retry_count`、`Monitor_retry_interval` 和 `Probe_timeout`
控制故障监视器本身的重新启动操作，而非服务器守护进程。

`Child_mon_level`
设置 PMF 要执行的监视级别。有关更多信息，请参见 `pmfadm`(1M) 手册页。

您可以在由 `User added code` 注释分界的区域内创建其他扩展属性。

实现回调方法

本节提供有关实现回调方法的一般信息。

访问资源和资源组属性信息

通常，回调方法需要访问资源属性。RMAPI 同时提供了可在回调方法中使用的 shell 命令和 C 函数，以访问系统定义的资源属性和扩展资源属性。请参见 `scha_resource_get`(1HA) 和 `scha_resource_get`(3HA) 手册页。

DSDL 提供一组用于访问系统定义的属性的 C 函数（每种属性一个函数）和一个用于访问扩展属性的函数。请参见 `scds_property_functions`(3HA) 和 `scds_get_ext_property`(3HA) 手册页。

您无法使用属性机制存储数据服务的动态状态信息，因为没有 API 函数可用于设置 `Status` 和 `Status_msg` 以外的资源属性。您应该将动态状态信息存储在全局文件中。

注 – 群集管理员可以使用 `scrgadm` 命令或通过图形管理命令或界面来设置特定资源属性。但是，请勿从任何回调方法中调用 `scrgadm`，这是因为在群集重新配置期间（即 RGM 调用方法时）`scrgadm` 将失败。

方法的幂等性

通常情况下，RGM 不会使用相同的参数对同一资源连续多次调用某个方法。但是，如果 `Start` 方法失败，RGM 会对资源调用 `Stop` 方法，即使该资源从未启动过。同样，即使资源守护进程能够终止其自身，RGM 可能仍会对其运行 `Stop` 方法。相同的情况也适用于 `Monitor_start` 和 `Monitor_stop` 方法。

由于这些原因，您必须在 `Stop` 和 `Monitor_stop` 方法中构建幂等性。使用相同参数对同一资源重复调用 `Stop` 或 `Monitor_stop` 的结果与调用一次的结果相同。

幂等性的一个含义是使 `Stop` 和 `Monitor_stop` 必须返回 0（成功），即使这时资源或监视器已经停止并且无工作要完成。

注 – `Init`、`Fini`、`Boot` 和 `Update` 方法也必须具有幂等性。`Start` 方法无需具有幂等性。

普通数据服务

普通数据服务 (GDS) 是一种机制，通过将简单的应用程序插入到 Sun Cluster 资源组管理器框架中，可以使其具有高可用性或可伸缩性。该机制不需要对数据服务进行编码，而编码是使应用程序具有高可用性或可伸缩性的常用方法。

GDS 模型依赖于预编译的资源类型 `SUNW.gds`，以与 RGM 框架进行交互式操作。有关附加信息，请参见第 10 章。

控制应用程序

每当节点加入或退出群集时，回调方法将启用 RGM 来控制基本资源（即应用程序）。

启动和停止资源

资源类型实现至少需要使用 `Start` 方法和 `Stop` 方法。RGM 将于适当时候在适当的节点上调用资源类型的方法程序，以使资源组脱机或联机。例如，群集节点崩溃后，RGM 将把该节点主控的所有资源组移到新的节点上。您必须实现 `Start` 方法，以通过重启未崩溃主机节点上每个资源的方法来提供 RGM。

在资源启动并可用于本地节点之前不能返回 `Start`方法。请确保需要较长初始化时间的资源类型的 `Start` 方法设置了足够长的超时。要确保超时足够，请在 RTR 文件中设置 `Start_timeout` 属性的默认值和最小值。

在 RGM 使资源组脱机的情况下，您必须实现 `Stop` 方法。例如，假设某个资源组在 Node1 上脱机，在 Node2 上恢复联机。在使资源组脱机的同时，RGM 将对该组中的资源调用 `Stop` 方法以停止 Node1 上的所有活动。当在 Node1 上对所有资源执行完 `Stop` 方法后，RGM 将使资源组在 Node2 上恢复联机。

在资源完全停止本地节点上的所有活动并完全关闭之前，`Stop` 方法不能返回。`Stop` 方法的最安全实现将终止与该资源相关的本地节点上的所有进程。需要较长时间进行关闭的资源类型的 `Stop` 方法应设置了足够长的超时。在 RTR 文件中设置 `Stop_timeout` 属性。

`Stop` 方法的失败或超时将导致资源组进入需要群集管理员介入的错误状态。要避免进入这种状态，`Stop` 和 `Monitor_stop` 方法实现应该尝试从所有可能出错的情况下恢复。理想的情况是，成功地停止本地节点上的资源及其监视器的所有活动之后，这些方法将在 0（成功）错误状态下退出。

确定使用哪种 Start 或 Stop 方法

与使用 Prenet_start 和 Postnet_stop 方法相比，本节介绍了一些有关何时使用 Start 和 Stop 方法的提示。您必须对客户机和数据服务的客户机-服务器联网协议的知识都有深入了解，才能确定要使用的正确方法。

使用网络地址资源的服务可能需要按照与逻辑主机名地址配置相对的特定顺序来执行启动或停止步骤。可选的回调方法 Prenet_start 允许资源类型实现在将同一资源组中的网络地址配置为启用之前执行特定的启动操作，而可选的回调方法 Postnet_stop 允许资源类型实现在将同一资源组中的网络地址配置为关闭之后执行特定的关闭操作。

在调用数据服务的 Prenet_start 方法之前，RGM 调用检测网络地址（但不将网络地址配置为启用）的方法。在调用数据服务的 Postnet_stop 方法之后，RGM 调用不检测网络地址的方法。RGM 使资源组联机时，采用以下顺序：

1. 检测网络地址。
2. 调用数据服务的 Prenet_start 方法（如果有）。
3. 将网络地址配置为启用。
4. 调用数据服务的 Start 方法（如果有）。

RGM 使资源组脱机时采用相反的顺序：

1. 调用数据服务的 Stop 方法（如果有）。
2. 将网络地址配置为关闭。
3. 调用数据服务的 Postnet_stop 方法（如果有）。
4. 取消检测网络地址。

在决定使用以下哪一个方法时，请首先考虑服务器端：Start、Stop、Prenet_start 或 Postnet_stop 方法。当使同时包含数据服务应用程序资源和网络地址资源的资源组联机时，RGM 将在调用该数据服务资源的 Start 方法之前，调用用来将网络地址配置为启用的方法。因此，如果数据服务需要在启动时将网络地址配置为启用，请使用 Start 方法启动该数据服务。

同样，当使同时包含数据服务资源和网络地址资源的资源组脱机时，RGM 将在调用数据服务资源的 Stop 方法之后，调用用来将网络地址配置为关闭的方法。因此，如果数据服务需要在停止时将网络地址配置为关闭，请使用 Stop 方法来停止该数据服务。

例如，要启动或关闭某个数据服务，您可能需要运行该数据服务的管理实用程序或库。有时，数据服务具有使用客户机-服务器网络接口进行管理的管理实用程序或库。即管理公用程序将调用服务器守护进程，因此可能需要将网络地址配置为启用来使用管理公用程序或库。在此情况中使用 Start 和 Stop 方法。

如果数据服务需要在启动和停止时将网络地址配置为关闭，请使用 Prenet_start 和 Postnet_stop 方法来启动和停止该数据服务。请考虑在进行群集重新配置（带有 SCHA_GIVEOVER 参数的 scha_control() 或带有 scswitch 命令的切换）之后，网络地址还是数据服务首先联机的不同情况会不会使客户机软件做出不同响应。例如，客户机实现可能进行最小限度的重试，当确定数据服务端口不可用后将立即放弃重试。

如果数据服务不需要在启动时将网络地址配置为启用，请在将网络接口配置为启用之前启动该数据服务。使用此方法启动数据服务，可确保当网络地址配置为启用时，数据服务能够立即响应客户机请求。其结果是，客户机不太可能停止重试。在这种情况下，请勿使用 `Start` 方法，最好使用 `Prenet_start` 方法来启动数据服务。

如果使用 `Postnet_stop` 方法，在将网络地址配置为关闭时，数据服务资源仍然可用。仅在将网络地址配置为关闭后，才能运行 `Postnet_stop` 方法。其结果是，数据服务的 TCP 或 UDP 服务端口或其 RPC 程序号始终可供网络上的客户机使用（除非网络地址也未响应）。

注 – 如果在群集中安装 RPC 服务，则该服务不能使用以下程序号：100141、100142 和 100248。这些编号分别专用于 Sun Cluster 守护进程 `rgmd_receptionist`、`fed` 和 `pmfd`。如果您安装的 RPC 服务使用上述程序号之一，请更改该 RPC 服务的程序号。

与使用 `Prenet_start` 和 `Postnet_stop` 方法相比，决定使用 `Start` 和 `Stop` 方法时，或者决定同时使用这两套方案，必须考虑服务器和客户机的要求和行为。

`Init`、`Fini` 和 `Boot` 方法

`Init`、`Fini` 和 `Boot` 这三种可选方法可启用 RGM 对资源执行初始化和终止代码。

当资源受管理后出现以下一种情况时，RGM 将运行 `Init` 方法对该资源执行一次性初始化：

- 该资源所在的资源组由非管理状态切换为受管理状态。
- 该资源在已处于受管理状态下的资源组中创建。

当资源不受管理后出现以下一种情况时，RGM 将运行 `Fini` 方法清理该资源：

- 该资源所在的资源组切换为非管理状态。
- 该资源从受管理资源组中删除。

清理操作必须具有幂等性。即如果已经完成了清理操作，`Fini` 将成功退出。

RGM 在新加入群集的节点（即刚刚对这些节点进行了引导或重新引导）上运行 `Boot` 方法。

使用 `Boot` 方法执行的初始化操作通常与使用 `Init` 执行的初始化操作相同。此初始化操作必须具有幂等性，即如果资源已在本地节点上进行了初始化，则 `Boot` 和 `Init` 将成功退出。

监视资源

通常需要使用监视器，以在资源上运行周期性故障探测来检测所探测的资源是否工作正常。如果故障探测失败，该监视器可以尝试在本地重新启动或请求对受影响的资源组进行故障转移。该监视器可通过调用 `scha_control()` RMAPI 函数或 `scds_fm_action()` DSDL 函数请求故障转移。

您还可以监视资源的性能，并调谐或报告性能。编写特定于资源类型的故障监视器是可选操作。即使您选择不编写故障监视器，Sun Cluster 本身进行的基本群集监视操作也会为资源类型带来好处。Sun Cluster 将检测主机硬件的故障、主机操作系统的严重故障以及主机的故障，以便可以在其公共网络上进行通信。

尽管 RGM 并不直接调用资源监视器，但是 RGM 为资源提供了自动启动的监视器。当使资源脱机时，RGM 将在停止资源本身之前，调用 `Monitor_stop` 方法来停止该资源在本地节点上的监视器。当使资源联机时，RGM 将在启动资源本身之后，调用 `Monitor_start` 方法。

`scha_control()` RMAPI 函数和 `scds_fm_action()` DSDL 函数（它调用 `scha_control()`）允许资源监视器请求将资源组故障转移到另一个节点上。作为它的一次完整性检查，`scha_control()` 将调用 `Monitor_check`（如果已定义）来确定请求的节点在主控包含资源的资源组方面是否足够可靠。如果 `Monitor_check` 报告该节点不可靠或方法超时，RGM 将寻找另一个节点来接收故障转移请求。如果 `Monitor_check` 在所有节点上都失败，将取消该故障转移操作。

资源监视器可以设置 `Status` 和 `Status_msg` 属性来反映监视器所监测到的资源状态视图。使用 `scha_resource_setstatus()` RMAPI 函数、`scha_resource_setstatus` 命令或 `scds_fm_action()` DSDL 函数来设置这些属性。

注 – 尽管 `Status` 和 `Status_msg` 属性对于资源监视器有特定的作用，但是任何程序均可以设置这些属性。

要获得使用 RMAPI 实现的故障监视器的示例，请参见第 86 页中的 “定义故障监视器”。要获得使用 DSDL 实现的故障监视器的示例，请参见第 125 页中的 “SUNW.xfnts 故障监视器”。有关在 Sun 提供的数据服务中内置的故障监视器的信息，请参见《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》。

向资源添加消息日志

如果您希望将状态消息记录到记录其他群集消息的同一个日志文件中，请使用方便的函数 `scha_cluster_getlogfacility()` 来检索用来记录群集消息的工具号。

使用此工具号和常规的 Solaris `syslog()` 函数将消息写入群集日志。您也可以通过普通的 `scha_cluster_get()` 接口访问群集日志工具信息。

提供进程管理

RMAPI 和 DSDL 提供了进程管理工具来实现资源监视器及资源控制回调。RMAPI 定义了以下工具：

进程监视工具 (PMF)：`pmfadm` 和 `rpc.pmfd`
: 提供了监视进程及其子进程以及在失败后重新启动进程的方法。该工具由 `pmfadm` 命令（用于启动和控制受监视的进程）和 `rpc.pmfd` 守护进程组成。

  DSDL 提供一组用于实现 PMF 功能的函数（以 `scds_pmf_` 名称开头）。有关 DSDL PMF 功能的概述和各个函数的列表，请参见第 182 页中的 “PMF 函数”。

  有关该命令和守护进程的详细信息，请参见 `pmfadm`(1M) 和 `rpc.pmfd`(1M) 手册页。

`halockrun`
: 一种用于在保留文件锁定时运行子程序的程序。此命令在 shell 脚本中使用很方便。

  有关此命令的详细信息，请参见 `halockrun`(1M) 手册页。

`hatimerun`
: 用于在超时控制下运行子程序的程序。此命令在 shell 脚本中使用很方便。

  DSDL 提供了用于实现 `hatimerun` 功能的 `scds_hatimerun()` 函数。

  有关此命令的详细信息，请参见 `hatimerun`(1M) 手册页。

提供资源的管理支持

群集管理员对资源执行的操作包括设置和更改资源属性。API 定义了 `Validate` 和 `Update` 回调方法，以便您可以创建挂钩到这些管理操作中的代码。

当创建资源时以及群集管理员更新资源或其包含的组的属性时，RGM 将调用可选的 Validate 方法。RGM 将把该资源及其资源组的属性值传送到 Validate 方法。RGM 将在由资源类型的 Init_nodes 属性表示的一组群集节点上调用 Validate。有关 Init_nodes 的信息，请参见第 213 页中的 “资源类型属性”或 rt_properties(5) 手册页。在进行创建或更新之前，RGM 将调用 Validate。在任一节点上来自方法的故障退出代码将导致创建或更新失败。

仅当群集管理员更改资源或组属性时（而不是在 RGM 设置属性时，或者监视器设置 Status 和 Status_msg 资源属性时），RGM 才调用 Validate。

RGM 将调用可选的 Update 方法来通知运行的资源已对属性进行了更改。群集管理员成功设置了资源或其组的属性后，RGM 将运行 Update。RGM 将对资源处于联机状态的节点调用此方法。此方法可用来通过 API 访问函数读取可能会影响活动资源的属性值，并相应地调整运行资源。

实现故障切换资源

故障转移资源组包含网络地址（例如内置资源类型 LogicalHostname 和 SharedAddress）和故障转移资源（例如用于故障转移数据服务的数据服务应用程序资源）。当数据服务进行故障转移或切换时，网络地址资源及其依赖的数据服务资源在群集节点之间移动。RGM 提供了多个支持故障切换资源实现的属性。

将布尔 Failover 资源类型属性设置为 TRUE，以限制资源在可同时在多个节点上联机的资源组中进行配置。该属性的默认值为 FALSE，因此您必须在 RTR 文件中将其声明为 TRUE，以用于故障转移资源。

Scalable 资源属性用于确定资源是否使用群集共享地址工具。对于故障转移资源，请将 Scalable 设置为 FALSE，这是因为故障转移资源不使用共享地址。

RG_mode 资源组属性允许群集管理员将资源组标识为故障转移或可伸缩。如果 RG_mode 的值为 FAILOVER，则 RGM 将资源组的 Maximum_primaries 属性设置为 1，并将该资源组限制为由单个节点主控。RGM 不允许在其 RG_mode 的值为 SCALABLE 的资源组中创建其 Failover 属性的值为 TRUE 的资源。

Implicit_network_dependencies 资源组属性用于指定 RGM 应该在资源组内的所有网络地址资源（LogicalHostname 和 SharedAddress）上强制执行非网络地址资源的隐含强依赖性。其结果是，在资源组中的网络地址配置为启用之前，将不会调用该组中的非网络地址（数据服务）资源的 Start 方法。
Implicit_network_dependencies 属性的默认值为 TRUE。

实现可伸缩资源

可伸缩资源可以同时在多个节点上联机。可伸缩资源包括数据服务，例如，Sun Cluster HA for Sun Java System Web Server（以前称为 Sun Cluster HA for Sun ONE Web Server）和 Sun Cluster HA for Apache。

RGM 提供了许多支持可伸缩资源实现的属性。

将布尔 `Failover` 资源类型属性设置为 `FALSE`，将允许在可同时在多个节点上联机的资源组中配置资源。

`Scalable` 资源属性用于确定资源是否使用群集共享地址工具。将该属性设置为 `TRUE`，这是因为可伸缩服务使用共享地址资源，以将该可伸缩服务的多个实例显示为客户机的单个服务。

`RG_mode` 属性使群集管理员可以将资源组标识为故障切换或可伸缩。如果 `RG_mode` 的值为 `SCALABLE`，RGM 将允许 `Maximum_primaries` 的值大于 1，这意味着该组可以同时由多个节点主控。RGM 将允许其 `Failover` 属性的值为 `FALSE` 的资源在其 `RG_mode` 的值为 `SCALABLE` 的资源组中实例化。

群集管理员将创建一个用于包含可伸缩服务资源的可伸缩资源组，以及一个用于包含可伸缩资源所依赖的共享地址资源的独立故障转移资源组。

群集管理员使用 `RG_dependencies` 资源组属性来指定在节点上使资源组联机和脱机的顺序。此顺序对于可伸缩服务很重要，因为可伸缩资源和其依赖的共享地址资源位于不同的资源组中。可伸缩数据服务需要在启动前将其网络地址（共享地址）资源配置为启用。因此，群集管理员必须设置包含可伸缩服务的资源组的 `RG_dependencies` 属性，以包括含有共享地址资源的资源组。

当您在 RTR 文件中为资源声明 `Scalable` 属性时，RGM 将自动为该资源创建以下一组可伸缩属性。

`Network_resources_used`
: 标识该资源所使用的共享地址资源。此属性默认为空字符串，因此群集管理员必须在创建资源时提供可伸缩服务使用的共享地址的实际列表。`scsetup` 命令和 SunPlex Manager 为可伸缩服务提供了自动设置所需资源和组的功能。

`Load_balancing_policy`
: 为资源指定负载平衡策略。您可以在 RTR 文件中明确设置该策略，或允许使用默认值 `LB_WEIGHTED`。无论是以上哪种情况，群集管理员都可以在创建资源时更改该值（除非您在 RTR 文件中将 `Load_balancing_policy` 的 `Tunable` 设置为 `NONE` 或 `FALSE`）。以下是您可以使用的合法值：

  `LB_WEIGHTED`
  : 根据在 `Load_balancing_weights` 属性中设置的权重在不同的节点间分配负载。

  `LB_STICKY`
  : 可伸缩服务的给定客户机（由客户机的 IP 地址标识）总是发送到同一群集节点。

LB_STICKY_WILD
连接到通配符粘滞服务 IP 地址的给定客户机（通过客户机的 IP 地址标识）总是发送到同一个群集节点，无论指向哪个端口号。

对于使用 LB_STICKY 或 LB_STICKY_WILD 的 Load_balancing_policy 的可伸缩服务，在服务处于联机状态时更改 Load_balancing_weights，可能会导致复位现有的客户机关联。在此情况下，一个不同的节点可能会处理后续的客户机请求（即使该客户机之前已由群集中的另一个节点处理过）。

同样地，在群集中启动该服务的一个新实例可能也会复位现有的客户机关联。

Load_balancing_weights
用来指定要分配给每个节点的负载。其格式为 *weight@node,weight@node*，其中 *weight* 是一个反映分配到指定 *node* 的相对负载部分的整数。分配到某个节点的负载部分是此节点的权数除以所有活动实例权数的和。例如，1@1,3@2 表明节点 1 接收到 ? 的负载，而节点 2 接收到 ? 的负载。

Port_list
用来标识服务器所侦听的端口。此属性缺省为空字符串。您可以在 RTR 文件中提供端口列表。否则，群集管理员在创建资源时必须提供实际端口列表。

您可以将由群集管理员配置的数据服务创建为具有可伸缩性或故障转移。要进行此操作，请在数据服务的 RTR 文件中将 Failover 资源类型属性和 Scalable 资源属性都声明为 FALSE。创建时请将 Scalable 属性指定为可调。

Failover 属性值 FALSE 允许将资源配置到可伸缩资源组中。通过在创建资源时将 Scalable 的值更改为 TRUE，群集管理员将可以启用共享地址来创建可伸缩服务。

另一方面，即使将 Failover 设置为 FALSE，群集管理员也可以将资源配置到故障转移资源组中，以实现故障转移服务。群集管理员不会更改值为 FALSE 的 Scalable 的值。要支持此情况，您应该检查 Scalable 属性的 Validate 方法。如果 Scalable 的值为 FALSE，请检验资源是否配置到故障转移资源组中。

《Sun Cluster 概念指南（适用于 Solaris OS）》中包含有关可伸缩资源的其他信息。

可伸缩服务的验证检查

每当在将可伸缩属性值设置为 TRUE 的情况下创建资源或更新资源时，RGM 都将验证各种资源属性。如果属性值未正确配置，RGM 将拒绝更新或创建尝试。RGM 将执行以下方面的检查：

- Network_resources_used 属性必须为非空，并且包含现有共享地址资源的名称。包含可伸缩资源的资源组 Nodelist 中的每个节点都必须显示在每个已命名的共享地址资源的 NetIfList 属性或 AuxNodeList 属性中。
- 包含可伸缩资源的资源组的 RG_dependencies 属性必须包括在可伸缩资源的 Network_resources_used 属性中列出的所有共享地址资源的资源组。
- Port_list 属性必须为非空，并且包含端口协议对的列表。必须将斜杠 (/) 附加到每个端口号的后面，斜杠后紧跟该端口所使用的协议。例如：

```
Port_list=80/tcp6,40/udp6
```

您可以指定以下协议值：

- `tcp`，用于 TCP IPv4
- `tcp6`（适用于 TCP IPv6）
- `udp`，用于 UDP IPv4
- `udp6`（适用于 UDP IPv6）

编写和测试数据服务

使用 TCP 持续连接保护服务器

在服务器端使用 TCP 持续连接保护服务器，以避免为关闭的（或网络分区的）客户机浪费系统资源。如果在持续运行了足够长时间的服务器中没有清理这些资源，当客户机崩溃或重新引导时，会最终导致浪费的资源量无限地增长。

如果客户机-服务器通信使用 TCP 流，则客户机和服务器都应启用 TCP 持续连接机制。这也适用于非 HA 单服务器情况。

其他面向连接的协议可能也需要具有持续连接机制。

在客户机端，使用 TCP 持续连接机制，将使客户机在网络地址资源进行故障转移或从一个物理主机切换到另一个物理主机时收到通知。网络地址资源的这种传送将中断 TCP 连接。但是，除非该客户机已启用了持续连接机制，否则当连接中断而当时该连接又正处于静止状态时，该客户机不一定会收到关于此情况的通知。

例如，假设客户机正在等待服务器对长时间运行请求的反应，且该客户机的请求消息已发送到服务器并已在 TCP 层得到了确认。在此情况下，客户机的 TCP 模块无需不断重新传送请求。另外，客户机应用程序将处于阻塞状态，等待对该请求的响应。

在可能的情况下，客户机应用程序除了使用 TCP 持续连接机制以外，还必须在其级别上执行自己的定期持续连接。TCP 持续连接机制在处理所有可能的边界问题上并不都是完美的。通常情况下，使用应用程序级的持续连接机制要求客户机-服务器协议支持空操作，或至少支持有效的只读操作（例如状态操作）。

测试 HA 数据服务

本节提供了如何在 HA 环境中测试数据服务实现的建议。测试情况只是一些建议，并不很全面。您需要访问测试平台 Sun Cluster 的配置，以便测试工作不会对生产计算机造成影响。

测试 HA 数据服务在资源组在物理主机之间移动的所有情况下是否能正常运行。这些情况包括系统崩溃以及使用 `scswitch` 命令的情况。测试客户机在这些情况下是否可以继续获取服务。

测试方法的幂等性。例如，用可多次调用原始方法的简短 shell 脚本暂时替换所有方法。

协调资源间的依赖性

有时，某个客户机-服务器数据服务在满足某个客户机请求的同时，也会向其他的客户机-服务器数据服务发出请求。例如，如果当数据服务 A 提供其服务时，数据服务 B 也必须提供其服务，则说明数据服务 A 依赖于数据服务 B。为了满足此要求，Sun Cluster 允许在资源组内配置资源的依赖性。这些依赖性会影响 Sun Cluster 启动和停止数据服务的顺序。有关详细信息，请参见 `scrgadm`(1M) 手册页。

如果您的资源类型的资源依赖于其他类型的资源，则您需要指导群集管理员正确配置资源和资源组。或者，提供正确配置它们的脚本或工具。如果依赖资源必须和“被依赖”资源在同一节点上运行，则必须在同一资源组中配置这两种资源。

决定是使用显式资源依赖性，还是忽略它们，并用您 HA 数据服务的自身代码轮询其他数据服务的可用性。如果依赖资源和被依赖资源可以在不同节点上运行，则可以在不同的资源组中对它们进行配置。在此情况下，需要进行轮询，这是因为不能跨组配置资源依赖性。

某些数据服务本身不直接存储数据。而是依赖其他后端数据服务存储其所有的数据。这样的数据服务将所有只读和更新请求转送到对后端数据服务的调用中。例如，假设一个将其所有数据都保留在 SQL 数据库（例如 Oracle）中的客户机-服务器日程日历服务。日程日历服务具有自己的客户机-服务器网络协议。例如，它可能已使用 RPC 规范语言（例如 ONC RPC）定义了自己的协议。

在 Sun Cluster 环境中，您可以使用 HA-ORACLE 使后端 Oracle 数据库具有高可用性。然后，就可以编写用于启动和停止日程日历守护进程的简单方法。群集管理员使用 Sun Cluster 注册日程日历资源类型。

如果日程日历应用程序必须与 Oracle 数据库在同一节点上运行，则群集管理员必须在与 HA-ORACLE 资源所在的同一资源组中配置日程日历资源并使其依赖于 HA-ORACLE 资源。使用 `scrgadm` 中的 `Resource_dependencies` 属性标记指定该依赖性。

如果 HA-ORACLE 资源可以在与日程日历资源不同的节点上运行，则群集管理员将它们配置到两个不同的资源组中。群集管理员可能会在 Oracle 资源组中配置日历资源组的资源组依赖性。但是，资源组依赖性仅当两个资源组同时在同一节点上启动或停止时才有效。因此，日历数据服务守护进程在启动后可能进行轮询，以等待 Oracle 数据库成为可用的数据库。在此情况下，日历资源类型的 `Start` 方法通常返回成功消息。但是，如果 `Start` 方法无限期地处于阻塞状态，它将使其资源组处于繁忙状态。此繁忙状态将阻止对该组进行任何进一步的状态更改（例如编辑、故障转移或切换）。但是，如果日历资源的 `Start` 方法超时或以非零状态退出，则其超时或非零退出状态可能会导致该资源组在两个或多个节点之间切换，而 Oracle 数据库仍不可用。

第 3 章

资源管理 API 参考

本章列出并简要介绍了构成资源管理 API (RMAPI) 的访问函数和回调方法。但是，RMAPI 手册页才是有关这些函数和方法的权威参考。

本章包括以下主题：

- 第 51 页中的 “RMAPI 访问方法” – 以 shell 脚本命令和 C 函数的形式
- 第 55 页中的 “RMAPI 回调方法” – `rt_callbacks`(1HA) 手册页中进行介绍

RMAPI 访问方法

API 提供了用于访问资源类型、资源和资源组属性以及其他群集信息的函数。这些函数是以 shell 命令和 C 函数的形式提供的，使您可以实现 shell 脚本或 C 程序形式的控制程序。

RMAPI Shell 命令

Shell 命令是在资源类型的回调方法的 shell 脚本实现中使用的，这些资源类型表示由群集的 RGM 控制的服务。可以使用这些命令完成以下任务：

- 访问有关资源类型、资源、资源组和群集的信息。
- 使用监视器时，设置资源的 `Status` 和 `Status_msg` 属性。
- 请求重新启动或重定位资源组。

注 – 此部分仅简要介绍了 shell 命令，1HA 手册页中提供了 shell 命令的权威性参考。如果未另加说明，与命令同名的手册页即与相应命令相关。

RMAPI 资源命令

使用这些命令，您可以访问关于资源的信息或设置资源的 `Status` 和 `Status_msg` 属性。

`scha_resource_get`
访问有关 RGM 控制下的资源或资源类型的信息。此命令提供的信息与 `scha_resource_get()` C 函数相同。有关详细信息，请参见 `scha_resource_get`(1HA) 手册页。

`scha_resource_setstatus`
设置 RGM 控制下的资源的 `Status` 和 `Status_msg` 属性。资源的监视器使用此命令来指明监视器识别出的资源状态。此命令提供与 `scha_resource_setstatus()` C 函数相同的功能。`scha_resource_setstatus`(1HA) 手册页中对此命令进行了详细介绍。

注 – 虽然对于资源监视器来说 `scha_resource_setstatus()` 有特定用途，但是任何程序都可以调用该函数。

资源类型命令

`scha_resourcetype_get`
访问有关向 RGM 注册的资源类型的信息。此命令提供与 `scha_resourcetype_get()` C 函数相同的功能。`scha_resourcetype_get`(1HA) 手册页中对此命令进行了详细介绍。

资源组命令

您可以使用这些命令访问关于资源组的信息或重新启动资源组。

`scha_resourcegroup_get`
访问关于 RGM 控制下的资源组的信息。此命令提供与 `scha_resourcetype_get()` C 函数相同的功能。`scha_resourcegroup_get`(1HA) 手册页中对此命令进行了详细介绍。

`scha_control`
请求重新启动 RGM 控制下的资源组或将其重定位到其他节点。此命令提供与 `scha_control()` C 函数相同的功能。`scha_control`(1HA) 手册页中对此命令进行了详细介绍。

群集命令

`scha_cluster_get`
访问关于群集的信息，例如群集名称、节点名称、ID、状态和资源组。此命令提供的信息与 `scha_cluster_get()` C 函数相同。`scha_cluster_get`(1HA) 手册页中对此命令进行了详细介绍。

C 函数

C 函数是在资源类型的回调方法的 C 程序实现中使用的，这些资源类型表示由群集的 RGM 控制的服务。可以使用这些函数完成以下任务：

- 访问关于资源类型、资源、资源组和群集的信息。
- 使用监视器时，设置资源的 `Status` 和 `Status_msg` 属性。
- 请求重新启动或重定位资源组。
- 将错误代码转换为相关错误消息。

注 – 此部分仅简要介绍了 C 函数，3HA 手册页中提供了 C 函数的权威性参考。如果未另加说明，与命令同名的手册页即与相应命令相关。有关 C 函数的输出参数和返回码的信息，请参见 `scha_calls`(3HA) 手册页。

资源函数

这些函数用于访问有关 RGM 所管理的资源的信息，或指明监视器识别出的资源状态。

`scha_resource_open()`、`scha_resource_get()` 和 `scha_resource_close()`
这些函数用于访问关于由 RGM 管理的资源的信息。`scha_resource_open()` 函数将对资源的访问进行初始化，并为 `scha_resource_get()`（该函数用于访问资源信息）返回句柄。`scha_resource_close()` 函数将使该句柄无效，并释放为 `scha_resource_get()` 返回值分配的内存。

当 `scha_resource_open()` 返回资源的句柄之后，可以通过群集重新配置或者管理操作更改资源。因此，`scha_resource_get()` 通过该句柄获取的信息就可能不准确。在对资源进行群集重新配置或管理操作的情况下，RGM 将把 `scha_err_seqid` 错误代码返回给 `scha_resource_get()`，以指明关于资源的信息可能已更改。此错误消息不是致命的。函数将成功返回。您可以选择忽略此消息并接受返回的信息，也可以关闭当前句柄并打开新句柄以访问关于资源的信息。

有一个手册页介绍了这三个函数。您可以通过 `scha_resource_open`(3HA)、`scha_resource_get`(3HA) 或 `scha_resource_close`(3HA) 中的任何一个函数访问此手册页。

`scha_resource_setstatus()`
设置 RGM 控制下的资源的 `Status` 和 `Status_msg` 属性。资源的监视器使用此函数指示该资源的状态。

注 – 虽然对于资源监视器来说 `scha_resource_setstatus()` 有特定用途，但是任何程序都可以调用该函数。

资源类型函数

这些函数用于访问关于向 RGM 注册的资源类型的信息。

`scha_resourcetype_open()`、`scha_resourcetype_get()` 和 `scha_resourcetype_close()`

`scha_resourcetype_open()` 函数将对资源的访问进行初始化，并为 `scha_resourcetype_get()`（该函数用于访问资源类型信息）返回句柄。`scha_resourcetype_close()` 函数将使句柄无效，并释放为 `scha_resourcetype_get()` 返回值分配的内存。

当 `scha_resourcetype_open()` 返回资源类型的句柄之后，可以通过群集重新配置或管理操作更改资源类型。因此，`scha_resourcetype_get()` 通过该句柄获取的信息可能不准确。在对资源类型进行群集重新配置或管理操作的情况下，RGM 将把 `scha_err_seqid` 错误代码返回给 `scha_resourcetype_get()`，以指明关于资源类型的信息可能已更改。此错误消息不是致命的。函数将成功返回。您可以选择忽略此消息并接受返回的信息，也可以关闭当前句柄并打开新句柄以访问关于资源类型的信息。

有一个手册页介绍了这三个函数。您可以通过任一单个函数 `scha_resourcetype_open`(3HA)、`scha_resourcetype_get`(3HA) 或 `scha_resourcetype_close`(3HA) 访问此手册页。

资源组函数

您可以使用这些函数访问关于资源组的信息或重新启动资源组。

`scha_resourcegroup_open()`、`scha_resourcegroup_get()` 和 `scha_resourcegroup_close()`

这些函数用于访问关于由 RGM 管理的资源组的信息。`scha_resourcegroup_open()` 函数用来初始化对资源组的访问并返回 `scha_resourcegroup_get ()` 的句柄，用于访问资源组信息。`scha_resourcegroup_close()` 函数将使句柄无效，并释放为 `scha_resourcegroup_get()` 返回值分配的内存。

当 `scha_resourcegroup_open()` 返回资源组句柄之后，可以通过群集重新配置或管理操作更改资源组。因此，`scha_resourcegroup_get()` 通过该句柄获取的信息可能不准确。在对资源组进行群集重新配置或管理操作的情况下，RGM 将把 `scha_err_seqid` 错误代码返回给 `scha_resourcegroup_get()`，以指明关于资源组的信息可能已更改。此错误消息不是致命的。函数将成功返回。您可以选择忽略此消息并接受返回的信息，也可以关闭当前句柄并打开新句柄以访问关于资源组的信息。

有一个手册页介绍了这三个函数。您可以通过任一单个函数 `scha_resourcegroup_open`(3HA)、`scha_resourcegroup_get`(3HA) 和 `scha_resourcegroup_close`(3HA) 访问此手册页。

`scha_control()`

请求重新启动 RGM 控制下的资源组或将其重定位到其他节点。`scha_control`(3HA) 手册页中对此函数进行了详细介绍。

群集函数

这些函数用于访问或返回关于群集的信息。

`scha_cluster_open()`、`scha_cluster_get()` 和 `scha_cluster_close()`
这些函数用于访问关于群集的信息，例如群集名称、节点名称、ID、状态和资源组。

当 `scha_cluster_open()` 返回群集的句柄之后，可以通过重新配置或管理操作更改群集。因此，`scha_cluster_get()` 通过该句柄获取的信息可能不准确。在对群集进行重新配置或管理操作的情况下，RGM 将把 `scha_err_seqid` 错误代码返回给 `scha_cluster_get()`，以指明关于群集的信息可能已更改。此错误消息不是致命的。函数将成功返回。您可以选择忽略此消息并接受返回的信息，也可以关闭当前句柄并打开新句柄以访问关于群集的信息。

有一个手册页介绍了这三个函数。您可以通过任一单个函数 `scha_cluster_open`(3HA)、`scha_cluster_get`(3HA) 和 `scha_cluster_close`(3HA) 访问此手册页。

`scha_cluster_getlogfacility()`
返回用作群集日志的系统日志工具的编号。将返回值与 `syslog()` Solaris 函数结合使用可以将事件和状态消息记录到群集日志中。`scha_cluster_getlogfacility`(3HA) 手册页中对此函数进行了详细介绍。

`scha_cluster_getnodename()`
返回在其上调用该函数的群集节点的名称。`scha_cluster_getnodename`(3HA) 手册页中对此函数进行了详细介绍。

公用程序函数

此函数将把错误代码转换为错误消息。

`scha_strerror()`
将某个 `scha_` 函数返回的错误代码转换为相应的错误消息。将此函数与 `logger` 命令结合使用可以将消息记录到 Solaris 系统日志 (`syslog`) 中。`scha_strerror`(3HA) 手册页中对此函数进行了详细介绍。

RMAPI 回调方法

回调方法是为实现资源类型由 API 提供的主要元素。当群集成员资格发生更改（例如节点引导或崩溃）时，回调方法使 RGM 可以控制群集中的资源。

注 – 回调方法是由具有超级用户权限或具有等效角色权限的 RGM 执行的，因为客户机程序控制了群集系统中的 HA 服务。安装和管理这些具有限制性文件拥有权和权限的方法。具体来讲，就是要赋予这些方法一个特权拥有者（例如 `bin` 或 `root`），并使这些方法不可写。

此部分介绍了回调方法参数和退出代码。介绍了以下几种类别的回调方法：

- 控制和初始化方法
- 管理支持方法
- 与网络相关的方法
- 监视器控制方法

注 – 此部分简要介绍了回调方法，包括何时运行方法和对资源的预期影响。但 `rt_callbacks`(1HA) 手册页才是回调方法的权威性参考。

可以提供给回调方法的参数

RGM 将运行回调方法，如下所示：

method `-R` *resource-name* `-T` *type-name* `-G` *group-name*

其中 method 是注册为 `Start`、`Stop` 或其他回调方法的程序的路径名。资源类型的回调方法在其注册文件中进行声明。

所有回调方法参数均以带标志的值的形式进行传递，如下所示：

- `-R` 表示资源实例的名称
- `-T` 表示资源的类型
- `-G` 表示资源配置在其中的组

使用变量和访问函数来检索关于资源的信息。

`Validate` 方法与其他参数（包括对其调用该方法的资源和资源组的属性值）一起调用。

`scha_calls`(3HA) 手册页包含更多信息。

回调方法退出代码

所有回调方法具有相同退出代码。定义这些退出代码是为了指定方法调用对资源状态的影响。`scha_calls`(3HA) 手册页对这些退出代码进行了详细介绍。退出代码的两大类别为：

- `0` – 方法已成功
- 所有非零值 – 方法已失败

RGM 还可以处理在执行回调方法时出现的异常失败，例如超时和核心转储。

方法实现必须使用每个节点上的 `syslog()` 来输出失败信息。虽然写入到 `stdout` 或 `stderr` 的输出当前显示在本地节点的控制台上，但这并不保证会将该输出传送给用户。

控制和初始化回调方法

主要的控制和初始化回调方法用于启动和停止资源。其它方法用来执行对资源的初始化和终止代码。

`Start`
当包含资源的资源组在群集节点上联机后，RGM 将在该节点上运行此方法。此方法激活该节点上的资源。

在 `Start` 方法激活的资源启动并在本地节点上可用之前，不应退出 Start 方法。因此，在退出之前，`Start` 方法应轮询该资源，以确定该资源是否已启动。另外，应为此方法设置足够长的超时值。例如，诸如数据库守护进程之类的特定资源需要较长的时间才能启动，因而要求方法有较长的超时值。

RGM 对 `Start` 方法失败的响应方式取决于 `Failover_mode` 属性的设置。

资源类型注册 (RTR) 文件中的 `Start_timeout` 属性用于为资源的 `Start` 方法设置超时值。

`Stop`
当包含资源的资源组在群集节点上脱机后，RGM 将在该节点上运行此必需方法。如果该资源处于激活状态，则此方法可取消激活该资源。

在 `Stop` 方法控制的资源完全停止该资源在本地节点上的所有活动并关闭所有文件描述符之前，不应退出 Stop 方法。否则，由于 RGM 假定资源已停止，而实际上该资源仍处于活动状态，可能会导致损坏数据。避免损坏数据的最安全的方法是终止与资源相关的本地节点上的所有进程。

在退出之前，`Stop` 方法应该轮询该资源，以确定该资源是否停止。另外，应为此方法设置足够长的超时值。例如，诸如数据库守护进程之类的特定资源需要较长的时间才能停止，因而要求方法有较长的超时值。

RGM 响应 `Stop` 方法失败的方法取决于 `Failover_mode` 属性的设置。请参见第 220 页中的 “资源属性” 。

RTR 文件中的 `Stop_timeout` 属性用于为资源的 `Stop` 方法设置超时值。

`Init`
当资源成为受管理的资源后，RGM 将运行此可选方法来执行对资源的一次性初始化。当 RGM 的资源组从不受管理状态切换为受管理状态时，或者当在受管理资源组中创建资源时，RGM 将运行此方法。对 `Init_nodes` 资源属性所标识的节点调用此方法。

`Fini`
当资源成为不受管理的资源之后，RGM 将运行此可选方法以清理该资源。当 RGM 的资源组切换为不受管理的状态时，或从受管理的资源组中删除资源时，RGM 将运行此方法。对 `Init_nodes` 资源属性所标识的节点调用此方法。

`Boot`
当包含资源的资源组已经受 RGM 管理之后，RGM 将运行此可选方法（类似于 `Init`）以初始化加入了群集的节点上的资源。RGM 将在由 `Init_nodes` 资源属性标识的节点上运行此方法。由于引导或重新引导节点而使节点加入或重新加入群集

时，将调用 Boot 方法。

注 – Init、Fini 或 Boot 方法的失败将导致 syslog() 函数生成错误消息。但是，RGM 对资源的管理不受其他方面的影响。

管理支持方法

对资源进行的管理操作包括设置和更改资源属性。Validate 和 Update 回调方法使资源类型实现可以执行这些管理操作。

Validate
: 创建资源时，以及群集管理员更新资源或其包含资源组的属性时，RGM 都将调用此可选方法。对资源类型的 Init_nodes 属性所标识的群集节点集调用此方法。Validate 是在应用创建或更新之前调用的。来自任何节点上的方法的失败退出代码都会导致取消创建或更新操作。

  仅当群集管理员更改资源或资源组属性时才会调用 Validate，而在 RGM 设置属性时，或者监视器设置 Status 和 Status_msg 资源属性时不调用该方法。

Update
: RGM 将运行此可选方法以通知正在运行的资源，通知将说明属性已被更改。设置资源或资源组的属性的管理操作成功后，RGM 将运行 Update。对资源处于联机状态的节点调用此方法。方法将使用 API 访问函数以读取可能影响活动资源的属性值并相应调整正在运行的资源。

  注 – Update 方法的失败将导致 syslog() 函数生成错误消息。但是，RGM 对资源的管理不受其他方面的影响。

与网络相关的回调方法

使用网络地址资源的服务可能需要以与网络地址配置相关的特定顺序执行启动或停止步骤。以下可选回调方法 Prenet_start 和 Postnet_stop 用于启用资源类型实现以在配置或取消配置相关网络地址之前和之后执行启动和关闭操作。

Prenet_start
: 在配置同一资源组中的网络地址之前，调用此可选方法以执行特殊的启动操作。

Postnet_stop
: 将同一资源组中的网络地址配置为关闭之后，调用此可选方法以执行特殊的关闭操作。

监视器控制回调方法

资源类型实现可以可选包括用于监视资源性能、报告资源的状态或在资源出现故障时采取措施的程序。`Monitor_start`、`Monitor_stop` 和 `Monitor_check` 方法支持在资源类型实现中实现资源监视器。

`Monitor_start`
: 资源启动后，将调用此可选方法来启动该资源的监视器。

`Monitor_stop`
: 在停止资源前，调用此可选方法停止资源的监视器。

`Monitor_check`
: 在将资源组重定向到某个节点之前，将调用此可选方法以评定该节点的可靠性。必须实现 `Monitor_check` 方法，这样该方法才不会与同时运行的其他方法发生冲突。

第 4 章

修改资源类型

本章讨论修改资源类型需要了解的问题。群集管理员用于升级资源的方法信息亦包括在其中。

本章包括以下主题：

有关修改资源类型的概述

群集管理员必须能够执行以下任务：

- 安装和注册现有资源类型的新版本
- 允许注册给定资源类型的多个版本
- 将现有资源升级到新版本资源类型，而不必删除和重新创建该资源

要升级的资源类型称为**支持升级**的资源类型。可以更改现有资源类型的以下元素：

- 资源类型属性的属性
- 声明的资源属性（包括标准属性和扩展属性）集
- 资源属性的属性，例如 `default`、`min`、`max`、`arraymin`、`arraymax` 或 `tunability`
- 声明的方法集
- 方法或监视器的实现

注 – 修改应用程序代码时，不是一定要修改资源类型。

为群集管理员提供升级资源类型的工具时，需要了解有哪些要求。本章介绍了设置这些工具需要了解的知识。

设置资源类型注册文件的内容

资源类型名称

资源类型名称的三个部分分别是在 RTR 文件中指定为 *vendor-id*、*resource-type* 和 *rt-version* 的属性。`scrgadm` 命令插入句号和冒号分界符来创建资源类型的名称：

vendor-id.*resource-type*:*rt-version*

vendor-id 前缀用来区别不同公司提供的具有相同名称的两个注册文件。为确保 *vendor-id* 是唯一的，请在创建资源类型时使用公司的股票代号。*rt-version* 用于区别相同基本资源类型的多个注册版本（升级）。

可以通过键入以下命令来获取全限定资源类型名称：

```
# scha_resource_get -O Type -R resource-name -G resource-group-name
```

使用 Sun Cluster 3.1 以前的版本注册的资源类型名称则继续使用以下语法：

vendor-id.*resource-type*

第 294 页中的 “资源类型名称的格式”中介绍了资源类型名称的格式。

指定 #$upgrade 和 #$upgrade_from 指令

要确保您正在修改的资源类型是支持升级的，请在该资源类型的 RTR 文件中加入 `#$upgrade` 指令。在 `#$upgrade` 指令后面，为要支持的资源类型的每个早期版本添加零个或多于零个 `#$upgrade_from` 指令。

`#$upgrade` 和 `#$upgrade_from` 指令必须位于 RTR 文件中的资源类型属性声明和资源声明部分之间。请参见 `rt_reg`(4) 手册页。

示例 4–1 RTR 文件中的 `#$upgrade_from` 指令

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$upgrade_from   "1.2"   WHEN_OFFLINE
#$upgrade_from   "1.3"   WHEN_OFFLINE
```

示例 4–1 RTR 文件中的 #$upgrade_from 指令　（续）

```
#$upgrade_from   "2.0"   WHEN_UNMONITORED
#$upgrade_from   "2.1"   ANYTIME
#$upgrade_from   ""      WHEN_UNMANAGED
```

#$upgrade_from 指令的格式如下所示：

```
#$upgrade_from version  tunability
```

version
RT_version。如果某个资源类型不具有版本，或具有的版本不是您以前在 RTR 文件中定义的版本，请指定空字符串 (“”)。

tunability
群集管理员可以升级指定的 RT_version 的条件和时间。

在 #$upgrade_from 指令中，请使用以下可调性值：

ANYTIME
在对群集管理员可以升级资源的时间没有限制时使用。在升级过程中，资源可以完全处于联机状态。

WHEN_UNMONITORED
在新资源类型版本的方法满足以下条件时使用：

- Update、Stop、Monitor_check 和 Postnet_stop 方法兼容以前的资源类型版本的启动方法（Prenet_stop 和 Start）
- Fini 方法兼容以前版本的 Init 方法

群集管理员必须在升级之前仅停止资源监视器程序。

WHEN_OFFLINE
当新资源类型版本的 Update、Stop、Monitor_check 或 Postnet_stop 方法满足以下条件时使用：

- 兼容以前版本的 Init 方法
- 不兼容以前资源类型版本的启动方法（Prenet_stop 和 Start）

群集管理员必须在升级之前使资源处于脱机状态。

WHEN_DISABLED
与 WHEN_OFFLINE 相似。但是，群集管理员必须在升级之前禁用资源。

WHEN_UNMANAGED
当新资源类型版本的 Fini 方法不兼容以前版本的 Init 方法时使用。升级之前，群集管理员必须将现有资源组切换到未管理状态。

如果 #$upgrade_from 指令的列表中没有显示资源类型的某个版本，则默认情况下，RGM 会将 WHEN_UNMANAGED 可调性强加于该版本。

AT_CREATION
用于防止将现有资源升级到新版本资源类型。群集管理员必须删除和重新创建资源。

更改 RTR 文件中的 RT_version

更改 RTR 文件中的内容时，需要更改的只是 RT_version 属性。为此属性选择值，该值可以清楚地表明此版本的资源类型是最新版本。

请**勿**在 RTR 文件中的 RT_version 字符串中使用以下字符（否则资源类型的注册将失败）：

- 空格键
- Tab
- 斜杠 (/)
- 反斜杠 (\)
- 星号 (*)
- 问号 (?)
- 逗号 (,)
- 分号 (;)
- 左方括号 ([)
- 右方括号 (])

从 Sun Cluster 3.1 开始，RT_version 属性已成为强制属性（在 Sun Cluster 3.0 中为可选属性）。

Sun Cluster 早期版本中的资源类型名称

Sun Cluster 3.0 中的资源类型名称不包含版本后缀，如下所示：

vendor-id . resource-type

即使在群集管理员将群集软件升级到 Sun Cluster 3.1 或更高版本之后，原来在 Sun Cluster 3.0 下注册的资源类型仍将继续具有遵循此语法的名称。与此相似，如果 RTR 文件是在运行 Sun Cluster 3.1 或更高版本的群集上注册的，则该文件中缺少 #$upgrade 指令的资源类型将被指定 Sun Cluster 3.0 格式的名称（不具有版本后缀）。

群集管理员可以在 Sun Cluster 3.0 中使用 #$upgrade 指令或 #$upgrade_from 指令来注册 RTR 文件。但是，Sun Cluster 3.0 不支持将现有资源升级到新资源类型。

群集管理员进行升级时将出现的情况

以下是群集管理员升级资源类型时必须执行的操作或者将出现的情况：

- 如果现有资源属性不满足新版本资源类型的验证条件，则群集管理员必须提供有效值。群集管理员必须在出现以下情况时这样做：
  - 新版本资源类型没有默认值并且使用了早期版本中未声明的属性。

- 现有资源所使用的属性值在新版本中未声明或无效。将从资源中删除未在新版本的资源类型中声明的已声明属性。

- 从不支持的资源类型版本进行的任何升级尝试都将失败。
- 升级之后，资源将从新版本资源类型继承所有属性的资源属性。
- 如果您在 RTR 文件中更改了资源类型的默认值，则现有资源将继承新的默认值。即使将属性声明为仅在 `AT_CREATION` 或 `WHEN_DISABLED` 时可调，仍会继承新的默认值。群集管理员创建的相同类型的属性也会继承此默认值。但是，如果群集管理员为属性指定新的默认值，则新默认值将覆盖 RTR 文件中指定的默认值。

注 – 将在 Sun Cluster 3.0 中创建的资源升级到 Sun Cluster 的更高版本时，这些资源不从资源类型继承新的默认资源属性。此限制仅适用于从 Sun Cluster 3.0 群集升级的 Sun Cluster 3.1 群集。群集管理员可以通过为属性指定值并覆盖默认值来克服此限制。

实现资源类型监视器代码

群集管理员可以在 Sun Cluster 3.0 中注册支持升级的资源类型。但是，Sun Cluster 将记录无版本后缀的资源类型名称。为了可以在 Sun Cluster 3.0 和 Sun Cluster 3.1 中正常运行，此资源类型的监视器代码必须能够处理这两种命名约定：

vendor-id`.`*resource-type*`:`*rt-version*
vendor-id`.`*resource-type*

第 294 页中的 “资源类型名称的格式”中介绍了资源类型名称的格式。

群集管理员不能以两个不同的名称注册相同的资源类型版本两次。为了使监视器代码可以确定正确的名称，请在监视器代码中调用以下命令：

```
scha_resourcetype_get -O RT_VERSION -T VEND.myrt
scha_resourcetype_get -O RT_VERSION -T VEND.myrt:vers
```

然后，将输出值与 `vers` 进行比较。对于特定的 `vers` 值，这两个命令中只有一个会成功。

确定安装要求和封包

确定资源类型软件包的安装要求和封包时，请记住以下两点：

- 注册新资源类型时，必须可以在磁盘上访问其 RTR 文件。
- 创建新类型的资源时，新类型的所有声明的方法路径名称和监视器程序必须在磁盘上并可执行。只要正在使用该资源，就必须在原来位置保留旧的方法和监视器程序。

要确定将使用的正确封包，请考虑以下问题：

- 是否更改 RTR 文件?
- 是否更改属性的缺省值或可调性?
- 是否更改属性的 `min` 或 `max` 值?
- 升级过程中是否添加或删除属性?
- 是否更改监视器代码?
- 是否更改方法代码?
- 新方法、监视器代码或两者是否兼容以前的版本?

这些问题的答案将有助于您确定要对新资源类型使用的正确封包。

更改 RTR 文件须知

修改资源类型时，您不是一定要创建新方法或监视器代码。例如，您可能仅更改资源属性的默认值或可调性。在这种情况下，由于您没有更改方法代码，因而只需要可读取 RTR 文件的新有效路径名称。

如果不再需要注册旧资源类型，则可以使新 RTR 文件覆盖以前的版本。否则，将 RTR 文件置于新路径中。

如果升级更改了属性的默认值或可调性，则使用新版本资源类型的 `Validate` 方法，来验证现有属性对于新资源类型是否有效。如果无效，则群集管理员可以将现有资源的属性更改为正确值。如果升级更改了 `min`、`max` 或 `type` 属性，则 `scrgadm` 命令将在群集管理员升级资源类型时自动验证这些约束。

如果升级添加了新属性或删除了旧属性，则可能需要更改回调方法或监视器代码。

更改监视器代码

如果仅更改资源类型的监视器代码，则可以使用软件包安装程序覆盖监视器二进制文件。

更改方法代码

如果仅更改资源类型中的方法代码，则必须确定新方法代码是否兼容以前的方法代码。此问题的答案将确定新方法代码是必须存储在新路径中还是覆写以前的方法。

如果您可以将新 `Stop`、`Postnet_stop` 和 `Fini` 方法（如果已声明）应用于由以前版本的 `Start`、`Prenet_stop` 或 `Init` 方法初始化或启动的资源，则可以使新方法覆写以前的方法。

如果将新默认值应用于属性将导致方法（例如 `Stop`、`Postnet_stop` 或 `Fini`）出现故障，则在升级资源类型时，群集管理员必须相应地限制资源的状态。

当通过限制 Type_version 属性的可调性来升级资源时，应让群集管理员限制资源的状态。

封包的一种方法是包括软件包中仍支持的资源类型的所有早期版本。此方法允许新版本的软件包替换旧版本的软件包，而无需覆写或删除方法的以前路径。您必须确定要支持的以前版本的数量。

确定要使用的封包方案

下表总结了要用于新资源类型的封包方案。

表 4–1 确定要使用的封包方案

更改类型	可调性值	封包方案
仅在 RTR 文件中进行属性更改。	ANYTIME	仅传送新 RTR 文件。
更新方法。	ANYTIME	将已更新的方法置于与以前方法不同的路径中。
安装新监视器程序。	WHEN_UNMONITORED	仅覆写以前版本的监视器。
更新方法。 新 Update 和 Stop 方法不兼容以前的 Start 方法。	WHEN_OFFLINE	将已更新的方法置于与以前方法不同的路径中。
更新方法并将新属性添加到 RTR 文件中。新方法需要新属性。 目标是允许所包含的资源组仍处于联机状态，但如果该资源组在某个节点上从脱机状态转变为联机状态，则防止该资源进入联机状态。	WHEN_DISABLED	覆写方法的以前版本。
更新方法并将新属性添加到 RTR 文件中。新方法不需要新属性。	ANYTIME	覆写方法的以前版本。
更新方法。新的 Fini 方法与旧的 Init 方法不兼容。	WHEN_UNMANAGED	将已更新的方法置于与以前方法不同的路径中。
更新方法。不修改 RTR 文件。	不适用。不修改 RTR 文件。	覆写方法的以前版本。因为您未对 RTR 文件进行更改，所以无需注册或升级资源。

针对已修改的资源类型提供的文档信息

《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》中的"Upgrading a Resource Type"为群集管理员提供了可以用于如何升级资源类型的说明。为使群集管理员可以升级您修改的资源类型，本节中介绍了其他信息来作为以上说明的补充。

通常，创建新资源类型时，您需要提供具有以下用途的文档信息：

- 说明要添加、更改或删除的属性
- 说明如何使属性符合新要求
- 说明对资源的可调性约束
- 调用所有新默认属性
- 必要时通知群集管理员可以将现有资源属性设置为它们的正确值。

有关安装升级软件包之前应做事情的信息

向群集管理员说明在节点上安装升级软件包之前必须执行的操作，具体如下：

- 如果升级软件包将覆写现有方法，则指示群集管理员在非群集模式下重新引导节点。
- 如果升级软件包仅更新监视器代码而不更改方法代码，则告知群集管理员使节点保持在群集模式下运行。还应告知群集管理员关闭所有资源类型的监视。
- 如果升级软件包仅更新 RTR 文件而不更改方法和监视器代码，则告知群集管理员使节点保持在群集模式下运行。还应告知群集管理员使针对所有资源类型的监视保持在打开状态。

有关何时升级资源的信息

向群集管理员说明何时可以将资源升级到新版本的资源类型。群集管理员根据 RTR 文件中每个版本资源的 `#$upgrade_from` 指令的可调性来升级资源类型，如下所示：

- 随时 (`ANYTIME`)
- 仅在资源未受监视时 (`WHEN_UNMONITORED`)
- 仅在资源处于脱机状态时 (`WHEN_OFFLINE`)
- 仅在禁用资源时 (`WHEN_DISABLED`)
- 仅在资源组未受管理时 (`WHEN_UNMANAGED`)

示例 4–2 `#$upgrade_from` 定义群集管理员何时可以进行升级的方式

此示例显示了 `#$upgrade_from` 指令的可调性是如何影响群集管理员可以将资源升级到新版本资源类型的条件的。

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$upgrade_from   "1.2"   WHEN_OFFLINE
#$upgrade_from   "1.3"   WHEN_OFFLINE
```

示例 4–2 `#$upgrade_from` 定义群集管理员何时可以进行升级的方式 （续）

```
#$upgrade_from   "2.0"   WHEN_UNMONITORED
#$upgrade_from   "2.1"   ANYTIME
#$upgrade_from   ""      WHEN_UNMANAGED
```

版本	群集管理员何时可以升级资源
1.1、1.2 或 1.3	仅在资源处于脱机状态时
2.0	仅在资源未受监视时
2.1	任何时候
所有其他版本	仅在资源组未受管理时

有关对资源属性所作更改的信息

当您对资源类型进行某些更改时，要求群集管理员在升级时修改现有资源的属性。本部分说明了这样的更改。您可以进行的这样的更改概如下示：

- 更改现有资源类型属性的默认设置
- 引入资源类型的新扩展属性
- 删除资源类型的现有属性
- 更改您为资源类型声明的标准属性集
- 更改诸如 `min`、`max`、`arraymin`、`arraymax`、`default` 和 `tunability` 之类的资源属性的属性
- 更改您已声明的方法集
- 更改方法或默认监视器的实现

第 5 章

数据服务样例

本章介绍 `in.named` 应用程序的一个 Sun Cluster 数据服务样例 HA-DNS。`in.named` 守护进程是域名服务 (DNS) 的 Solaris 实现。数据服务样例说明了如何使用资源管理 API 使应用程序具有高可用性。

资源管理 API 支持 shell 脚本接口和 C 程序接口。本章中的应用程序样例是使用 shell 脚本接口编写的。

本章包括以下主题：

数据服务样例概述

数据服务样例在群集节点之间启动、停止、重新启动和切换 DNS 应用程序以便对群集事件（例如管理操作、应用程序故障或节点故障）做出响应。

应用程序的重启操作由进程监视工具 (PMF) 管理。如果失败的应用程序数超过了故障次数窗口中的故障计数，故障监视器会将包含应用程序资源的资源组故障转移到另一个节点上。

数据服务样例以 `PROBE` 方法的形式提供故障监视，该方法使用 `nslookup` 命令来确保应用程序正常运行。如果探测检测到挂起的 DNS 服务，它将通过在本地重新启动 DNS 应用程序来尝试改正这种情形。如果本地重新启动 DNS 应用程序没有改进该情况，并且探测重复检测到服务所出现的问题，则探测将尝试把服务故障转移到群集中的另一个节点上。

特别指出，数据服务样例包括以下元素：

- 用于定义数据服务的静态属性的资源类型注册文件。
- 通过 RGM 运行的 `Start` 回调方法，用来在包含 HA-DAS 数据服务的资源组联机时启动 `in.named` 守护进程。
- 通过 RGM 运行的 `Stop` 回调方法，用来在包含 HA-DNS 的资源组脱机时停止 `in.named` 守护进程。
- 用于通过检验 DNS 服务器是否正在运行来检查服务的可用性的故障监视器。故障监视器通过用户定义的 `PROBE` 方法实现，并通过 `Monitor_start` 和 `Monitor_stop` 回调方法启动和停止。
- 通过 RGM 运行的 `Validate` 回调方法，用于验证服务的配置目录是否可访问。
- 通过 RGM 运行的 `Update` 回调方法，用来在群集管理员更改资源属性值时重新启动故障监视器。

定义资源类型注册文件

本实例中的资源类型注册 (RTR) 文件中定义了 DNS 资源类型的静态配置。该类型的资源继承 RTR 文件中定义的属性。

资源组管理器 (RGM) 在群集管理员注册 HA-DNS 数据服务时读取 RTR 文件中的信息。

RTR 文件概述

RTR 文件采用定义好的格式。在文件中将依次定义资源类型属性、系统定义的资源属性和扩展属性。有关更多信息，请参见 `rt_reg`(4) 手册页和第 31 页中的 “设置资源和资源类型属性”。

以下各节介绍了样例 RTR 文件中的特定属性。其中列出了该文件的不同部分。有关样例 RTR 文件内容的完整列表，请参见第 243 页中的 “资源类型注册文件列表”。

RTR 文件样例中的资源类型属性

RTR 文件样例的开头部分是注释，其后跟有用来定义 HA-DNS 配置的资源类型属性，如下所示。

注 – 资源组、资源和资源类型的属性名称**不**区分大小写。在指定属性名称时，您可以使用大小写字母的任意组合。

```
#
# Copyright (c) 1998-2005 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#

#pragma ident   "@(#)SUNW.sample   1.1   00/05/24 SMI"

Resource_type = "sample";
Vendor_id = SUNW;
RT_description = "Domain Name Service on Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;

RT_basedir=/opt/SUNWsample/bin;
Pkglist = SUNWsample;

Start         =   dns_svc_start;
Stop          =   dns_svc_stop;

Validate      =   dns_validate;
Update        =   dns_update;

Monitor_start =   dns_monitor_start;
Monitor_stop  =   dns_monitor_stop;
Monitor_check =   dns_monitor_check;
```

提示 – 您必须将 `Resource_type` 属性声明为 RTR 文件中的第一项。否则，资源类型注册将失败。

以下信息介绍了这些属性：

- 可以只通过 `Resource_type` 属性来指定资源类型名称 (`sample`)，或通过使用 *vendor-id* 作为前缀，后跟一个句点 (.)，后面加上资源类型属性来指定资源类型名称 (`SUNW.sample`)。

  指定 *vendor-id* 时，使用用于定义资源类型的公司证券交易符号。在群集中资源类型的名称必须唯一。
- `RT_version` 属性用于将数据服务样例的版本标识为供应商指定的版本。
- `API_version` 属性用于标识 Sun Cluster 的版本。例如，`API_version = 2` 表明数据服务可以在从 Sun Cluster 3.0 开始的 Sun Cluster 的所有版本上运行；`API_version = 5` 表明数据服务可以安装在从 Sun Cluster 3.1 9/04 开始的 Sun

Cluster 的所有版本上。但是，API_version = 5 还表明数据服务无法安装在 Sun Cluster 3.1 9/04 发行之前的 Sun Cluster 的所有版本上。有关该属性的详细信息，请参见第 213 页中的 “资源类型属性”中有关 API_version 条目下的内容。

- Failover = TRUE 表示数据服务无法在可以在多个节点上同时联机的资源组中运行。
- RT_basedir 指向 /opt/SUNWsample/bin，将其作为完善相关路径（例如回调方法路径）的目录路径。
- Start、Stop 和 Validate 提供了通过 RGM 运行的回调方法程序的相应路径。这些路径是 RT_basedir 所指定的目录的相对路径。
- Pkglist 将 SUNWsample 标识为包含数据服务样例安装的软件包。

在此 RTR 文件中没有指定的资源类型属性（例如 Single_instance、Init_nodes 和 Installed_nodes 被设置为其默认值。第 213 页中的 “资源类型属性”包含资源类型属性（包括其默认值）的完整列表。

群集管理员不能更改 RTR 文件中资源类型属性的值。

RTR 文件样例中的资源属性

根据约定，在 RTR 文件中的资源类型属性之后声明资源属性。资源属性包括由 Sun Cluster 软件提供的系统定义的属性以及您定义的扩展属性。您可以为每一种类型指定许多属性的属性，它们由 Sun Cluster 软件提供，例如最小值、最大值和默认值。

RTR 文件中的系统定义的属性

以下列表显示了某个样例 RTR 文件中系统定义的属性。

```
# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.

# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.

# The MIN value for all method timeouts is set to 60 seconds. This
# prevents administrators from setting shorter timeouts, which do not
# improve switchover/failover performance, and can lead to undesired
# RGM actions (false failovers, node reboot, or moving the resource group
# to ERROR_STOP_FAILED state, requiring operator intervention). Setting
# too-short method timeouts leads to a *decrease* in overall availability
# of the data service.
{
   PROPERTY = Start_timeout;
   MIN=60;
   DEFAULT=300;
}
```

```
{
   PROPERTY = Stop_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Validate_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Update_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Monitor_Start_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Monitor_Stop_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Thorough_Probe_Interval;
   MIN=1;
   MAX=3600;
   DEFAULT=60;
   TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
   PROPERTY = Retry_count;
   MIN=0;
   MAX=10;
   DEFAULT=2;
   TUNABLE = ANYTIME;
}

# Set Retry_interval as a multiple of 60 since it is converted from seconds
# to minutes, rounding up. For example, a value of 50 (seconds)
# is converted to 1 minute. Use this property to time the number of
# retries (Retry_count).
{
   PROPERTY = Retry_interval;
   MIN=60;
   MAX=3600;
   DEFAULT=300;
   TUNABLE = ANYTIME;
}
```

```
{
   PROPERTY = Network_resources_used;
   TUNABLE = AT_CREATION;
   DEFAULT = "";
}
```

尽管 Sun Cluster 软件提供了系统定义的属性，您也可以使用资源属性的属性设置不同的默认值。有关您可用来应用到资源属性中的属性的完整列表，请参见第 240 页中的 “资源特性属性”。

有关样例 RTR 文件中系统定义的资源属性，请注意以下几点：

- 对于所有超时，Sun Cluster 给出了最小值（1 秒钟）和默认值（3600 秒，或 1 小时）。样例 RTR 文件将最小超时值更改为 60 秒，将默认值更改为 300 秒。群集管理员可以接受此默认值，或将超时值更改为其他值，60 秒或更大值。Sun Cluster 没有最大值上限。
- 属性 `Thorough_probe_interval`、`Retry_count` 和 `Retry_interval` 的 `TUNABLE` 属性被设置为 `ANYTIME`。这些设置表示群集管理员可以更改这些属性的值，即使数据服务正在运行。这些属性由通过数据服务样例实现的故障监视器使用。数据服务样例将实现 `Update` 方法，以在通过管理操作更改这些或其它资源属性后停止和重启故障监视器。请参见第 98 页中的 “Update 方法的工作方式”。
- 资源属性按以下分类：
  - **必需**。创建资源时，群集管理员必须指定一个值。
  - **可选**。如果群集管理员没有指定值，系统将提供默认值。
  - **条件**。只有在 RTR 文件中声明后，RGM 才能创建属性。

  数据服务样例的故障监视器使用 `Thorough_probe_interval`、`Retry_count`、`Retry_interval` 和 `Network_resources_used` 条件属性，因此需要在 RTR 文件中对它们进行声明。有关如何给属性分类的信息，请参见 `r_properties`(5) 手册页或第 220 页中的 “资源属性”。

RTR 文件中的扩展属性

样例 RTR 文件的结尾处是扩展属性，如该列表所示。

```
# Extension Properties

# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, DNS, specify the path of the DNS configuration file on
# PXFS (typically named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "The Configuration Directory Path";
}
```

```
# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 120;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

样例 RTR 文件定义两种扩展属性，`Confdir` 和 `Probe_timeout`。`Confdir` 属性指定 DNS 配置目录的路径。该目录包含 DNS 正常操作所需的 `in.named` 文件。数据服务样例的 `Start` 和 `Validate` 方法使用该属性验证启动 DNS 之前配置目录和 `in.named` 文件是否可访问。

配置数据服务之后，`Validate` 方法将检验新目录是否可以访问。

数据服务样例的 `PROBE` 方法不是 Sun Cluster 回调方法，而是用户定义的方法。因此，Sun Cluster 没有为其提供 `Probe_timeout` 属性。要使群集管理员可以配置 `Probe_timeout` 值，需要在 RTR 文件中定义扩展属性。

为所有方法提供通用功能

本节介绍以下用于数据服务样例中所有回调方法的功能：

- 第 77 页中的 “标识命令解释程序并输出路径”
- 第 78 页中的 “声明 `PMF_TAG` 和 `SYSLOG_TAG` 变量”
- 第 78 页中的 “分析函数参数”
- 第 80 页中的 “生成错误消息”
- 第 80 页中的 “获取属性信息”

标识命令解释程序并输出路径

shell 脚本中的第一行必须用来标识命令解释程序。数据服务样例中的每个方法脚本都可以标识命令解释程序，如下所示：

```
#!/bin/ksh
```

应用程序样例中的所有方法脚本都将导出 Sun Cluster 二进制数据和库的路径，而不依赖用户的 `PATH` 设置。

```
##############################################################################
# MAIN
##############################################################################
```

```
export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH
```

声明 PMF_TAG 和 SYSLOG_TAG 变量

所有方法脚本（除了 Validate）都使用 pmfadm来启动或停止数据服务或监视器，并传递资源名称。每个脚本都定义一个 PMF_TAG 变量，该变量可以传递至 pmfadm 以标识数据服务或监视器。

同样地，每个方法脚本都使用 logger 命令将消息记录到系统日志中。每个脚本都定义一个 SYSLOG_TAG 变量，该变量可以和 -t 选项一起传递至 logger，以标识正在为其记录消息的资源的资源类型、资源名称和资源组。

所有方法都以同一种方式定义 SYSLOG_TAG，如以下样例代码所示。dns_probe、dns_svc_start、dns_svc_stop 和 dns_monitor_check 方法按如下所示定义 PMF_TAG（pmfadm 和 logger 的用法来自 dns_svc_stop 方法）。

```
#############################################################################
# MAIN
#############################################################################

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info \
          -t [$SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"
```

dns_monitor_start、dns_monitor_stop 和 dns_update 方法按以下方式定义 PMF_TAG（pmfadm 的用法来自 dns_monitor_stop 方法）：

```
#############################################################################
# MAIN
#############################################################################

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
...
# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

分析函数参数

RGM 运行所有的回调方法（除了 Validate），如下所示：

method-name -R *resource-name* -T *resource-type-name* -G *resource-group-name*

方法名是实现回调方法的程序的路径名。数据服务指定 RTR 文件中各个方法的路径名。这些路径名与 RT_basedir 属性（也在 RTR 文件中）指定的目录有关。例如，在数据服务样例的 RTR 文件中，基目录和方法名按以下方式指定：

```
RT_basedir=/opt/SUNWsample/bin;
Start = dns_svc_start;
Stop =  dns_svc_stop;
...
```

所有回调方法参数都将作为带标志的值传递。参数 -R 表示资源实例的名称。参数 -T 表示资源的类型。参数 -G 表示资源配置到的组。有关回调方法的更多信息，请参见 rt_callbacks(1HA) 手册页。

注 – Validate 方法与其他参数（即，在其上调用该方法的资源和资源组的属性值）一起调用。有关更多信息，请参见第 94 页中的 “处理属性更新”。

每个回调方法都需要一个函数来解析传递该函数的参数。因为所有回调方法都传送相同的参数，所以数据服务提供了一个用于应用程序中所有回调方法的分析函数。

以下样例显示了样例应用程序中的回调方法所使用的 parse_args() 函数。

```
#############################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
      typeset opt

      while getopts 'R:G:T:' opt
      do
             case "$opt" in
             R)
                  # Name of the DNS resource.
                  RESOURCE_NAME=$OPTARG
                  ;;
             G)
                  # Name of the resource group in which the resource is
                  # configured.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
             T)
                  # Name of the resource type.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
             *)
                  logger -p ${SYSLOG_FACILITY}.err \
                  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                  "ERROR: Option $OPTARG unknown"
                  exit 1
                      ;;
             esac
```

```
    done
}
```

注 – 虽然应用程序样例中的 PROBE 方法是用户定义的（不是 Sun Cluster 回调方法），但是也使用与调用回调方法相同的变量来调用该方法。因此，该方法包含一个与另一个回调方法使用的解析函数相同的解析函数。

分析函数在 MAIN 中调用，如下所示：

```
parse_args "$@"
```

生成错误消息

回调方法应使用 syslog() 函数将错误消息输出到最终用户。数据服务样例中的所有回调方法都使用 scha_cluster_get 命令检索用于群集日志的 syslog() 函数的数目，如下所示：

```
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
```

该值存储在 shell 变量 SYSLOG_FACILITY 中，可以用作 logger 命令在群集日志中记录消息的工具。例如，数据服务样例中的 Start 方法检索 syslog() 函数并记录已启动数据服务的消息，如下所示：

```
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
...
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG] \
          "${ARGV0} HA-DNS successfully started"
fi
```

有关更多信息，请参见 scha_cluster_get(1HA) 手册页。

获取属性信息

大多数回调方法需要获取有关数据服务的资源和资源类型属性的信息。为实现此操作，API 提供了 scha_resource_get() 函数。

系统定义的属性和扩展属性都可用。系统定义的属性是预定义的。由您在 RTR 文件中定义扩展属性。

当使用 scha_resource_get() 获取系统定义的属性的值时，使用 -O 选项指定属性的名称。命令仅返回该属性的**值**。例如，在数据服务样例中，Monitor_start 方法需要找到探测程序，这样才能启动该程序。该探测程序位于数据服务的基目录中，由 RT_basedir 属性指向该基目录。Monitor_start 方法检索 RT_basedir 的值并将其放到 RT_BASEDIR 变量中，如下所示：

```
RT_BASEDIR=‘scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘
```

对于扩展属性，必须使用 -O 选项指定某属性为扩展属性。您还必须提供属性的名称，将其作为最后一个参数。对于扩展属性，命令将同时返回属性的**类型**和**值**。例如，在数据服务样例中，探测程序检索 Probe_timeout 扩展属性的类型和值并使用 awk 命令将值仅放于 PROBE_TIMEOUT shell 变量中，如下所示：

```
probe_timeout_info=‘scha_resource_get -O Extension \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME Probe_timeout‘
PROBE_TIMEOUT=‘echo $probe_timeout_info | awk ’{print $2}’‘
```

控制数据服务

数据服务必须提供 Start 或 Prenet_start 方法以在群集中激活应用程序守护进程，还必须提供 Stop 或 Postnet_stop 方法以在群集中停止应用程序守护进程。数据服务样例中实现了 Start 和 Stop 方法。有关何时改为使用 Prenet_start 和 Postnet_stop 的信息，请参见第 41 页中的 “确定使用哪种 Start 或 Stop 方法”。

Start 方法的工作方式

当包含数据服务资源的资源组在某节点上进入联机状态或该资源组已联机并启用了该资源时，RGM 将在该群集节点上运行 Start 方法。在样例应用程序中，Start 方法在该节点上激活 in.named DNS 守护进程。

本小节介绍了应用程序样例中的 Start 方法的重要方面，本节没有介绍所有回调方法通用的功能，例如 parse_args() 函数。本节也没有介绍 syslog() 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 Start 方法的完整列表，请参见第 246 页中的 “Start 方法代码列表”。

Start 方法的用途

在尝试启动 DNS 之前，数据服务样例中的 Start 方法将验证是否可以访问配置目录和配置文件 (named.conf) 以及该目录和文件是否可用。named.conf 中的信息对 DNS 的成功操作是必需的。

该回调方法使用 PMF (pmfadm) 启动 DNS 守护进程 (in.named)。如果 DNS 崩溃或无法启动，则 PMF 将在指定的时间间隔内尝试按指定的次数启动 DNS 守护进程。重试的次数和时间间隔都是由数据服务的 RTR 文件中的属性指定的。

检验配置

要进行操作，DNS 需要位于配置目录下的 named.conf 文件中的信息。因此，Start 方法在尝试启动 DNS 之前，将执行一些完整性检查，以验证目录和文件是否可访问。

Confdir 扩展属性提供了指向配置目录的路径。属性本身是在 RTR 文件中定义的，但是，群集管理员在配置数据服务时需指定实际位置。

在数据服务样例中，Start 方法通过使用 scha_resource_get() 函数检索配置目录的位置。

注 – 因为 Confdir 是扩展属性，所以 scha_resource_get() 将同时返回类型和值。awk 命令仅检索值并将该值放入 shell 变量 CONFIG_DIR 中。

```
# find the value of Confdir set by the cluster administrator at the time of
# adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the "type" as well as the "value" for the
# extension properties. Get only the value of the extension property
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

Start 方法使用 CONFIG_DIR 的值来验证目录是否可访问。如果该目录不可访问，则 Start 将记录一条错误消息，并在出错状态下退出。请参见第 83 页中的 “Start 退出状态”。

```
# Check if $CONFIG_DIR is accessible.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG] \
          "${ARGV0} Directory $CONFIG_DIR is missing or not mounted"
   exit 1
fi
```

在启动应用程序守护进程之前，此方法将执行 final 检查，以检验 named.conf 文件是否存在。如果该文件不存在，则 Start 将记录一条错误消息，并在出错状态下退出。

```
# Change to the $CONFIG_DIR directory in case there are relative
# pathnames in the data files.
cd $CONFIG_DIR

# Check that the named.conf file is present in the $CONFIG_DIR directory
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG] \
          "${ARGV0} File $CONFIG_DIR/named.conf is missing or empty"
   exit 1
fi
```

启动应用程序

该方法使用进程管理器工具 (pmfadm) 来启动应用程序。使用 pmfadm 命令，可以设置在指定时间范围内尝试重新启动应用程序的次数。RTR 文件包含两种属性 : Retry_count 指定尝试重新启动应用程序的次数，Retry_interval 指定执行此操作的时间间隔。

Start 方法使用 scha_resource_get() 函数检索 Retry_count 和 Retry_interval 的值，并将其值存储于 shell 变量中。Start 方法使用 -n 和 -t 选项将这些值传递给 pmfadm。

```
# Get the value for retry count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`
# Get the value for retry interval from the RTR file. This value is in seconds
# and must be converted to minutes for passing to pmfadm. Note that the
# conversion rounds up; for example, 50 seconds rounds up to 1 minute.
((RETRY_INTRVAL=`scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME` / 60))

# Start the in.named daemon under the control of PMF. Let it crash and restart
# up to $RETRY_COUNT times in a period of $RETRY_INTERVAL; if it crashes
# more often than that, PMF will cease trying to restart it.
# If there is a process already registered under the tag
# <$PMF_TAG>, then PMF sends out an alert message that the
# process is already running.
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Log a message indicating that HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Start 退出状态

Start 方法在基本应用程序实际运行且可用之前不应该成功退出，特别是在其他数据服务依赖于它的情况下。一种验证成功的方法是探测应用程序，以确保在 Start 方法退出之前它已运行。对于复杂的应用程序，例如数据库，请确保将 RTR 文件中的 Start_timeout 属性值设置得足够高，以允许应用程序有初始化和从崩溃中恢复的时间。

注 – 由于数据服务样例中的应用程序资源 (DNS) 快速启动，因此数据服务样例不会进行轮询以验证在成功退出之前该资源是否已运行。

如果此方法无法启动 DNS 并在失败状态下退出，RGM 将检查用来确定如何作出反应的 Failover_mode 属性。数据服务样例不会明确设置 Failover_mode 属性，因此该属性的默认值为 NONE（除非群集管理员覆盖了默认值并指定了其他值）。在这种情况下，RGM 仅执行设置数据服务状态的操作。群集管理员需要在同一节点上执行重新启动，或故障转移到其他节点。

Stop 方法的工作方式

当使包含有 HA-DNS 资源的资源组在群集节点上脱机时，或当该资源组处于联机状态但资源已被禁用时，RGM 将在该群集节点上运行 Stop 方法。该方法将停止该节点上的 in.named (DNS) 守护进程。

本小节介绍了应用程序样例中的 Stop 方法的重要方面，本节没有介绍所有回调方法通用的功能，例如 parse_args() 函数。本节也没有介绍 syslog() 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 Stop 方法的完整列表，请参见第 249 页中的 “Stop 方法代码列表”。

Stop 方法的用途

当尝试停止数据服务时，需要考虑以下两个主要方面。第一方面是进行顺序停机。通过 pmfadm 发送 SIGTERM 信号是完成顺序停机的最佳方法。

第二方面是要确保数据服务确实已停止，以避免使其处于 Stop_failed 状态。达到将数据服务置于该状态的最佳方法是通过 pmfadm 发送一个 SIGKILL 信号。

数据服务样例中的 Stop 方法将同时考虑这两个方面。它先发送一个 SIGTERM 信号。如果该信号不能停止数据服务，该方法将发送 SIGKILL 信号。

在尝试停止 DNS 之前，此 Stop 方法将检验该进程是否确实正在运行。如果进程正在运行，则 Stop 将使用 PMF (pmfadm) 来停止进程。

确保此 Stop 方法具有幂等性。尽管 RGM 不应在没有首先调用其 Start 方法来启动数据服务的情况下两次调用 Stop 方法，RGM 也可以在资源上调用 Stop 方法，即使该资源从未启动或已自动关闭。因此，即使 DNS 未运行，此 Stop 方法也可以在成功状态下退出。

停止应用程序

Stop 方法提供了用来停止数据服务的双重方法：顺序或平稳方法通过 pmfadm 使用 SIGTERM 信号，突然或强硬方法使用 SIGKILL 信号。Stop 方法获取 Stop_timeout 值（时间量，Stop 方法必须在该期间返回）。Stop 将此时间的 80% 分配给平稳停止，15% 分配给突然停止（5% 保留），如以下样例代码所示。

```
STOP_TIMEOUT='scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME'
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))
```

```
((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))
```

Stop 方法将使用 pmfadm -q 检验 DNS 守护进程是否正在运行。如果 DNS 守护进程正在运行，Stop 首先使用 pmfadm -s 发送 TERM 信号终止 DNS 进程。如果经过超时值的 80% 后该信号还无法终止进程，Stop 将发送 SIGKILL 信号。如果该信号在超时值的 15% 范围内仍无法终止进程，方法将记录一条错误消息并以出错状态退出。

如果 pmfadm 终止了该进程，该方法将记录一条说明该进程已停止并在成功状态下退出的消息。

如果 DNS 进程未运行，该方法将记录一条说明该进程未运行并仍在成功状态下退出的消息。以下样例代码说明了 Stop 如何使用 pmfadm 停止 DNS 进程。

```
# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG; then
   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $RESOURCE_NAME.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err \
          -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"

      # Since the data service did not stop with a SIGTERM signal, use
      # SIGKILL now and wait for another 15% of the total timeout value.
      pmfadm -s $PMF_TAG -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESSFUL"
         exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS is not started"

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service resource in STOP_FAILED State.
   exit 0
fi

# Could successfully stop DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
    "HA-DNS successfully stopped"
exit 0
```

Stop 退出状态

Stop 方法在基本应用程序实际停止之前不应该成功退出，特别是在其他数据服务依赖于它的情况下。如果未做到这一点，可能会导致数据毁坏。

对于复杂的应用程序（例如数据库），请确保在 RTR 文件中设置了足够大的 Start_timeout 属性值，使应用程序有足够长的时间在停止时进行清除。

如果此方法无法停止 DNS 并在失败状态下退出，RGM 将检查用来确定如何作出反应的 Failover_mode 属性。数据服务样例不会明确设置 Failover_mode 属性，因此该属性的默认值为 NONE（除非群集管理员覆盖了默认值并指定了其他值）。这种情况下，RGM 仅将数据服务的状态设置为 Stop_failed，而不会采取其他操作。群集管理员需要强制停止应用程序并清除 Stop_failed 状态。

定义故障监视器

样例应用程序执行基本的故障监视，以监视 DNS 资源 (in.named) 的可靠性。故障监视器由以下元素组成：

- dns_probe，用户定义的程序，使用 nslookup 验证由数据服务样例控制的 DNS 资源是否正在运行。如果 DNS 未运行，此方法将尝试在本地重启该 DNS，或按照重启尝试的次数请求 RGM 将数据服务重定位到其它节点。
- dns_monitor_start，启动 dns_probe 的回调方法。如果启用了监视，RGM 将在数据服务样例联机之后自动调用 dns_monitor_start。
- dns_monitor_stop，停止 dns_probe 的回调方法。使数据服务样例脱机之前，RGM 自动调用 dns_monitor_stop。
- dns_monitor_check，一种回调方法，当 PROBE 程序将数据服务故障转移到新节点时该方法调用 Validate 方法，以验证配置目录是否可用。

探测程序的工作方式

dns_probe 程序执行连续运行过程，验证由数据服务样例控制的 DNS 资源是否正在运行。dns_probe 由 dns_monitor_start 方法启动，该方法在数据服务样例联机之后由 RGM 自动运行。数据服务由 dns_monitor_stop 方法停止，RGM 在使数据服务样例脱机之前运行该方法。

本小节介绍了应用程序样例中的PROBE 方法的重要方面，它没有介绍所有回调方法通用的功能，例如 parse_args() 函数。本节也没有介绍 syslog() 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 PROBE 方法的完整列表，请参见第 252 页中的 “PROBE 程序代码列表”。

探测程序的用途

探测程序以死循环的形式运行。它使用 nslookup 验证正确的 DNS 资源是否正在运行。如果 DNS 正在运行，探测将休眠预定的时间间隔（由系统定义的属性 Thorough_probe_interval 设置），然后再次检查。如果 DNS 未运行，此程序将尝试在本地重启 DNS，或按照重启尝试的次数请求 RGM 将数据服务重定位到其它节点。

获取属性值

此程序需要以下属性的值：

- Thorough_probe_interval – 用来设置探测程序休眠的时间段
- Probe_timeout??在执行探测的 nslookup 命令上强制执行探测的超时值
- Network_resources_used — 用来获取 DNS 服务器的 IP 地址
- Retry_count 和 Retry_interval??确定尝试重新启动的次数以及计算这些次数的时间段
- RT_basedir??获取包含 PROBE 程序和 gettime 实用程序的目录

scha_resource_get() 函数获取这些属性的值并将其存储于 shell 变量中，如下所示：

```
PROBE_INTERVAL=‘scha_resource_get -O Thorough_probe_interval \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME‘

PROBE_TIMEOUT_INFO=‘scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Probe_timeout‘
Probe_timeout=‘echo $probe_timeout_info | awk ’{print $2}’‘

DNS_HOST=‘scha_resource_get -O Network_resources_used -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘

RETRY_COUNT=‘scha_resource_get -O Retry_count -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

RETRY_INTERVAL=‘scha_resource_get -O Retry_interval -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

RT_BASEDIR=‘scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
 $RESOURCEGROUP_NAME‘
```

注 – 对于系统定义的属性，例如 Thorough_probe_interval，scha_resource_get() 函数仅返回值。对于扩展属性，例如 Probe_timeout，scha_resource_get() 函数返回类型和值。使用 awk 命令仅获取值。

检查服务的可靠性

探测本身是 nslookup 命令的无限 while 循环。while 循环之前，将设置一个临时文件以放置 nslookup 回复。probefail 和 retries 变量均初始化为 0。

```
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

while 循环执行以下任务：

- 设置探测程序的休眠间隔
- 使用 hatimerun 启动 nslookup，传递 Probe_timeout 值并标识目标主机
- 根据 nslookup 返回码成功与否设置 probefail 变量
- 如果 probefail 设置为 1（失败），验证对 nslookup 的回复来自于数据服务样例，而不是某一其他 DNS 服务器

下面是 while 循环代码。

```
while :
do
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep
   # for a duration of THOROUGH_PROBE_INTERVAL.
   sleep $PROBE_INTERVAL

   # Run an nslookup command of the IP address on which DNS is serving.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
   > $DNSPROBEFILE 2>&1

      retcode=$?
      if [ $retcode -ne 0 ]; then
            probefail=1
      fi

   # Make sure that the reply to nslookup comes from the HA-DNS
   # server and not from another nameserver mentioned in the
   # /etc/resolv.conf file.
   if [ $probefail -eq 0 ]; then
# Get the name of the server that replied to the nslookup query.
   SERVER=` awk ' $1=="Server:" { print $2 }' \
   $DNSPROBEFILE | awk -F. ' { print $1 } ' `
   if [ -z "$SERVER" ]; then
      probefail=1
      else
         if [ $SERVER != $DNS_HOST ]; then
            probefail=1
         fi
   fi
fi
```

比较重新启动和故障转移

如果 probefail 变量是非 0 值（成功），则 nslookup 命令超时，或回复来自于服务样例的 DNS 以外的服务器。在每种情况下，DNS 服务器都不能按照预期的情况发挥作用，且故障监视器将调用 decide_restart_or_failover() 函数以确定是在本地重启数据服务，还是请求 RGM 将数据服务重定位到其它节点。如果 probefail 变量为 0，将生成探测成功的消息。

```
if [ $probefail -ne 0 ]; then
      decide_restart_or_failover
else
      logger -p ${SYSLOG_FACILITY}.err\
      -t [$SYSLOG_TAG]\
      "${ARGV0} Probe for resource HA-DNS successful"
fi
```

decide_restart_or_failover() 函数使用时间窗口 (Retry_interval) 和故障计数 (Retry_count) 来确定是在本地重新启动 DNS，还是请求 RGM 将数据服务重新定位到其他节点。该函数执行以下条件逻辑。第 252 页中的 “PROBE 程序代码列表” 中 decide_restart_or_failover() 的代码列表包含代码。

- 如果这是首次故障，请重启数据服务。记录一条错误消息并取消 retries 变量中的计数器。
- 如果不是首次故障，但是时间已经超出了窗口的范围，请重启数据服务。记录一条错误消息，复位计数器并滑动窗口。
- 如果时间仍在窗口范围内，但是已超过重试计数器，请故障转移到另一个节点。如果故障转移不成功，将记录错误并以状态 1（失败）退出探测程序。
- 如果时间仍处于窗口的范围内，但是未超出重试计数器的计数范围，请重启数据服务。记录错误消息，并在 retries 变量中撞击计数器。

如果在指定时间间隔内达到了重启的最大次数，函数将请求 RGM 将数据服务重定位到其它节点。如果重启的次数在所限制范围之内，或者已超出了时间间隔，以致重新开始计数时，该函数将尝试在同一节点上重启 DNS。请注意以下关于此函数的信息：

- gettime 实用程序用来跟踪两次重启操作之间的时间。这是一个 C 程序，位于 (RT_basedir) 目录中。
- Retry_count 和 Retry_interval 系统定义的资源属性确定尝试重新启动的次数以及计数的时间间隔。在 RTR 文件中，这些属性的默认值为在时间段 5 分钟（300 秒）内进行两次尝试，尽管群集管理员可以更改这些值。
- 系统将调用 restart_service() 函数尝试在同一节点上重新启动数据服务。有关该函数的信息，请参见下一节，第 89 页中的 “重启数据服务” 。
- scha_control() API 函数带有 GIVEOVER 选项使包含数据服务样例的资源组脱机并在其他节点上使其重新联机。

重启数据服务

decide_restart_or_failover() 调用 restart_service() 函数，以尝试在同一节点上重新启动数据服务。 该函数执行以下逻辑：

- 确定数据服务是否仍在 PMF 之下注册。如果服务仍处于注册状态，函数将执行以下操作：
  - 为数据服务获取 `Stop` 方法名称和 `Stop_timeout` 值
  - 使用 `hatimerun` 为数据服务启动 `Stop` 方法，传递 `Stop_timeout` 值
  - 如果数据服务成功停止，则为数据服务获取 `Start` 方法名称和 `Start_timeout` 值
  - 使用 `hatimerun` 为数据服务启动 `Start` 方法，传递 `Start_timeout` 值
- 如果数据服务已不在 PMF 之下注册，则表示数据服务已超过了 PMF 下允许的最大重试次数。系统将使用 `GIVEOVER` 选项调用 `scha_control()` 函数，以将数据服务故障转移到其他节点。

```
function restart_service
{

        # To restart the data service, first verify that the
        # data service itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the data service is still registered under
                # PMF, first stop the data service and start it back up again.

                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                STOP_METHOD=`scha_resource_get -O STOP \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                                "${ARGV0} Stop method failed."
                        return 1
                fi

                # Obtain the START method name and the START_TIMEOUT value for
                # this resource.
                START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                START_METHOD=`scha_resource_get -O START \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                                "${ARGV0} Start method failed."
                        return 1
                fi
```

```
        else
                # The absence of the TAG for the dataservice
                # implies that the data service has already
                # exceeded the maximum retries allowed under PMF.
                # Therefore, do not attempt to restart the
                # data service again, but try to failover
                # to another node in the cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}
```

探测程序退出状态

如果尝试在本地重新启动失败，并且尝试故障转移到其他节点也失败，则数据服务样例的 PROBE 程序将以失败退出。此程序将记录消息 `Failover attempt failed`。

Monitor_start 方法的工作方式

RGM 调用 `Monitor_start` 方法，以在使数据服务样例联机之后启动 `dns_probe` 方法。

本节介绍样例应用程序的 `Monitor_start` 方法的主要方面。本节没有介绍所有回调方法通用的功能，例如 `parse_args()` 函数。本节也没有介绍 `syslog()` 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 `Monitor_start` 方法的完整列表，请参见第 257 页中的 “Monitor_start 方法代码列表”。

Monitor_start 方法的用途

该方法使用 PMF (`pmfadm`) 来启动探测。

启动探测程序

`Monitor_start` 方法获取 `RT_basedir` 属性的值，用来为 PROBE 程序构造完整的路径名。该方法通过使用 `pmfadm` 的无限重试选项（`-n -1`、`-t -1`）启动探测，意味着如果探测无法启动，PMF 将在无限时间期限内无限次地启动探测。

```
# Find where the probe program resides by obtaining the value of the
# RT_basedir property of the resource.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
```

```
$RESOURCEGROUP_NAME`

# Start the probe for the data service under PMF. Use the infinite retries
# option to start the probe. Pass the resource name, type, and group to the
# probe program.
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
   $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME
```

Monitor_stop 方法的工作方式

数据服务样例脱机时，RGM 调用 Monitor_stop 方法以停止 dns_probe 的执行。

本节介绍样例应用程序的 Monitor_stop 方法的主要方面。本节没有介绍所有回调方法通用的功能，例如 parse_args() 函数。本节也没有介绍 syslog() 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 Monitor_stop 方法的完整列表，请参见第 259 页中的 “Monitor_stop 方法代码列表”。

Monitor_stop 方法的用途

该方法使用 PMF (pmfadm)，以检查探测是否正在运行，如果是则将其停止。

停止监视器

Monitor_stop 方法将使用 pmfadm -q 查看探测程序是否正在运行，如果正在运行，将使用 pmfadm -s 停止该程序。如果探测程序已停止，该方法仍将成功退出，这可以确保该方法的幂等性。

注意 – 请确保使用 KILL 信号和 pmfadm 来停止探测，而不应使用可以屏蔽的信号（例如 TERM）。否则，Monitor_stop 方法可能会无限期挂起并最终超时。原因是 PROBE 方法在必须重新启动或故障转移数据服务时调用 scha_control()。作为使数据服务脱机过程的一部分，当 scha_control() 调用 Monitor_stop 时，如果 Monitor_stop 使用可以屏蔽的信号，则 Monitor_stop 将挂起，等待 scha_control() 完成，并且 scha_control() 将挂起等待 Monitor_stop 完成。

```
# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG; then
   pmfadm -s $PMF_TAG KILL
   if [ $? -ne 0 ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Could not stop monitor for resource " \
            $RESOURCE_NAME
```

```
          exit 1
   else
         # could successfully stop the monitor. Log a message.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor for resource " $RESOURCE_NAME \
            " successfully stopped"
   fi
fi
exit 0
```

Monitor_stop 退出状态

如果 Monitor_stop 方法无法停止 PROBE 方法，它将记录一条错误消息。RGM 使数据服务样例在主节点上进入 MONITOR_FAILED 状态，这样可以应急该节点。

在探测程序停止之前，Monitor_stop 不应退出。

Monitor_check 方法的工作方式

无论 PROBE 方法何时尝试将包含数据服务的资源组故障转移到新节点上，RGM 都将调用 Monitor_check 方法。

本节介绍样例应用程序的 Monitor_check 方法的主要方面。本节没有介绍所有回调方法通用的功能，例如 parse_args() 函数。本节也没有介绍 syslog() 函数的使用。通用功能在第 77 页中的 “为所有方法提供通用功能”中进行介绍。

有关 Monitor_check 方法的完整列表，请参见第 260 页中的 “Monitor_check 方法代码列表”。

Monitor_check 方法必须执行，因此它不会与并发运行的其他方法发生冲突。

Monitor_check 方法调用 Validate 方法以验证 DNS 配置目录在新节点上是否可用。Confdir 扩展属性指向 DNS 配置目录。因此，Monitor_check 获取 Validate 方法的路径和名称以及 Confdir 的值。它将把此值传送到 Validate，如下所示。

```
# Obtain the full path for the Validate method from
# the RT_basedir property of the resource type.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAM?

# Obtain the name of the Validate method for this resource.
VALIDATE_METHOD=`scha_resource_get -O Validate \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?

# Obtain the value of the Confdir property in order to start the
# data service. Use the resource name and the resource group entered to
# obtain the Confdir value set at the time of adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
```

```
 -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the type as well as the value for extension
# properties. Use awk to get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Call the validate method so that the dataservice can be failed over
# successfully to the new node.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

请参见第 94 页中的 “Validate 方法的工作方式”，以查看样例应用程序如何验证某节点主管数据服务的适当性。

处理属性更新

数据服务样例执行 Validate 和 Update 方法，以处理群集管理员对属性的更新。

Validate 方法的工作方式

创建资源时以及通过管理操作更新资源或其资源组的属性时，RGM 将调用 Validate 方法。在进行创建或更新之前，RGM 将调用 Validate，如果从任何节点上的方法返回失败出口代码都将导致创建或更新操作取消。

仅当群集管理员更改资源或资源组属性时 RGM 才调用 Validate，RGM 设置属性或监视器设置资源属性 Status 和 Status_msg 时 RGM 不执行该调用。

注 – 只要 PROBE 方法尝试将数据服务故障转移到新节点，Monitor_check 方法也会明确调用 Validate 方法。

Validate 方法的用途

RGM 使用传递给其他方法的其他参数调用 Validate，包括要更新的属性和值。因此，数据服务样例中的此方法必须执行不同的 parse_args() 函数以处理其他参数。

数据服务样例中的 Validate 方法用来检验单个属性 Confdir 扩展属性。此属性指向 DNS 配置目录，这对于能否成功地进行 DNS 操作来说至关重要。

注 – 由于在 DNS 运行时不能更改配置目录，所以 Confdir 属性在 RTR 文件中被声明为 TUNABLE = AT_CREATION。因此，更新后系统从不调用 Validate 方法来验证 Confdir 属性，仅在创建数据服务资源时执行该操作。

如果 Confdir 是 RGM 传递给 Validate 的属性之一，parse_args() 函数将检索并保存其值。Validate 验证 Confdir 的新值所指向的目录是否可访问，以及该目录中是否存在 named.conf 文件并且该文件是否包含数据。

如果 parse_args() 函数无法从由 RGM 传递的命令行参数中检索 Confdir 的值，Validate 仍会尝试验证 Confdir 属性。Validate 使用 scha_resource_get() 从静态配置中获取 Confdir 的值。Validate 执行同样的检查，验证配置目录是否可访问并且是否包含非空文件 named.conf。

如果 Validate 在失败状态下退出，所有属性（而不仅仅是 Confdir）的更新或创建操作都将失败。

Validate 方法分析函数

由于 RGM 向 Validate 方法传递了一组与其他回调方法不同的参数，因此 Validate 需要与其他方法不同的函数来解析这些参数。有关传递给 Validate 和其他回调方法的参数的更多信息，请参见 rt_callbacks(1HA) 手册页。以下样例代码显示了 Validate 的 parse_args() 函数。

```
#############################################################################
# Parse Validate arguments.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Name of the DNS resource.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Name of the resource group in which the resource is
                  # configured.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
         T)
                  # Name of the resource type.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
         r)
                  # The method is not accessing any system defined
```

```
                # properties so this is a no-op
                ;;
        g)
                # The method is not accessing any resource group
                # properties, so this is a no-op
                ;;
        c)
                # Indicates the Validate method is being called while
                # creating the resource, so this flag is a no-op.
                ;;
        u)
                # Indicates the updating of a property when the
                # resource already exists. If the update is to the
                # Confdir property then Confdir should appear in the
                # command-line arguments. If it does not, the method must
                # look for it specifically using scha_resource_get.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Extension property list. Separate the property and
                # value pairs using "=" as the separator.
                PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                VAL=`echo $OPTARG | awk -F= '{print $2}'`
                # If the Confdir extension property is found on the
                # command line, note its value.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
   done
}
```

相对其他方法的 `parse_args()` 函数，该函数提供了标志 (`R`) 以捕获资源名称，(`G`) 以捕获资源组名称，(`T`) 以捕获 RGM 传递的资源类型。

忽略 `r` 标志（表示系统定义的属性）`g` 标志（表示资源组属性）和 `c` 标志（表示资源创建期间进行了验证）。忽略它们，是因为更新资源时，系统调用该方法验证扩展属性。

`u` 标记将 `UPDATE_PROPERTY` shell 变量的值设置为 `1` (`TRUE`)。`x` 标志捕获要更新的属性名称和值。如果 `Confdir` 是要更新的属性之一，其值将被放于 shell 变量 `CONFDIR` 中，并且变量 `CONFDIR_FOUND` 将被设置为 `1` (`TRUE`)。

验证 `Confdir`

在其 `MAIN` 函数中，`Validate` 首先将 `CONFDIR` 变量设置为空字符串，并将 `UPDATE_PROPERTY` 和 `CONFDIR_FOUND` 设置为 `0`。

```
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0
```

Validate 调用 parse_args() 解析由 RGM 传递的参数。

```
parse_args "$@"
```

Validate 检查 Validate 是否因属性更新而被调用。Validate 还会检查 Confdir 扩展属性是否在命令行中。Validate 验证 Confdir 属性是否有值，如果没有，则以失败状态退出并显示错误消息。

```
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
         config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
            -G $RESOURCEGROUP_NAME Confdir`
         CONFDIR=`echo $config_info | awk '{print $2}'`
fi

# Verify that the Confdir property has a value. If not there is a failure
# and exit with status 1
if [[ -z $CONFDIR ]]; then
         logger -p ${SYSLOG_FACILITY}.err \
            "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
         exit 1
fi
```

注 – 特别地，上述代码检查是否因更新而调用 Validate ($UPDATE_PROPERTY == 1) 以及是否**未**在命令行中找到该属性 (CONFDIR_FOUND == 0)。在这种情况下，代码将使用 scha_resource_get() 检索 Confdir 的现有值。如果在命令行中找到了 Confdir (CONFDIR_FOUND == 1)，则 CONFDIR 的值来自于 parse_args() 函数，而不是来自于 scha_resource_get()。

Validate 方法使用 CONFDIR 的值来验证目录是否可访问。如果目录不可访问，Validate 将记录一条错误消息并以出错状态退出。

```
# Check if $CONFDIR is accessible.
if [ ! -d $CONFDIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Directory $CONFDIR missing or not mounted"
   exit 1
fi
```

在验证 Confdir 属性的更新之前，Validate 将执行 final 检查来检验 named.conf 文件是否存在。如果文件不存在，该方法将记录一条错误消息并以出错状态退出。

```
# Check that the named.conf file is present in the Confdir directory
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
         exit 1
```

```
fi
```

如果通过了最终检查，Validate 将记录表示成功的消息并以成功状态退出。

```
# Log a message indicating that the Validate method was successful.
logger -p ${SYSLOG_FACILITY}.err \
   -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

Validate 退出状态

如果 Validate 以成功状态退出 (0)，系统将用新值创建 Confdir。如果 Validate 以失败状态退出 (1)，则不会创建 Confdir 和其他任何属性，并生成指出原因的消息。

Update 方法的工作方式

RGM 通过运行 Update 方法来通知正在运行的资源其属性已更改。群集管理员成功设置了资源或其组的属性后，RGM 将运行 Update。在资源联机所在的节点上调用该方法。

Update 方法的用途

Update 方法不更新属性。RGM 更新属性。Update 方法通知正在运行的进程已执行了更新。在数据服务样例中，受属性更新影响的唯一进程是故障监视器。因而，故障监视器进程是 Update 方法停止并重新启动的进程。

Update 方法必须验证故障监视器是否正在运行，然后使用 pmfadm 命令中止它。该方法获取执行故障监视的探测程序的位置，并使用 pmfadm 命令将其重新启动。

使用 Update 停止监视器

Update 方法使用 pmfadm -q 验证监视器是否正在运行，如果是，则使用 pmfadm -s TERM 中止它。如果监视器成功终止，将会向群集管理员发送一条描述该结果的消息。如果无法停止监视器，Update 将以失败状态退出并向群集管理员发送一条错误消息。

```
if pmfadm -q $RESOURCE_NAME.monitor; then

# Kill the monitor that is running already
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
```

```
                "${ARGV0} Could not stop the monitor"
      exit 1
   else
   # could successfully stop DNS. Log a message.
      logger -p ${SYSLOG_FACILITY}.err \
             -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                "Monitor for HA-DNS successfully stopped"
   fi
```

重启监视器

要重新启动监视器，Update 方法必须定位实现探测程序的脚本。探测程序位于数据服务的基目录中，RT_basedir 属性指向该基目录。Update 检索 RT_basedir 的值并将其存储于 RT_BASEDIR 变量中，如下所示。

```
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME`
```

Update 使用 RT_BASEDIR 的值和 pmfadm 重新启动 dns_probe 程序。如果成功，Update 将以成功状态退出并向群集管理员发送一条描述该结果的消息。如果 pmfadm 无法启动探测程序，Update 将以失败状态退出并记录一条错误消息。

Update 退出状态

Update 方法的失败将导致资源处于“更新失败”状态。该状态对资源的 RGM 管理没有影响，但是会通过 syslog() 函数将更新操作的失败告知管理工具。

第 6 章

数据服务开发库

本章概述了组成数据服务开发库 (DSDL) 的应用程序编程接口。DSDL 是在 `libdsdev.so` 库中实现的，它包含在 Sun Cluster 软件包中。

本章包括以下主题：

DSDL 概述

DSDL API 位于资源管理应用程序编程接口 (RMAPI) 的顶层。因此，DSDL API 本身不取代 RMAPI，而是封装并扩展了 RMAPI 的功能。通过提供针对特定的 Sun Cluster 集成问题的预设解决方案，DSDL 简化了数据服务开发工作，这样，您就可以将主要开发时间投入到应用程序内在的高可用性和可伸缩性问题上。将应用程序启动、关闭和监视过程与 Sun Cluster 相集成花费的时间更少。

管理配置属性

所有回调方法都需要访问配置属性。DSDL 采用以下方式支持对属性的访问：

- 初始化环境
- 提供一组公用函数以检索属性值

scds_initialize() 函数必须在每个回调方法的开头进行调用，它执行以下操作：

- 检查并处理 RGM 传递给回调方法的命令行参数（argc 和 argv[]），无需您写入命令行解析函数。
- 设置其他 DSDL 函数使用的内部数据结构。例如，从 RGM 检索属性值的公用函数将在这些结构中存储检索出来的值，同样地，来自命令行的值将优先于从 RGM 检索的值存储在这些数据结构中。
- 初始化日志环境并验证故障监视器探测设置。

注 – 对于 Validate 方法，scds_initialize() 将解析通过命令行传递的属性值，而无需写入 Validate 的解析函数。

DSDL 提供了几组函数用于检索资源类型、资源、资源组属性以及常用扩展属性。这些函数通过使用以下约定对属性的访问进行了标准化：

- 每个函数仅使用一个句柄参数（scds_initialize() 返回）。
- 每个函数都对应一个特定的属性。函数的返回值类型与其检索的属性值类型相匹配。
- 由于已预先通过 scds_initialize() 计算了这些值，因此，函数不会返回错误。除非向命令行传递新值，否则函数将从 RGM 检索各个值。

启动和停止数据服务

Start 方法用于执行在群集节点上启动数据服务所需的操作。通常，这些操作包括检索资源属性、查找应用程序特定的可执行文件和配置文件，以及使用正确的命令参数启动应用程序。

scds_initialize() 函数用于检索资源配置。Start 方法可以使用属性公用函数来检索特定属性（例如，Confdir_list，它标识用于应用程序启动的配置目录和文件）的值。

Start 方法可以调用 scds_pmf_start() 来启动在进程监视器工具 (PMF) 控制下的应用程序。PMF 使您可以指定要应用到进程的监视级别，它还提供了在失败的情况下重新启动进程的能力。有关使用 DSDL 实现的 Start 方法的示例，请参见第 118 页中的“xfnts_start 方法”。

Stop 方法必须是幂等的，这样即使在应用程序没在运行的节点上进行调用，Stop 方法也可以成功退出。如果 Stop 方法失败，将把正在停止的资源设置为 STOP_FAILED 状态，该状态可以导致群集执行硬重启。

要避免将资源置入 STOP_FAILED 状态，Stop 方法必须尽力停止资源。scds_pmf_stop() 函数用于阶段性尝试停止资源。此函数将先尝试使用 SIGTERM 信号停止资源，如果失败，则使用 SIGKILL 信号。有关更多信息，请参见 scds_pmf_stop(3HA) 手册页。

实现故障监视器

DSDL 通过提供预确定的模型极大地降低了实现故障监视器的复杂程度。在节点上启动资源时，Monitor_start 方法用于启动在 PMF 控制下的故障监视器。只要资源还在节点上运行，故障监视器就会一直循环运行。DSDL 故障监视器的高级逻辑为：

- scds_fm_sleep() 函数使用 Thorough_probe_interval 属性来确定探测之间的时间间隔。在此时间间隔内被 PMF 检测到的任何应用程序进程失败都将导致重新启动资源。
- 探测本身将返回一个表示失败严重程度的值，该值的范围从 0（无失败）到 100（完全失败）。
- 探测返回值被发送给 scds_action() 函数，该函数保留在 Retry_interval 属性的时间间隔内累积的失败历史记录。
- scds_action() 函数用于确定失败时执行哪些操作，如下所示：
  - 如果累积失败值小于 100，不执行任何操作。
  - 如果累积失败值达到 100（完全失败），则重新启动数据服务。如果超出了 Retry_interval，则重置历史记录。
  - 如果在 Retry_interval 指定的时间内重新启动的次数超出 Retry_count 属性的值，则将数据服务故障转移。

访问网络地址信息

DSDL 提供了用于返回资源和资源组的网络地址信息的公用函数。例如，scds_get_netaddr_list() 将检索资源所使用的网络地址资源，从而使故障监视器可以探测应用程序。

DSDL 还提供了一组用来进行基于 TCP 的监视操作的函数。通常，这些函数用于建立与服务的简单套接字连接、向服务器读取和写入数据，以及断开与服务的连接。可以将探测的结果发送到 DSDL scds_fm_action() 函数以确定要采取的操作。

有关基于 TCP 的故障监视的示例，请参见第 130 页中的 “xfnts_validate 方法”。

调试资源类型实现

DSDL 有一些内置功能，可用于帮助调试数据服务。

DSDL 公用程序 `scds_syslog_debug()` 提供了一个基本框架，可用来向资源类型实现添加调试语句。可以在每个群集节点上为每个资源类型实现动态设置调试**级别**（级别范围从 1 到 9）。所有资源类型回调方法都读取名为 `/var/cluster/rgm/rt/`*rtname*`/loglevel` 的文件，该文件仅包含一个 1 至 9 之间的整数。DSDL 函数 `scds_initialize()` 将读取此文件并在内部将调试级别设置为指定级别。默认的调试级别 0 用于指定数据服务不记录调试消息。

`scds_syslog_debug()` 函数将在 `LOG_DEBUG` 优先级上使用 `scha_cluster_getlogfacility()` 函数所返回的工具。可以在 `/etc/syslog.conf` 文件中配置这些调试消息。

可以使用 `scds_syslog()` 函数将一些调试消息转换为资源类型的常规操作的信息型消息（可能属于 `LOG_INFO` 优先级）。请注意，第 8 章中的样例 DSDL 应用程序大量使用了 `scds_syslog_debug()` 和 `scds_syslog()` 函数。

启用具有高可用性的本地文件系统

可以使用 `HAStoragePlus` 资源类型使本地文件系统在 Sun Cluster 环境中具有高可用性。本地文件系统分区必须位于全局磁盘组中。必须启用关联切换转移，并且必须针对故障转移配置 Sun Cluster 环境。此设置使群集管理员可以通过任何直接连接到多主机磁盘的主机访问位于那些多主机磁盘上的任何文件系统。对于一些 I/O 增强数据服务，强烈建议使用具有高可用性的本地文件系统。《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》中的“Enabling Highly Available Local File Systems”包含关于配置 `HAStoragePlus` 资源类型的信息。

第 7 章

设计资源类型

本章介绍了数据服务开发库 (DSDL) 在设计和实现资源类型方面的典型应用，还着重介绍了设计资源类型以验证资源配置以及启动、停止和监视该资源等内容。另外，本章介绍了如何使用 DSDL 来实现资源类型回调方法。

有关附加信息，请参见 `rt_callbacks`(1HA) 手册页。

要完成这些任务，您需要访问资源的属性设置。DSDL 实用程序 `scds_initialize()` 提供了访问这些资源属性的统一方法。本函数设计成在每个回调方法的开头部分进行调用。此实用程序函数用于从群集框架中检索资源的所有属性，并使其可用于 `scds_get`*name*`()` 函数系列。

本章包括以下主题：

资源类型注册文件

资源类型注册 (RTR) 文件指明有关 Sun Cluster 软件中的资源类型的详细信息。包括以下详细信息：

- 实现所需要的属性

- 这些属性的数据类型和默认值
- 用于资源类型实现的回调方法的文件系统路径
- 系统定义的属性的各种设置

DSDL 附带的样例 RTR 文件对于大多数资源类型实现来说是足够的。您只需编辑一些基本元素，例如，资源类型回调方法的资源类型名称和路径名称。如果需要新属性才能实现资源类型，可以将其声明为资源类型实现的 RTR 文件中的扩展属性，并可通过 DSDL `scds_get_ext_property()` 实用程序访问该属性。

Validate 方法

资源类型实现的 `Validate` 回调方法用于检查提议的资源设置（由对资源提议的属性设置指定）对于资源类型来说是否可以接受。

在以下两种情况下，资源组管理器 (RGM) 将调用资源类型实现的 `Validate` 方法：

- 创建属于该资源类型的新资源时
- 更新资源或资源组的属性时

这两种情况可以通过传递给资源的 `Validate` 方法的命令行选项是 `-c`（创建）还是 `-u`（更新）进行区分。

`Validate` 方法在节点集的每个节点上都要调用，该节点集由资源类型属性 `Init_nodes` 的值定义。如果 `Init_nodes` 设置为 `RG_PRIMARIES`，则在可以托管包含资源的资源组的每个节点（主节点）上调用 `Validate`。如果 `Init_nodes` 设置为 `RT_INSTALLED_NODES`，则在安装资源类型软件的每个节点（通常是群集中的所有节点）上调用 `Validate`。

`Init_nodes` 的默认值为 `RG_PRIMARIES`（请参见 `rt_reg`(4) 手册页）。在调用 `Validate` 方法时，RGM 尚未创建资源（在创建回调的情况下）或尚未应用正被更新的属性的更新值（在更新回调的情况下）。

注 – 如果使用的是由 `HAStoragePlus` 资源类型管理的本地文件系统，请使用 `scds_hasp_check()` 函数检查该资源类型的状态。此信息来自该资源所依赖的所有 `SUNW.HAStoragePlus` 资源的状态（联机或脱机），它是通过使用为该资源定义的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系统属性获得的。有关 `scds_hasp_check()` 函数返回的状态代码的完整列表，请参见 `scds_hasp_check`(3HA) 手册页。

DSDL 函数 `scds_initialize()` 按照以下方式处理这些情况：

- 如果正在创建资源，则通过命令行传递提议资源属性时，`scds_initialize()` 就会将其解析。这样，您就可以使用提议资源属性值，就好像已在系统中创建了该资源一样。

- 如果正在更新资源或资源组，则从命令行读入正被群集管理员更新的提议属性值。其他属性（其属性值尚未被更新）则通过使用资源管理 API 从 Sun Cluster 读入。如果使用的是 DSDL，则无需考虑这些任务。可以验证某个资源的所有属性是否均可用。

假设实现资源属性验证的函数名为 `svc_validate()`，该函数使用 `scds_get_`*name*`()` 系列函数查看要验证的属性。假定可接受的资源设置由此函数的 0 返回代码表示，则资源类型的 `Validate` 方法可表示为以下代码段：

```
int
main(int argc, char *argv[])
{
   scds_handle_t handle;
   int rc;

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Initialization Error */
   }
   rc = svc_validate(handle);
   scds_close(&handle);
   return (rc);
}
```

验证函数还应当记录资源验证失败的原因。不过，如果忽略细节（第 8 章包含更实际的验证函数处理），可以实现更简单的示例 `svc_validate()` 函数，如下所示：

```
int
svc_validate(scds_handle_t handle)
{
   scha_str_array_t *confdirs;
   struct stat    statbuf;
   confdirs = scds_get_confdir_list(handle);
   if (stat(confdirs->str_array[0], &statbuf) == -1) {
   return (1);   /* Invalid resource property setting */
   }
   return (0);   /* Acceptable setting */
}
```

因此，只需考虑 `svc_validate()` 函数的实现。

Start 方法

RGM 将对所选群集节点调用资源类型实现的 `Start` 回调方法，以启动该资源。通过命令行传送资源组名称、资源名称和资源类型名称。`Start` 方法用于执行在群集节点中启动数据服务资源所需的操作。通常，这涉及使用正确的命令行参数检索资源属性、找到应用程序特定的可执行文件和/或配置文件以及启动应用程序。

使用 DSDL，资源配置已经由 `scds_initialize()` 公用程序检索。应用程序的启动操作可以包含在函数 `svc_start()` 中。可以调用另一个函数 `svc_wait()` 来检验是否确实启动了应用程序。`Start` 方法的简化代码如下：

```
int
main(int argc, char *argv[])
{
   scds_handle_t handle;

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Initialization Error */
   }
   if (svc_validate(handle) != 0) {
   return (1);   /* Invalid settings */
   }
   if (svc_start(handle) != 0) {
   return (1);   /* Start failed */
   }
   return (svc_wait(handle));
}
```

该启动方法实现调用 `svc_validate()` 来验证资源配置。如果失败，其原因不是资源配置与应用程序配置不匹配，就是此群集节点上当前存在系统问题。例如，资源所需的群集文件系统当前可能无法在此群集节点上使用。在这种情况下，尝试在此群集节点上启动资源是毫无意义的。最好是让 RGM 在其它节点上启动该资源。

但是请注意，前述声明假定 `svc_validate()` 十分保守，它仅检查应用程序绝对需要的群集节点上的资源。否则，资源可能无法在所有群集节点上启动，并因此进入 `START_FAILED` 状态。有关此状态的说明，请参见 `scswitch`(1M) 手册页和《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》。

如果资源在节点上成功启动，则 `svc_start()` 函数必须返回 0。如果启动函数遇到问题，则该函数必须返回非零值。此函数失败后，RGM 将尝试在其它群集节点上启动该资源。

要尽量利用 DSDL，`svc_start()` 函数可以调用 `scds_pmf_start()` 实用程序以启动进程监视器工具 (PMF) 下的应用程序。此实用程序还使用 PMF 的故障回调操作功能来检测进程故障。有关更多信息，请参见 `pmfadm`(1M) 手册页中 `-a` 操作参数的说明。

Stop 方法

RGM 将对群集节点调用资源类型实现的 `Stop` 回调方法，以停止该应用程序。`Stop` 方法的回调语义学需要以下因素：

- `Stop` 方法必须是**幂等**方法，因为即使 `Start` 方法在节点上没有成功完成，`Stop` 方法也可以被 RGM 调用。因此，即使应用程序当前没有在群集节点上运行并且没有需要它执行的工作，`Stop` 方法也必须成功（退出零）。
- 如果资源类型的 `Stop` 方法在群集节点上失败（退出非零值），则正被停止的资源将进入 `STOP_FAILED` 状态。根据资源上的 `Failover_mode` 设置，这种情况可能会导致 RGM 执行群集节点的硬重启。

因此，必须设计 Stop 方法让此方法一定要停止应用程序。如果采用其他方法无法终止应用程序，您甚至可能需要采用 SIGKILL 突然中止应用程序。

您还必须确保此方法能够及时停止应用程序，因为框架将 Stop_timeout 属性的终止视为停止失败，并因而将资源置入 STOP_FAILED 状态。

DSDL 实用程序 scds_pmf_stop() 在第一次尝试使用 SIGTERM 软停止应用程序时应当足以用于大多数应用程序。此函数随后将 SIGKILL 传递给进程。此函数假定应用程序是使用 scds_pmf_start() 在 PMF 下启动的。有关此实用程序的详细信息，请参见第 182 页中的 “PMF 函数”。

假定停止应用程序的应用程序特定函数名为 svc_stop()，它采用以下代码实现了 Stop 方法：

```
if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR)
{
   return (1);   /* Initialization Error */
}
return (svc_stop(handle));
```

前述 svc_stop() 函数的实现与是否包括 scds_pmf_stop() 函数是不相关的。确定是否包括 scds_pmf_stop() 函数取决于应用程序是否通过 Start 方法在 PMF 下启动。

Stop 方法的实现中没有使用 svc_validate() 方法，因为即使系统当前遇到问题，Stop 方法也应当尝试在此节点上停止应用程序。

Monitor_start 方法

RGM 将调用 Monitor_start 方法以启动资源的故障监视器。故障监视器将监视正受该资源管理的应用程序的运行状况。资源类型实现通常将故障监视器作为单独在后台运行的守护进程实现。Monitor_start 回调方法用于使用正确的参数启动此守护进程。

由于监视器守护进程本身很容易失败（例如，该守护进程可能会停止，而使应用程序处于不受监视的状态），因此您应当使用 PMF 启动监视器守护进程。DSDL 实用程序 scds_pmf_start() 内置了对启动故障监视器的支持。此实用程序使用相对于 RT_basedir 的路径名来确定监视器守护进程的资源类型回调方法实现的位置。此实用程序使用由 DSDL 管理的 Monitor_retry_interval 和 Monitor_retry_count 扩展属性防止无限制地重新启动守护进程。

此实用程序还将为所有回调方法定义的相同命令行语法（即，-R *resource* -G *resource-group* -T *resource-type*）强加到监视器守护进程上，尽管监视器守护进程决不会被 RGM 直接调用。最后，此实用程序还允许监视器守护进程实现自身启用 scds_initialize() 实用程序来设置自己的环境。主要用于设计监视器守护进程本身方面。

Monitor_stop 方法

RGM 将调用 Monitor_stop 方法来停止通过使用 Monitor_start 方法启动的故障监视器守护进程。处理此回调方法失败与处理 Stop 方法失败的方式刚好相同。因此，Monitor_stop 方法必须是幂等的并且要像 Stop 方法一样强健。

如果您使用 scds_pmf_start() 公用程序启动故障监视器守护进程，则请使用 scds_pmf_stop() 公用程序停止该守护进程。

Monitor_check 方法

RGM 将在指定资源的节点上运行针对资源的 Monitor_check 回调方法以确定群集节点是否能够控制资源。换言之，RGM 运行此方法来确定正受资源管理的应用程序是否能够在节点上成功运行。

通常，这种情况涉及确保应用程序所需的所有系统资源在群集节点上确实可用。正如第 106 页中的 “Validate 方法”中所讨论的那样，实现的函数 svc_validate() 至少要确定该情况。

根据资源类型实现正在管理的特定应用程序，可以写入 Monitor_check 方法以执行其他任务。必须实现 Monitor_check 方法，这样此方法才能与同时运行的其他方法不发生冲突。如果使用的是 DSDL，则 Monitor_check 方法应该调用 svc_validate() 函数，该函数用于实现应用程序特定的资源属性验证。

Update 方法

RGM 将调用资源类型实现的 Update 方法将群集管理员所做的所有更改应用到活动资源的配置中。仅对资源当前处于联机状态的节点（如果有）调用 Update 方法。

必须保证资源类型实现可接受刚对资源配置所做的更改，因为 RGM 在运行 Update 方法之前先运行 Validate 方法。Validate 方法是在更改资源或资源组属性之前调用的，Validate 方法可以禁止提议更改。Update 方法在应用更改内容之后调用，以便有机会向活动（联机）资源通知这些新设置。

确定需要能够动态更新的属性时必须谨慎，并在 RTR 文件中用 TUNABLE = ANYTIME 标记那些属性。通常，您可以指定故障监视器守护进程使用的资源类型实现的哪些属性需要能够动态更新。但是，Update 方法的实现至少要重新启动监视器守护进程。

可以使用的可能属性包括：

- `Thorough_probe_interval`
- `Retry_count`
- `Retry_interval`
- `Monitor_retry_count`
- `Monitor_retry_interval`
- `Probe_timeout`

这些属性的影响范围包括：故障监视器守护进程检查服务的运行状况的方法、守护进程执行检查的频率、守护进程用于跟踪错误的历史时间间隔，以及由 PMF 设置的重新启动阈值。为了实现这些属性的更新，DSDL 中提供了实用程序 `scds_pmf_restart()`。

如果需要能够动态更新资源属性，但修改该属性可能会影响正在运行的应用程序，那么，您必须执行正确操作。必须确保对该属性的更新能够正确地应用到所有正在运行的应用程序实例。目前，您不能使用 DSDL 以这种方式动态更新资源属性。不能通过命令行将修改后的属性传递给 `Update`（而对于 `Validate` 则可以）。

`Init`、`Fini` 和 `Boot` 方法的说明

这些方法都是**一次性操作**方法，是由资源管理 API 规范定义的。DSDL 中附带的实现样例并未说明这些方法的用法。不过，DSDL 中的所有工具也可用于这些方法，您应当会需要这些方法。通常，对于资源类型实现来说，`Init` 和 `Boot` 方法完全相同，都可以实现**一次性操作**。通常，`Fini` 方法将执行**撤销**操作，用于撤销 `Init` 或 `Boot` 方法的操作。

设计故障监视器守护进程

使用 DSDL 的资源类型实现通常都有一个执行以下职责的故障监视器守护进程：

- 定期监视被管理的应用程序的运行状况。监视器守护进程的这一特殊职责很大程度上取决于特殊应用程序，并可能由于资源类型的不同而大有不同。DSDL 包含一些内置实用程序函数，这些函数用于执行基于 TCP 的简单服务的运行状况检查。您可以使用这些实用程序实现使用基于 ASCII 的协议（例如 HTTP、NNTP、IMAP 和 POP3）的应用程序。
- 使用资源属性 `Retry_interval` 和 `Retry_count` 跟踪应用程序遇到的问题。应用程序完全失败时，故障监视器需要确定 PMF 操作脚本是否应当重新启动服务，或者应用程序失败是否累积得太快需要执行故障转移。DSDL 实用程序 `scds_fm_action()` 和 `scds_fm_sleep()` 可以帮助实现这种机制。

- 通常采取的措施包括重新启动应用程序或尝试故障转移包含的资源组。DSDL 实用程序 `scds_fm_action()` 实现了此算法。此实用程序将为此目的计算在过去的 `Retry_interval` 秒内当前累积的探测失败数。
- 更新资源状态以使应用程序的运行状况的状态可用于 `scstat` 命令以及群集管理 GUI。

设计 DSDL 实用程序的目的在于使故障监视器守护进程的主循环可由本节末尾的伪代码表示。

实现使用 DSDL 的故障监视器时，请牢记以下因素：

- `scds_fm_sleep()` 可以迅速地检测到应用程序进程停止，因为通过 PMF 通知应用程序进程的停止是异步进行的。因此，所需的故障检测时间明显减少，从而增加了服务的可用性。否则，故障监视器可能频繁地唤醒以检查服务的运行状况并查看应用程序进程是否已停止。
- 如果 RGM 拒绝尝试将使用 `scha_control` API 的服务故障转移，则 `scds_fm_action()` 将**重置**或忽略其当前的失败历史记录。此函数将重置其当前的失败历史记录，因为该历史记录已超出 `Retry_count`。如果监视器守护进程在下一次迭代中唤醒，但无法成功完成守护进程的运行状况检查，则监视器守护进程将重新尝试调用 `scha_control()` 函数。该调用可能被再次拒绝，因为导致它在上次迭代中被拒绝的情况仍然有效。重置历史记录可确保故障监视器在下次迭代时至少在本地尝试解决这种情况（例如，通过重新启动应用程序）。
- 在重新启动失败的情况下，`scds_fm_action()` 将**不**重置应用程序失败历史记录，因为通常在情况自身不进行修正时，您将立即发布 `scha_control()`。
- 实用程序 `scds_fm_action()` 将根据失败历史记录将资源状态更新为 `SCHA_RSSTATUS_OK`、`SCHA_RSSTATUS_DEGRADED` 或 `SCHA_RSSTATUS_FAULTED`。从而使此状态可用于群集系统管理。

在大多数情况下，您可以在独立的实用程序中（例如，`svc_probe()`）实现应用程序特定的运行状况检查操作。您可以将其集成到以下通用主循环中。

```
for (;;) {
   /* sleep for a duration of thorough_probe_interval between
   *  successive probes.
   */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));
   /* Now probe all ipaddress we use. Loop over
   * 1. All net resources we use.
   * 2. All ipaddresses in a given resource.
   * For each of the ipaddress that is probed,
   * compute the failure history.
   */
   probe_result = 0;
   /* Iterate through the all resources to get each
   * IP address to use for calling svc_probe()
   */
   for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
   /* Grab the hostname and port on which the
   * health has to be monitored.
   */
```

```
hostname = netaddr->netaddrs[ip].hostname;
port = netaddr->netaddrs[ip].port_proto.port;
/*
* HA-XFS supports only one port and
* hence obtaint the port value from the
* first entry in the array of ports.
*/
ht1 = gethrtime();
/* Latch probe start time */
probe_result = svc_probe(scds_handle, hostname, port, timeout);
/*
* Update service probe history,
* take action if necessary.
* Latch probe end time.
*/
ht2 = gethrtime();
/* Convert to milliseconds */
dt = (ulong_t)((ht2 - ht1) / 1e6);
/*
* Compute failure history and take
* action if needed
*/
(void) scds_fm_action(scds_handle,
probe_result, (long)dt);
}       /* Each net resource */
}       /* Keep probing forever */
```

第 8 章

DSDL 资源类型实现样例

本章介绍了使用数据服务开发库 (DSDL) 实现的资源类型样例 `SUNW.xfnts`。此数据服务是使用 C 语言编写的。底层应用程序为 X Font Server，它是基于 TCP/IP 的服务。附录 C 包含 `SUNW.xfnts` 资源类型中每个方法的完整代码。

本章包括以下主题：

X Font Server

X Font Server 是基于 TCP/IP 的服务，该服务向其客户机提供字体文件。客户机将连接该服务器，请求提供字体集，该服务器将从磁盘读取相应的字体文件并将它们提供给客户机。X Font Server 守护进程包含位于 `/usr/openwin/bin/xfs` 的服务器二进制文件。该守护进程通常通过 `inetd` 启动。但是，对于当前的样例，假定 `/etc/inetd.conf` 文件中的正确条目已被禁用（例如，通过使用 `fsadmin -d` 命令），以使守护进程处于 Sun Cluster 软件的单独控制下。

X Font Server 配置文件

默认情况下，X Font Server 从 `/usr/openwin/lib/X11/fontserver.cfg` 文件中读取其配置信息。此文件中的目录条目包含可用于守护进程提供服务的字体目录列表。群集管理员可以在群集文件系统中找到字体目录。通过在系统的字体数据库中维护一个单独的副本，此位置优化了 X Font Server 在 Sun Cluster 上的使用。如果群集管理员需要更改此位置，则必须编辑 `fontserver.cfg` 以反映新的字体目录路径。

为了简化配置，群集管理员还可以将配置文件本身置于群集文件系统中。`xfs` 守护进程提供了覆盖该文件默认的内置位置的命令行参数。`SUNW.xfnts` 资源类型使用以下命令启动在 Sun Cluster 软件控制下的守护进程。

`/usr/openwin/bin/xfs -config` *location-of-configuration-file*`/fontserver.cfg \`
`-port` *port-number*

在 `SUNW.xfnts` 资源类型实现中，可以使用 `Confdir_list` 属性来管理 `fontserver.cfg` 配置文件的位置。

TCP 端口号

`xfs` 服务器守护进程侦听的 TCP 端口号通常为 “fs” 端口，该端口在 `/etc/services` 文件中通常定义为 `7100`。但是，群集管理器附带的 `xfs` 命令的 `-port` 选项使群集管理器可以覆盖默认设置。

可以使用 `SUNW.xfnts` 资源类型中的 `Port_list` 属性设置默认值并使群集管理器可以将 `-port` 选项与 `xfs` 命令结合使用。在 RTR 文件中您需要将此属性的缺省值定义为 `7100/tcp`。在 `SUNW.xfnts Start` 方法中，可以将 `Port_list` 传递给 `xfs` 命令行上的 `-port` 选项。因此，此资源类型的用户无需指定端口号（默认端口为 `7100/tcp`）。群集管理员在配置资源类型时，可以指定其他 `Port_list` 属性值。

SUNW.xfnts RTR 文件

本节介绍了 `SUNW.xfnts` RTR 文件中的几个主要属性。本节并未介绍该文件中每个属性的作用，对于此类说明，请参见“第 31 页中的 “设置资源和资源类型属性””。

`Confdir_list` 扩展属性用于标识配置目录（或目录列表），如下所示：

```
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "The Configuration Directory Path(s)";
}
```

Confdir_list 属性未指定缺省值。创建资源时，群集管理员必须指定目录名称。该值以后不能更改，因为可调性被限制为 AT_CREATION。

Port_list 属性用于标识服务器守护进程进行侦听的端口，如下所示：

```
{
        PROPERTY = Port_list;
        DEFAULT = 7100/tcp;
        TUNABLE = ANYTIME;
}
```

由于属性将声明默认值，因此，群集管理员可以在创建资源时指定新值或接受默认值。任何人以后都不能更改该值，因为可调性被限制为 AT_CREATION。

函数和回调方法的命名约定

通过了解以下约定可以标识各种样例代码段：

- RMAPI 函数以 scha_ 开头。
- DSDL 函数以 scds_ 开头。
- 回调方法以 xfnts_ 开头。
- 用户编写的函数以 svc_ 开头。

scds_initialize() 函数

DSDL 要求每个回调方法在方法的开头都要调用 scds_initialize() 函数。此函数可执行以下操作：

- 检查和处理框架传递给数据服务方法的命令行参数（argc 和 argv）。方法无需处理任何附加命令行参数。
- 设置内部数据结构，以供 DSDL 中的其他函数使用。
- 初始化记录环境。
- 验证故障监视器探测程序设置。

使用 scds_close() 函数可以收回由 scds_initialize() 分配的资源。

xfnts_start 方法

当包含数据服务资源的资源组在群集节点上联机或启用资源时，RGM 将在该群集节点上运行 Start 方法。在资源类型样例 SUNW.xfnts 中，xfnts_start 方法将激活该节点上的 xfs 守护进程。

xfnts_start 方法将调用 scds_pmf_start() 以启动 PMF 下的守护进程。PMF 提供了自动失败通知和重新启动功能，并集成了故障监视器。

注 – xfnts_start 将首先调用执行一些必要的**内务处理**函数的 scds_initialize()。第 117 页中的 “scds_initialize() 函数” 和 scds_initialize(3HA) 手册页包含更多信息。

启动 X Font Server 之前验证服务

在 xfnts_start 方法尝试启动 X Font Server 之前，它将调用 svc_validate() 来验证正确配置是否已就位以支持 xfs 守护进程。

```
rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }
```

有关详细信息，请参见“第 130 页中的 “xfnts_validate 方法””。

使用 svc_start() 启动服务

xfnts_start 方法将调用在 xfnts.c 文件中定义的 svc_start() 方法来启动 xfs 守护进程。本节将介绍 svc_start ()。

用于启动 xfs 守护进程的命令如下：

```
# xfs -config config-directory/fontserver.cfg -port port-number
```

Confdir_list 扩展属性用于标识 *config-directory*，Port_list 系统属性用于标识 *port-number*。当群集管理员配置数据服务时，将提供这些属性的具体值。

xfnts_start 方法将这些属性声明为字符串数组。xfnts_start 方法使用 scds_get_ext_confdir_list() 和 scds_get_port_list() 函数获取由群集管理员设置的值。scds_property_functions(3HA) 手册页中介绍了这些函数。

```
scha_str_array_t *confdirs;
scds_port_list_t    *portlist;
```

```
scha_err_t   err;

   /* get the configuration directory from the confdir_list property */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /* obtain the port to be used by XFS from the Port_list property */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }
```

请注意，`confdirs` 变量指向数组的第一个元素 (`0`)。

`xfnts_start` 方法将使用 `sprintf()` 来设置 `xfs` 命令行的格式。

```
/* Construct the command to start the xfs daemon. */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);
```

请注意，输出被重定向到 `/dev/null` 以抑制由守护进程生成的消息。

`xfnts_start` 方法将 `xfs` 命令行传递给 `scds_pmf_start()` 以启动在 PMF 控制下的数据服务。

```
scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

   if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
   }
```

关于对 `scds_pmf_start()` 的调用，请注意以下几点：

- `SCDS_PMF_TYPE_SVC` 参数将程序标识为作为数据服务应用程序启动。此方法还可以启动故障监视器或某个其他类型的应用程序。
- `SCDS_PMF_SINGLE_INSTANCE` 参数将此标识为单实例资源。
- `cmd` 参数是先前生成的命令行。
- 最后一个参数 `-1` 用于指定子监视级别。`-1` 值指定 PMF 监视所有子进程和原始进程。

返回前，`svc_pmf_start()` 将释放为 `portlist` 结构分配的内存。

```
scds_free_port_list(portlist);
return (err);
```

从 svc_start() 返回

即使 svc_start() 成功返回，底层应用程序也可能无法启动。因此，svc_start() 必须探测应用程序以验证在返回成功消息前运行了此应用程序。探测还必须考虑到应用程序可能无法立即可用，因为它需要一些时间才能启动。svc_start() 方法将调用在 xfnts.c 文件中定义的 svc_wait() 以验证应用程序是否正在运行。

```
/* Wait for the service to start up fully */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Calling svc_wait to verify that service has started.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "Successfully started the service.");
   } else {
      scds_syslog(LOG_ERR, "Failed to start the service.");
   }
```

svc_wait() 函数将调用 scds_get_netaddr_list() 以获取探测应用程序所需的网络地址资源。

```
/* obtain the network resource to use for probing */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

svc_wait() 函数用于获取 Start_timeout 和 Stop_timeout 值。

```
svc_start_timeout = scds_get_rs_start_timeout(scds_handle)
   probe_timeout = scds_get_ext_probe_timeout(scds_handle)
```

为了计算启动服务器可能需要的时间，svc_wait() 将调用 scds_svc_wait() 并传递相当于 Start_timeout 值百分之三的超时值。svc_wait() 函数将调用 svc_probe() 函数来验证应用程序是否已启动。svc_probe() 方法用来建立指定端口上服务器的简单套接字连接。如果无法连接端口，svc_probe() 将返回一个值 100，该值表示完全失败。如果连接成功但从端口断开连接时失败，svc_probe() 将返回一个值 50。

svc_probe() 失败或部分失败时，svc_wait() 使用超时值 5 调用 scds_svc_wait()。scds_svc_wait() 方法将探测频率限制为每五秒钟一次。此方法也可用来计算尝试启动该服务的次数。如果在资源的 Retry_interval 属性指定的时间内，尝试次数超出了资源的 Retry_count 属性的值，scds_svc_wait() 函数将返回失败。在这种情况下，svc_start() 函数也返回失败。

```
#define    SVC_CONNECT_TIMEOUT_PCT    95
#define    SVC_WAIT_PCT       3
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

   do {
      /*
       * probe the data service on the IP address of the
       * network resource and the portname
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Success. Free up resources and return */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

       /* Call scds_svc_wait() so that if service fails too
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }

   /* Rely on RGM to timeout and terminate the program */
   } while (1);
```

注 – 退出前，xfnts_start 方法将调用 scds_close() 来收回由 scds_initialize() 分配的资源。第 117 页中的 “scds_initialize() 函数” 和 scds_close(3HA) 手册页包含更多信息。

---

xfnts_stop 方法

由于 xfnts_start 方法使用 scds_pmf_start() 启动 PMF 下的服务，因此，xfnts_stop 将使用 scds_pmf_stop() 来停止该服务。

注 – xfnts_stop 将首先调用执行一些必要的**内务处理**函数的 scds_initialize()。第 117 页中的 “scds_initialize() 函数” 和 scds_initialize(3HA) 手册页包含更多信息。

---

xfnts_stop 方法将调用在 xfnts.c 文件中定义的 svc_stop() 方法，如下所示：

```
scds_syslog(LOG_ERR, "Issuing a stop request.");
   err = scds_pmf_stop(scds_handle,
       SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
       scds_get_rs_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Successfully stopped HA-XFS.");
   return (SCHA_ERR_NOERR); /* Successfully stopped */
```

关于在 svc_stop() 中对 scds_pmf_stop() 函数的调用，请注意以下几点：

- SCDS_PMF_TYPE_SVC 参数将程序标识为作为数据服务应用程序停止。此方法还可以停止故障监视器或某个其他类型的应用程序。
- SCDS_PMF_SINGLE_INSTANCE 参数用于标识信号。
- SIGTERM 参数将标识用于停止资源实例的信号。如果此信号无法停止该实例，scds_pmf_stop() 将发送 SIGKILL 来停止该实例，如果仍然失败，则会返回超时错误。有关详细信息，请参见 scds_pmf_stop(3HA) 手册页。
- 超时值即该资源的 Stop_timeout 属性的值。

注 – 退出前，xfnts_stop 方法将调用 scds_close() 来收回由 scds_initialize() 分配的资源。第 117 页中的 “scds_initialize() 函数” 和 scds_close(3HA) 手册页包含更多信息。

---

xfnts_monitor_start 方法

在节点上启动资源后，RGM 将在该节点上调用 Monitor_start 方法以启动故障监视器。xfnts_monitor_start 方法将使用 scds_pmf_start() 来启动 PMF 下的监视器守护进程。

注 – xfnts_monitor_start 将首先调用执行一些必要的**内务处理**函数的 scds_initialize()。第 117 页中的 “scds_initialize() 函数” 和 scds_initialize(3HA) 手册页包含更多信息。

---

xfnts_monitor_start 方法用于调用在 xfnts.c 文件中定义的 mon_start 方法，如下所示：

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling Monitor_start method for resource <%s>.",
      scds_get_resource_name(scds_handle));

    /* Call scds_pmf_start and pass the name of the probe. */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Started the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully started Monitor */
}
```

关于在 svc_mon_start() 中对 scds_pmf_start() 函数的调用，请注意以下几点：

- SCDS_PMF_TYPE_MON 参数标识程序作为故障监视器启动。此方法还可以启动数据服务或某个其他类型的应用程序。
- SCDS_PMF_SINGLE_INSTANCE 参数将此标识为单实例资源。
- xfnts_probe 参数用于标识要启动的监视器守护进程。假定监视器守护进程与其他回调程序位于同一目录下。
- 最后一个参数 0 用于指定子监视级别。在这种情况下，此值将指定 PMF 仅监视监视器守护进程。

注 – 退出前，xfnts_monitor_start 方法将调用 scds_close() 以收回由 scds_initialize() 分配的资源。第 117 页中的 “scds_initialize() 函数”和 scds_close(3HA) 手册页包含更多信息。

xfnts_monitor_stop 方法

由于 xfnts_monitor_start 方法使用 scds_pmf_start() 来启动 PMF 下的监视器守护进程，xfnts_monitor_stop 将使用 scds_pmf_stop() 来停止监视器守护进程。

注 – xfnts_monitor_stop 将首先调用执行一些必要的**内务处理**函数的 scds_initialize()。第 117 页中的 “scds_initialize() 函数”和 scds_initialize(3HA) 手册页包含更多信息。

xfnts_monitor_stop 方法用于调用在 xfnts.c 文件中定义的 mon_stop() 方法，如下所示：

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");

   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully stopped monitor */
}
```

关于在 svc_mon_stop() 中对 scds_pmf_stop() 函数的调用，请注意以下几点：

- SCDS_PMF_TYPE_MON 参数用于标识程序应作为故障监视器停止。此方法还可以停止数据服务或某个其他类型的应用程序。
- SCDS_PMF_SINGLE_INSTANCE 参数将此标识为单实例资源。
- SIGKILL 参数将标识用于停止资源实例的信号。如果此信号无法停止该实例，则 scds_pmf_stop() 将会返回超时错误。有关详细信息，请参见 scds_pmf_stop(3HA) 手册页。

- 超时值是资源的 Monitor_stop_timeout 属性的值。

注 – 退出前，xfnts_monitor_stop 方法将调用 scds_close() 以收回由 scds_initialize() 分配的资源。第 117 页中的 “scds_initialize() 函数” 和 scds_close(3HA) 手册页包含更多信息。

---

xfnts_monitor_check 方法

只要故障监视器尝试将包含该资源的资源组故障转移到其他节点，RGM 就将调用 Monitor_check 方法。xfnts_monitor_check 方法将调用 svc_validate() 方法以验证正确配置是否已就位以支持 xfs 守护进程。有关详细信息，请参见“第 130 页中的 “xfnts_validate 方法””。xfnts_monitor_check 的代码如下：

```
/* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
       scds_syslog(LOG_ERR, "Failed to initialize the handle.");
       return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
        "monitor_check method "
        "was called and returned <%d>.", rc);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method run as part of monitor check */
   return (rc);
}
```

SUNW.xfnts 故障监视器

在节点上启动资源后，RGM 不直接调用 PROBE 方法，而是调用 Monitor_start 方法来启动监视器。xfnts_monitor_start 方法用于启动在 PMF 控制下的故障监视器。xfnts_monitor_stop 方法用于停止故障监视器。

SUNW.xfnts 故障监视器可以执行以下操作：

- 通过使用为检查基于 TCP 的简单服务专门设计的实用程序（例如 xfs），定期监视 xfs 服务器守护进程的运行状况。

- 跟踪应用程序在时间窗口内遇到的问题（使用 Retry_count 和 Retry_interval 属性），并在应用程序完全失败时决定是重新启动还是故障转移数据服务。scds_fm_action() 和 scds_fm_sleep() 函数为此跟踪和决定机制提供了内置支持。
- 使用 scds_fm_action() 实现故障转移或重新启动决策。
- 更新资源状态，并使资源状态可用于管理工具和 GUI。

xfonts_probe 主循环

xfonts_probe 方法用于实现循环。实现循环前，xfonts_probe 将执行以下操作：

- 检索 xfnts 资源的网络地址资源，如下所示：

```
/* Get the ip addresses available for this resource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

- 调用 scds_fm_sleep() 并将 Thorough_probe_interval 的值作为超时值传递。探测将根据探测之间的 Thorough_probe_interval 值进行休眠，如下所示：

```
timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {
      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));
```

xfnts_probe 方法用于实现以下循环：

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
```

```
            * HA-XFS supports only one port and
            * hence obtain the port value from the
            * first entry in the array of ports.
            */
           ht1 = gethrtime(); /* Latch probe start time */
           scds_syslog(LOG_INFO, "Probing the service on port: %d.", port);

           probe_result =
           svc_probe(scds_handle, hostname, port, timeout);

           /*
            * Update service probe history,
            * take action if necessary.
            * Latch probe end time.
            */
           ht2 = gethrtime();

           /* Convert to milliseconds */
           dt = (ulong_t)((ht2 - ht1) / 1e6);

           /*
            * Compute failure history and take
            * action if needed
            */
           (void) scds_fm_action(scds_handle,
               probe_result, (long)dt);
       }   /* Each net resource */
   }    /* Keep probing forever */
```

svc_probe() 函数将实现探测程序逻辑。svc_probe() 的返回值被传递给 scds_fm_action()，后者用于确定是重新启动应用程序，是将资源组故障转移，还是不进行任何操作。

svc_probe() 函数

svc_probe() 函数通过调用 scds_fm_tcp_connect() 建立与指定端口的简单套接字连接。如果连接失败，svc_probe() 将返回一个值 100，该值表示完全失败。如果连接成功，但是断开连接失败，svc_probe() 将返回一个值 50，该值表示部分失败。如果连接和断开连接都成功，svc_probe() 将返回一个值 0，该值表示成功。

svc_probe() 的代码如下：

```
int svc_probe(scds_handle_t scds_handle,
char *hostname, int port, int timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
   char   testcmd[2048];
   int    time_used, time_remaining;
   time_t      connect_timeout;
```

```
/*
 * probe the data service by doing a socket connection to the port
 * specified in the port_list property to the host that is
 * serving the XFS data service. If the XFS service which is configured
 * to listen on the specified port, replies to the connection, then
 * the probe is successful. Else we will wait for a time period set
 * in probe_timeout property before concluding that the probe failed.
 */

/*
 * Use the SVC_CONNECT_TIMEOUT_PCT percentage of timeout
 * to connect to the port
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "Failed to connect to port <%d> of resource <%s>.",
       port, scds_get_resource_name(scds_handle));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Compute the actual time it took to connect. This should be less than
 * or equal to connect_timeout, the time allocated to connect.
 * If the connect uses all the time that is allocated for it,
 * then the remaining value from the probe_timeout that is passed to
 * this function will be used as disconnect timeout. Otherwise, the
 * the remaining time from the connect call will also be added to
 * the disconnect timeout.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Use the remaining time(timeout - time_took_to_connect) to disconnect
 */

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
```

```
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
       "%d seconds during connect operation and exceeded the "
       "timeout by %d seconds. Attempting disconnect with timeout"
       " %d ",
       connect_timeout,
       abs(time_used),
       SVC_DISCONNECT_TIMEOUT_SECONDS);

   time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Return partial failure in case of disconnection failure.
 * Reason: The connect call is successful, which means
 * the application is alive. A disconnection failure
 * could happen due to a hung application or heavy load.
 * If it is the later case, don't declare the application
 * as dead by returning complete failure. Instead, declare
 * it as partial failure. If this situation persists, the
 * disconnect call will fail again and the application will be
 * restarted.
 */
rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
if (rc != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "Failed to disconnect to port %d of resource %s.",
       port, scds_get_resource_name(scds_handle));
   /* this is a partial failure */
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

t2 = (hrtime_t)(gethrtime()/1E9);
time_used = (int)(t2 - t1);
time_remaining = timeout - time_used;

/*
 * If there is no time left, don't do the full test with
 * fsinfo. Return SCDS_PROBE_COMPLETE_FAILURE/2
 * instead. This will make sure that if this timeout
 * persists, server will be restarted.
 */
if (time_remaining <= 0) {
   scds_syslog(LOG_ERR, "Probe timed out.");
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

/*
 * The connection and disconnection to port is successful,
 * Run the fsinfo command to perform a full check of
 * server health.
 * Redirect stdout, otherwise the output from fsinfo
 * ends up on the console.
```

```
     */
    (void) sprintf(testcmd,
        "/usr/openwin/bin/fsinfo -server %s:%d > /dev/null",
        hostname, port);
    scds_syslog_debug(DBG_LEVEL_HIGH,
        "Checking the server status with %s.", testcmd);
    if (scds_timerun(scds_handle, testcmd, time_remaining,
       SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

       scds_syslog(LOG_ERR,
          "Failed to check server status with command <%s>",
          testcmd);
       return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }
    return (0);
}
```

完成时，`svc_probe()` 将返回一个值，来表示操作是成功 (`0`)、是部分失败 (`50`) 还是完全失败 (`100`)。`xfnts_probe` 方法将把此值传送到 `scds_fm_action()`。

确定故障监视器操作

`xfnts_probe` 方法将调用 `scds_fm_action()` 来确定要执行的操作。`scds_fm_action()` 的逻辑如下：

- 保留 `Retry_interval` 属性值内的累积失败历史记录。
- 如果累积失败次数达到 100 次（完全失败），则重新启动数据服务。如果超出 `Retry_interval` 的值，将重置该历史记录。
- 如果在 `Retry_interval` 指定的时间内，重新启动次数超出了 `Retry_count` 属性的值，则故障转移该数据服务。

例如，假定探测建立了与 `xfs` 服务器的连接，但断开连接时失败。这表明该服务器正在运行，但是可能处于挂起状态或恰好处于临时装入状态。断开连接失败将向 `scds_fm_action()` 发送部分失败 (`50`)。此值虽然小于用来重启该数据服务的阈值，但是它将保留在失败历史记录中。

如果在下次探测时断开服务器连接再次失败，值 `50` 将被添加到由 `scds_fm_action()` 维护的失败历史记录中。累积失败值现在为 `100`，因此 `scds_fm_action()` 将重新启动数据服务。

xfnts_validate 方法

创建资源时，以及群集管理员更新资源的属性或其包含组时，RGM 将调用 `Validate` 方法。在进行创建或更新之前，RGM 将调用 `Validate`，任何节点上该方法返回失败出口代码都将导致创建或更新操作取消。

仅当群集管理员更改资源或资源组属性，或当监视器设置 Status 和 Status_msg 资源属性时，RGM 才会调用 Validate。RGM 在设置属性时不调用 Validate。

注 – 只要 PROBE 方法尝试将数据服务故障转移到新节点上，Monitor_check 方法还将显式调用 Validate 方法。

RGM 将对传递给其他方法的那些正被更新的属性和值调用 Validate 和其他参数。在 xfnts_validate 的开头调用 scds_initialize() 将解析 RGM 传递给 xfnts_validate 所有参数和存储在 scds_handle 参数中的信息。xfnts_validate 调用的子例行程序将使用此信息。

xfnts_validate 方法将调用 svc_validate() 以验证以下条件：

- 已经为该资源设置了 Confdir_list 属性并定义了一个目录。

```
scha_str_array_t *confdirs;
   confdirs = scds_get_ext_confdir_list(scds_handle);

/* Return error if there is no confdir_list extension property */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }
```

- Confdir_list 所指定的目录包含 fontserver.cfg 文件。

```
(void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suppress lint error because errno.h prototype
       * is missing void arg
       */
      scds_syslog(LOG_ERR,
          "Failed to access file <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }
```

- 该服务器守护进程二进制文件可以在该群集节点上访问。

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
      scds_syslog(LOG_ERR,
          "Cannot access XFS binary : <%s> ", strerror(errno));
      return (1);
   }
```

- Port_list 属性将指定单个端口。

```
scds_port_list_t   *portlist;
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
```

```
            "Could not access property Port_list: %s.",
            scds_error_string(err));
        return (1); /* Validation Failure */
    }

#ifdef TEST
    if (portlist->num_ports != 1) {
        scds_syslog(LOG_ERR,
            "Property Port_list must have only one value.");
        scds_free_port_list(portlist);
        return (1); /* Validation Failure */
    }
#endif
```

- 包含数据服务的资源组还至少包含一个网络地址资源。

```
scds_net_resource_list_t *snrlp;
    if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
        != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "No network address resource in resource group: %s.",
            scds_error_string(err));
        return (1); /* Validation Failure */
    }

    /* Return an error if there are no network address resources */
    if (snrlp == NULL || snrlp->num_netresources == 0) {
        scds_syslog(LOG_ERR,
            "No network address resource in resource group.");
        rc = 1;
        goto finished;
    }
```

在返回之前，`svc_validate()` 将释放所有分配的资源。

```
finished:
    scds_free_net_list(snrlp);
    scds_free_port_list(portlist);

    return (rc); /* return result of validation */
```

注 – 退出前，`xfnts_validate` 方法将调用 `scds_close()` 以收回由 `scds_initialize()` 分配的资源。第 117 页中的 “`scds_initialize()` 函数”和 `scds_close`(3HA) 手册页包含更多信息。

---

xfnts_update 方法

RGM 将调用 Update 方法以通知正在运行的资源其属性已更改。唯一可为 xfnts 数据服务更改的属性与故障监视器有关。因此，只要更新了属性，xfnts_update 方法就将调用 scds_pmf_restart_fm() 以重新启动故障监视器。

```
/* check if the Fault monitor is already running and if so stop
 * and restart it. The second parameter to scds_pmf_restart_fm()
 * uniquely identifies the instance of the fault monitor that needs
 * to be restarted.
 */

scds_syslog(LOG_INFO, "Restarting the fault monitor.");
result = scds_pmf_restart_fm(scds_handle, 0);
if (result != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "Failed to restart fault monitor.");
   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);
   return (1);
}

scds_syslog(LOG_INFO,
"Completed successfully.");
```

注 – 如果故障监视器有多个实例，则 scds_pmf_restart_fm() 的第二个参数将唯一标识要被重新启动的故障监视器实例。示例中的值 0 用于指明故障监视器只有一个实例。

第 9 章

SunPlex Agent Builder

本章介绍了 SunPlex Agent Builder 和 Agent Builder 的 Cluster Agent 模块。两种工具均可以自动创建在资源组管理器 (Resource Group Manager, RGM) 的控制下运行的资源类型或数据服务。**资源类型**是包裹在应用程序外围的封装程序，使应用程序可以在 RGM 的控制下运行于群集环境中。

本章包括以下主题：

Agent Builder 概述

Agent Builder 提供了图形用户界面 (GUI)，用于指定关于要创建的应用程序和要创建资源类型的种类的信息。Agent Builder 支持网络可识别的应用程序和非网络可识别的应用程序。**网络可识别的应用程序**使用网络与客户机通信。**非网络可识别的应用程序**是独立的应用程序。

注 – 如果无法访问 Agent Builder 的 GUI 版本，可以通过命令行界面访问 Agent Builder。请参见第 150 页中的 “使用 Agent Builder 的命令行版本的方法”。

根据您指定的信息，Agent Builder 将生成以下软件：

- 一组用于故障转移的 C、Korn shell (`ksh`) 或通用数据服务 (GDS) 源文件或者与资源类型的方法回调对应的可伸缩资源类型。这些文件既可用于网络可识别（客户机/服务器模型）的应用程序又可用于非网络可识别（无客户机）的应用程序。

- 定制的资源类型注册 (RTR) 文件（如果您生成的是 C 或 Korn shell 源代码）。
- 用来启动、停止和删除资源类型的实例（资源）的定制实用程序脚本以及用来说明如何使用以上每一种文件的定制手册页。
- 包含二进制（如果您生成的是 C 源代码）、RTR 文件（如果您生成的是 C 或 Korn shell 源代码）和实用程序脚本的 Solaris 软件包。

使用 Agent Builder，您还可以为具有多个独立进程树（进程监视工具 [PMF] 必须对每个进程树分别进行监视和重新启动）的应用程序生成资源类型。

Agent Builder 使用前须知

使用 Agent Builder 之前，需要了解如何创建具有多个独立进程树的资源类型。

Agent Builder 可以为具有一个以上独立进程树的应用程序创建资源类型。说进程树是独立的是因为 PMF 分别监视和启动每个进程树。PMF 使用每个进程树自己的标记启动它们。

注 – 只有在指定生成的源代码为 C 或 GDS 时，才可以使用 Agent Builder 创建具有多个独立的进程树的资源类型。对于 Korn shell，不能使用 Agent Builder 创建上述资源类型。要针对 Korn shell 创建上述资源类型，必须手动写入代码。

在基本应用程序具有多个独立的进程树的情况下，不能通过仅指定一个命令行来启动该应用程序，而是必须创建一个文本文件，并在每一行中指定指向用来启动一个应用程序进程树的命令的完整路径。此文件不能包含空行。需要在 Agent Builder“配置”屏幕上的“启动命令”文本字段中指定此文本文件。

确保此文件不具有执行权限，从而使 Agent Builder 可以识别此文件。此文件的用途是从包含多个命令的单一可执行脚本启动多个进程树。如果此文本文件具有执行权限，则出现在群集上的资源不会带有任何问题或错误。但是，将在一个 PMF 标记下启动所有命令。结果是 PMF 不能分别监视和重新启动每个进程树。

使用 Agent Builder

本节介绍了如何使用 Agent Builder。此外，本节还包括使用 Agent Builder 之前您必须完成的任务。本节还介绍了在生成资源类型代码之后充分利用 Agent Builder 的方法。

包括以下主题：

分析应用程序

使用 Agent Builder 之前，必须确定要实现高可用性或可伸缩性的应用程序是否满足所需条件。Agent Builder 无法执行仅基于应用程序的运行时特性的分析。第 27 页中的 “分析应用程序的适用性” 提供了关于此主题的更多信息。

Agent Builder 并不总是能够为应用程序创建完整的资源类型。但是，多数情况下，Agent Builder 至少可以提供部分解决方案。例如，更复杂的应用程序可能需要 Agent Builder 在默认情况下不生成的其他代码。其他代码的示例包括添加其他属性的验证检查的代码，或调谐 Agent Builder 不显示的参数的代码。在这些情况下，您必须更改所生成的源代码或 RTR 文件。Agent Builder 正是提供了这种灵活性。

Agent Builder 将注释置于生成的源代码中的特定点处，您可以在此处添加您自己的资源类型代码。更改源代码之后，您可以使用 Agent Builder 生成的 make 程序的描述文件重新编译该源代码并重新生成资源类型软件包。

即使在整个编写资源类型代码的过程中都没有使用 Agent Builder 生成的任何代码，您也可以使用 Agent Builder 提供的 make 程序的描述文件和结构为您的资源类型创建 Solaris 软件包。

安装和配置 Agent Builder

Agent Builder 不需要专门进行安装。`SUNWscdev` 软件包中包含 Agent Builder，默认情况下，安装 Sun Cluster 软件时安装此软件包。《Sun Cluster 软件安装指南（适用于 Solaris OS）》中介绍了这方面的更多信息。

使用 Agent Builder 之前，请检验是否满足以下要求：

- `$PATH` 变量中包含 Java 运行时环境。Agent Builder 基于 Java 开发工具包（不低于 1.3.1 版）进行工作。如果 `$PATH` 变量中未包含 Java 开发工具包，Agent Builder 命令 (`scdsbuilder`) 将返回并显示错误消息。
- 已安装不低于 Solaris 8 OS 版本的开发者系统支持软件组。
- `$PATH` 变量中包含 `cc` 编译器。Agent Builder 使用 `$PATH` 变量中第一处 `cc` 来识别用于生成资源类型的 C 二进制代码的编译器。如果 `$PATH` 中未包含 `cc`，则 Agent Builder 将禁用生成 C 代码的选项。请参见第 142 页中的 “使用“创建”屏幕” 。

注 – 除了标准 `cc` 编译器，您还可以在 Agent Builder 中使用其他编译器。要使用其他编译器，请在 `$PATH` 中将符号链接 `cc` 更改为其他编译器的符号链接，例如 `gcc`。或者，将 make 程序的描述文件中的编译器定义（当前为 `CC=cc`）更改为其他编译器的完整路径。例如，在 Agent Builder 生成的 make 程序的描述文件中，将 `CC=cc` 更改为 `CC=`*pathname*`/gcc`。此种情况下，您不能直接运行 Agent Builder。而必须使用 `make` 和 `make pkg` 命令来生成数据服务代码和软件包。

Agent Builder 屏幕

Agent Builder 是一个包含两个步骤的向导，每个步骤都有对应的屏幕。Agent Builder 提供了以下两个屏幕，用来指导您完成创建新资源类型的过程：

1. **“创建”屏幕。**需要在此屏幕上提供关于要创建的资源类型的基本信息，例如其名称和生成的文件的工作目录。工作目录是您创建和配置资源类型模板的位置。还可以指定以下信息：

   - 要创建的资源种类（可伸缩或故障转移）
   - 基本应用程序是否为网络可识别（即，应用程序是否使用网络与其客户机进行通信）的应用程序
   - 要生成的代码的类型（C、Korn shell [`ksh`] 或 GDS）

   有关 GDS 的信息，请参见第 10 章。您必须在此屏幕上提供所有信息并选择“创建”以生成相应的输出。然后，您就可以看到“配置”屏幕。
2. **“配置”屏幕。**在此屏幕上，必须指定可以传送到任何 UNIX shell 以启动基本应用程序的完整命令行。您还可以提供停止和探测应用程序的命令（可选操作）。如果您尚未指定这两个命令，则生成的输出将使用信号来停止应用程序并提供默认探测机制。有关探测命令的说明，请参见第 144 页中的 “使用“配置”屏幕”。通过“配置”屏幕还可以更改以下三个命令中每个命令的超时值：启动、停止、探测。

启动 Agent Builder

注 – 如果无法访问 Agent Builder 的 GUI 版本，可以通过命令行界面访问 Agent Builder。请参见第 150 页中的 “使用 Agent Builder 的命令行版本的方法”。

如果从现有资源类型的工作目录启动 Agent Builder，则 Agent Builder 将以现有资源类型的值初始化“创建”和“配置”屏幕。

通过键入以下命令启动 Agent Builder：

```
% /usr/cluster/bin/scdsbuilder
```

将显示“创建”屏幕。

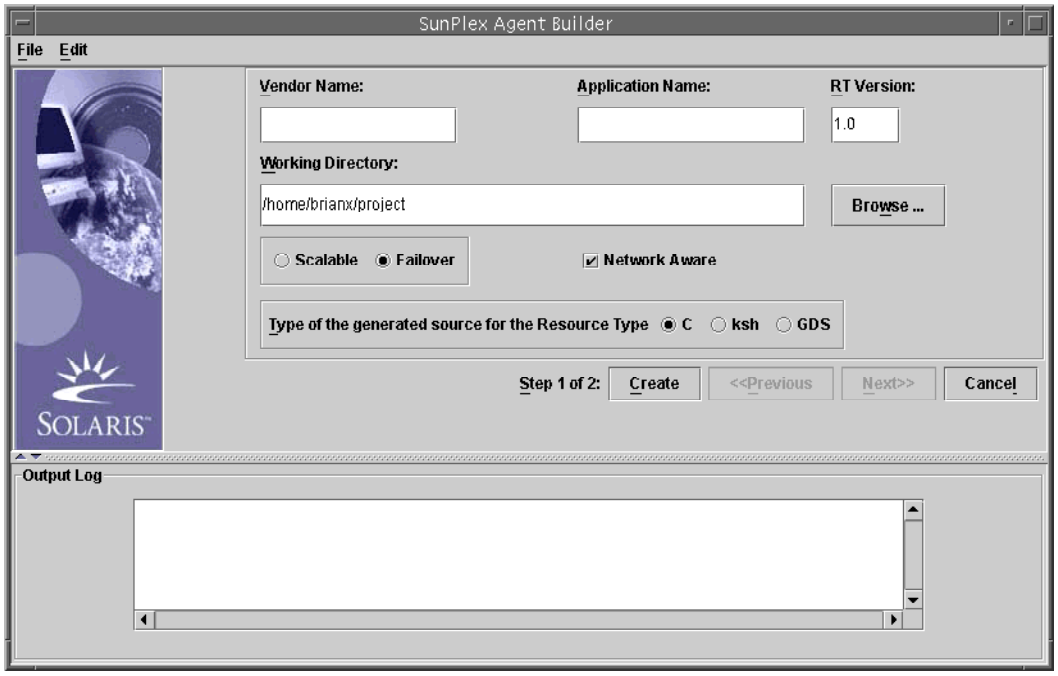


图 9–1 Agent Builder 的“创建”屏幕

浏览 Agent Builder

可以通过执行以下操作在“创建”和“配置”屏幕上输入信息：

- 在字段中键入信息
- 浏览目录结构并选择一个文件或目录
- 选择一组互斥的单选按钮中的一个单选按钮，例如选择“可伸缩”或“故障转移”
- 选择“网络可识别”复选框将基本应用程序标识为网络可识别应用程序，或不选择此复选框将其标识为非网络可识别应用程序。

使用每个屏幕底部的各个按钮，您可以分别完成结束任务、进入上一屏或下一屏或退出 Agent Builder 的操作。Agent Builder 将在必要时亮显或灰显这些按钮。

例如，完成“创建”屏幕上的各个字段并选择所需选项后，如果单击屏幕底部的“创建”按钮，则“上一步”和“下一步”按钮将灰显，因为不存在上一个屏幕，而且在您完成此步骤之前也不能进入下一步。

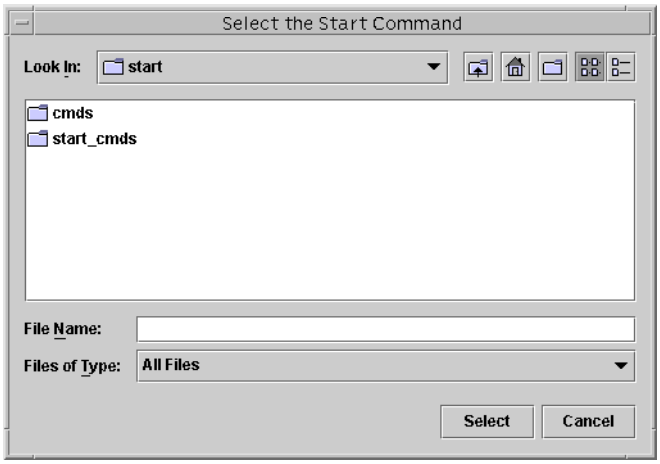
Agent Builder 将在屏幕底部的“输出记录”区域中显示进度消息。Agent Builder 完成此步骤后，它将显示成功消息或警告消息。将突出显示“下一步”，或如果此屏幕是最后一屏，则仅突出显示“取消”。

您可以随时单击“取消”以退出 Agent Builder。

“浏览”命令

您可以在某些 Agent Builder 字段中键入信息。而其他字段则需要单击“浏览”以浏览目录结构并选择文件或目录。

单击“浏览”时，将显示一个与以下屏幕相似的屏幕。



双击某个文件夹以打开该文件夹。当将光标移动到某个文件上时，该文件的名称将显示在“文件名”字段中。在找到所需文件并将光标移动到该文件上后，单击“选择”。

注 – 如果正在浏览以查找目录，请将光标移动到所需的目录上，然后单击“打开”。如果该目录不包含子目录，则 Agent Builder 将关闭该浏览窗口并将已将光标移至其上的目录的名称放置在相应字段中。如果该目录具有子目录，请单击“关闭”关闭浏览窗口并重新显示上一屏。Agent Builder 将把已将光标移至其上的目录的名称放置在相应字段中。

“浏览”屏幕右上角的图标的作用如下：

图标	目的
	此图标可使您在目录树中向上移动一级。
	此图标可使您返回主文件夹。
	此图标可用来在当前选定文件夹下创建新文件夹。
	此图标用于以后在不同视图之间进行切换。

Agent Builder 菜单

Agent Builder 提供了“文件”和“编辑”下拉式菜单。

Agent Builder “文件”菜单

“文件”菜单包含以下两个选项：

- **“装入资源类型”选项。**装入现有资源类型。Agent Builder 提供了浏览屏幕，您可以从中选择现有资源类型的工作目录。如果资源类型存在于启动 Agent Builder 的目录中，则 Agent Builder 将自动载入此资源类型。通过“装入资源类型”选项，可以从任何目录启动 Agent Builder，并选择现有资源类型以用作创建新资源类型的模板。请参见第 149 页中的 “重复使用使用 Agent Builder 创建的代码”。
- **“退出”选项。**退出 Agent Builder。您也可以单击“创建”或“配置”屏幕上的“取消”来退出 Agent Builder。

Agent Builder “编辑”菜单

“编辑”菜单包含以下两个选项：

- **“清除输出日志”选项。**清除输出日志中的信息。每当您选择“创建”或“配置”时，Agent Builder 都将向输出记录中附加状态消息。如果要不断地在 Agent Builder 中对源代码进行更改并重新生成输出，而且要隔离不同的状态消息，请在每次使用之前保存并清除日志文件。
- **“保存日志文件”选项。**将日志输出保存到文件中。Agent Builder 提供了浏览屏幕，通过该屏幕可以选择目录并指定文件名。

使用“创建”屏幕

创建资源类型的第一个步骤是完成“创建”屏幕（即启动 Agent Builder 后显示的屏幕）上的各项内容。下图显示键入了信息后的“创建”屏幕。

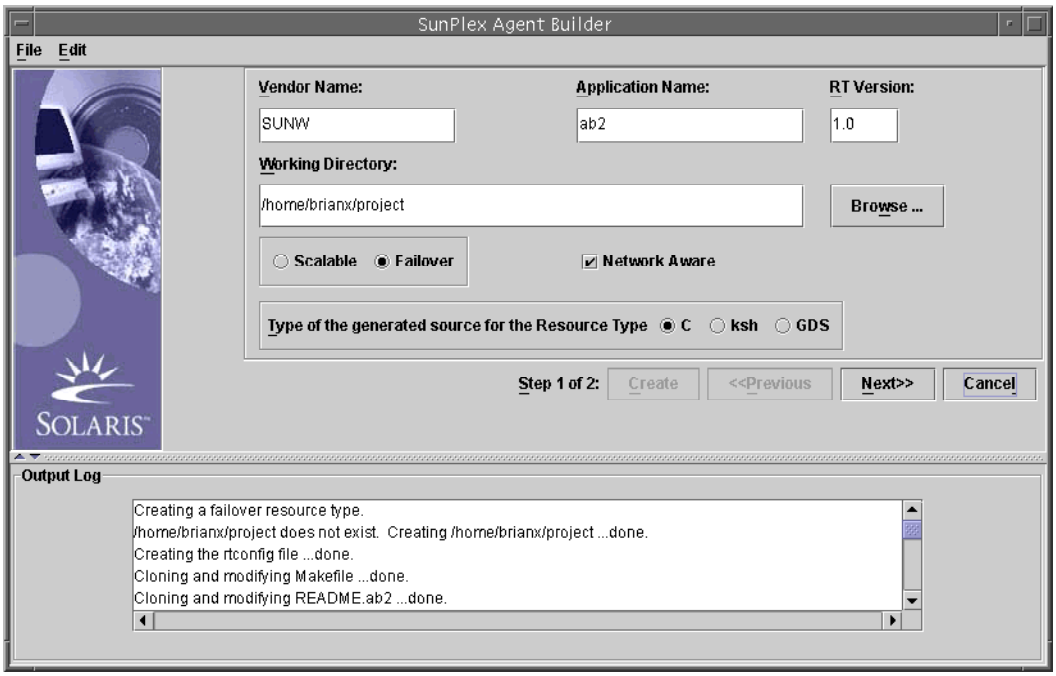


图 9–2 键入信息后的 Agent Builder“创建”屏幕

“创建”屏幕中包含以下字段、单选按钮和复选框：

- **“供应商名”。**用于标识资源类型供应商的名称。通常是指定供应商的库存符号。但是，任何能够唯一标识供应商的名称都是有效的。仅可使用字母数字字符。
- **“应用程序名”。**资源类型的名称。仅可使用字母数字字符。

注 – 供应商名称和应用程序名称合在一起组成资源类型的全名。从 Solaris 9 操作系统开始，供应商名称和应用程序名称的组合可以超过九个字符。但是，如果您使用的是早于 Solaris 9 的操作系统版本，则供应商名称和应用程序名称的组合不能超过九个字符。

- **“RT 版本”。**生成的资源类型的版本。RT 版本用于区分同一基本资源类型的多个已注册版本或升级版本。

  不能在“RT 版本”字段中使用以下字符：

  - 空格键
  - Tab
  - 斜杠 (/)
  - 反斜杠 (\)
  - 星号 (*)
  - 问号 (?)
  - 逗号 (,)
  - 分号 (;)
  - 左方括号 ([)
  - 右方括号 (])
- **“工作目录”。**Agent Builder 在该目录下创建一个目录结构，用于包含为目标资源类型创建的所有文件。在任意一个工作目录中仅能创建一个资源类型。Agent Builder 将此字段初始化为可以从中启动 Agent Builder 的目录的路径。您还可以键入其他名称或使用“浏览”找到其他目录。

  在工作目录下，Agent Builder 将使用资源类型的名称创建子目录。例如，如果 `SUNW` 是供应商名称，`ftp` 是应用程序名称，则 Agent Builder 将把此子目录命名为 `SUNWftp`。

  Agent Builder 将在此子目录下放置目录资源类型的所有目录和文件。请参见第 151 页中的 “Agent Builder 创建的目录结构”。
- **“可伸缩”或“故障转移”。**指定目标资源类型为“故障转移”或“可伸缩”。
- **“网络可识别”。**指定基本应用程序是否支持网路，即它是否使用网络与客户机进行通信。选择“网络可识别”复选框以指定网络可识别，或不选择该复选框以指定非网络可识别。
- **"C" 和 "ksh"。**指定生成的源代码所用的语言。尽管这些选项是互斥的，但是通过 Agent Builder，您可以使用 Korn shell 生成的代码创建资源类型，并再次使用相同的信息创建 C 生成的代码。请参见第 149 页中的 “重复使用使用 Agent Builder 创建的代码”。
- **"GDS"。**指定此服务为通用数据服务。第 10 章包含关于创建和配置通用数据服务的更多详细信息。

注 – 如果 `$PATH` 变量中不包含 `cc` 编译器，则 Agent Builder 将灰显 C 单选按钮并允许您选择 ksh 单选按钮。要指定其他编译器，请参见第 137 页中的 “安装和配置 Agent Builder” 结尾处的注释。

指定所需信息后，请单击“创建”。屏幕底部的“输出记录”区域用来显示 Agent Builder 执行的操作。您可以从“编辑”菜单中选择“保存输出日志”来保存输出日志中的信息。

完成操作后，Agent Builder 将显示一条成功消息或警告消息。

- 如果 Agent Builder 不能完成此步骤，请查看输出日志中的详细信息。
- 如果 Agent Builder 成功完成此步骤，请单击“下一步”以显示“配置”屏幕。通过“配置”屏幕，您可以完成资源类型的生成。

注 – 虽然生成完整资源类型的过程只有两步，但是您也可以在完成第一步（创建）之后退出 Agent Builder，您已指定的信息或 Agent Builder 已完成的工作并不会丢失。请参见第 149 页中的 “重复使用使用 Agent Builder 创建的代码”。

使用“配置”屏幕

Agent Builder 完成资源类型的创建后，如果您在“创建”屏幕上单击“下一步”，将显示如下图所示的“配置”屏幕。必须先创建资源类型才能访问“配置”屏幕。

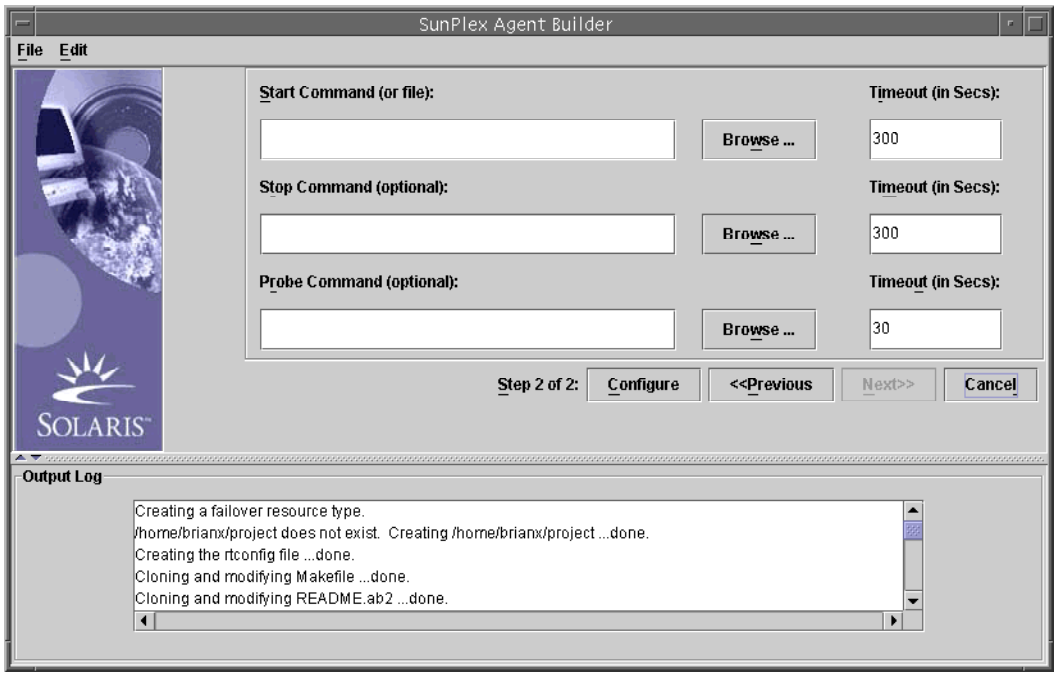


图 9–3 Agent Builder 的“配置”屏幕

“配置”屏幕上包含以下字段：

- **“启动命令”。**可以传送到任意 UNIX shell 以启动基本应用程序的完整命令行。必须指定启动命令。您可以在提供的字段中键入该命令，也可以使用“浏览”找到包含用来启动应用程序的命令的文件。

  完整的命令行中必须包含启动应用程序所需的所有内容，例如主机名、端口号、配置文件的路径。您还可以指定第 147 页中的 “使用属性变量”中介绍的属性变量。如果基于 Korn shell 的应用程序要求您在命令行上指定主机名，则可以使用 Agent Builder 定义的 `$hostnames` 变量。请参见第 146 页中的 “使用 Agent Builder 基于 Korn Shell 的 $hostnames 变量”。

  请不要将命令置于双引号 (“”) 之间。

注 – 如果基本应用程序具有多个独立的进程树（每一个进程树都在进程监视工具 (PMF) 控制下以其各自的标记来启动），则不能只指定一个命令。您必须创建包含多个单独命令（启动每个进程树）的文本文件，并在“启动命令”文本字段中指定此文件的路径。请参见第 136 页中的 “Agent Builder 使用前须知”。本节列出了此文件正确运行需要的一些特性。

- **“停止命令”。**可以传送到任意 UNIX shell 以停止基本应用程序的完整命令行。您可以在提供的字段中键入该命令，也可以使用“浏览”找到包含用来停止应用程序的命令的文件。您还可以指定第 147 页中的 “使用属性变量”中介绍的属性变量。如果基于 Korn shell 的应用程序要求您在命令行上指定主机名，则可以使用 Agent Builder 定义的 `$hostnames` 变量。请参见第 146 页中的 “使用 Agent Builder 基于 Korn Shell 的 $hostnames 变量”。

  此命令是可选的。如果不指定停止命令，则生成的代码将按以下步骤使用信号（包含在 `Stop` 方法中）来停止应用程序：

  - `Stop` 方法发送 `SIGTERM` 来停止应用程序，等待应用程序退出的时间长度为超时值的 80%。
  - 如果 `SIGTERM` 信号未成功，则 `Stop` 方法将发送 `SIGKILL` 来停止应用程序，等待应用程序退出的时间长度为超时值的 15%。
  - 如果 `SIGKILL` 不成功，则退出 `Stop` 方法（不成功）。余下的 5% 的超时值被认为是系统开销。

注意 – 请确保该停止命令在应用程序完全停止之后返回。

- **“探测命令”。**此命令是一个可定期运行的命令，用来检查应用程序的运行状况并返回介于 `0`（成功）和 `100`（完全失败）之间的退出状态值。此命令是可选的。您可以键入该命令的完整路径，也可以使用“浏览”找到包含用来探测应用程序的命令的文件。

  通常，您需指定基本应用程序的简单客户机。如果不指定探测命令，则生成的代码将只是连接至资源使用的端口或与其断开连接。如果这样的连接和断开连接成功，则生成的代码将声明应用程序运行状况良好。您还可以指定第 147 页中的 “使用属性变量”中介绍的属性变量。如果基于 Korn shell 的应用程序要求您在命令行上的探测命令中指定主机名，则可以使用 Agent Builder 定义的 `$hostnames` 变量。请参见第 146 页中的 “使用 Agent Builder 基于 Korn Shell 的 $hostnames 变量”。

  请不要将命令置于双引号 (””) 之间。
- **“超时”。**每个命令的超时值（以秒为单位）。可以指定新值，或者接受 Agent Builder 提供的默认值。启动命令和停止命令的默认值为 300 秒，探测命令的默认值为 30 秒。

使用 Agent Builder 基于 Korn Shell 的 `$hostnames` 变量

对于许多应用程序（特别是网络可识别的应用程序），必须通过命令行将应用程序在其上侦听和响应用户请求的主机名传送至该应用程序。多数情况下，主机名是必须在“配置”屏幕上为目标资源类型的启动、停止和探测命令指定的主机名参数。但是，应用程序侦听时所在的主机的主机名是特定于群集的。只有当资源在群集上运行时才能确定主机名。Agent Builder 生成资源类型代码时无法确定主机名。

为了解决此问题，Agent Builder 提供了 `$hostnames` 变量，您可以在命令行为启动、停止和探测命令指定此变量。

注 – $hostnames 变量只适用于基于 Korn shell 的服务。不支持对基于 C 和 GDS 的服务使用 $hostnames 变量。

指定 $hostnames 变量的方法就像指定实际主机名一样，例如：

```
% /opt/network_aware/echo_server -p port-no -l $hostnames
```

当目标资源类型的资源在群集上运行时，将用为资源配置的 LogicalHostname 或 SharedAddress 主机名替换 $hostnames 变量的值。为资源配置的主机名位于资源的 Network_resources_used 资源属性中。

如果配置的 Network_resources_used 属性具有多个主机名，则 $hostnames 变量将包含所有主机名，并用逗号将其隔开。

使用属性变量

您还可以通过使用属性变量，从 RGM 框架恢复选定 Sun Cluster 资源类型、资源和资源组属性的值。Agent Builder 将为属性变量扫描启动、探测或停止命令字符串，并在执行命令之前用这些值替换变量。

注 – 基于 Korn shell 的服务不支持使用属性变量。

属性变量列表

本节列出了可以使用的属性变量。附录 A中介绍了 Sun Cluster 的资源类型、资源和资源组属性。

资源属性变量

- HOSTNAMES
- RS_CHEAP_PROBE_INTERVAL
- RS_MONITOR_START_TIMEOUT
- RS_MONITOR_STOP_TIMEOUT
- RS_NAME
- RS_NUM_RESTARTS
- RS_RESOURCE_DEPENDENCIES
- RS_RESOURCE_DEPENDENCIES_WEAK
- RS_RETRY_COUNT
- RS_RETRY_INTERVAL
- RS_SCALABLE
- RS_START_TIMEOUT

- RS_STOP_TIMEOUT
- RS_THOROUGH_PROBE_INTERVAL
- SCHA_STATUS

资源类型属性变量

- RT_API_VERSION
- RT_BASEDIR
- RT_FAILOVER
- RT_INSTALLED_NODES
- RT_NAME
- RT_RT_VERSION
- RT_SINGLE_INSTANCE

资源组属性变量

- RG_DESIRED_PRIMARIES
- RG_GLOBAL_RESOURCES_USED
- RG_IMPLICIT_NETWORK_DEPENDENCIES
- RG_MAXIMUM_PRIMARIES
- RG_NAME
- RG_NODELIST
- RG_NUM_RESTARTS
- RG_PATHPREFIX
- RG_PINGPONG_INTERVAL
- RG_RESOURCE_LIST

属性变量的语法

您可以在属性名称前面加百分号 (%) 来表示属性变量，如下例所示：

```
/opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

根据上面的示例，Agent Builder 可以解释这些属性变量，并使用以下值启动 echo_server 脚本：

```
/opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

Agent Builder 替换属性变量的方式

Agent Builder 将按如下方式解释属性变量的类型：

- 将整型替换为其实际值（例如，300）。
- 将布尔型替换为字符串 TRUE 或 FALSE。
- 将字符串型替换为实际字符串（例如，phys-node-1）。

- 将字符串型列表替换为列表中的所有成员，每个字符串由逗号隔开（例如，`phys-node-1,phys-node-2,phys-node-3`）。
- 将整型列表替换为列表中的所有成员，每个整数由逗号隔开（`1,2,3`）。
- 枚举类型将按字符串格式被替换为该类型本身的值。

重复使用使用 Agent Builder 创建的代码

Agent Builder 使您可以通过以下方式重复使用已完成的工作：

- 您可以克隆使用 Agent Builder 创建的现有资源类型。
- 您可以编辑 Agent Builder 生成的源代码，并重新编译该代码来创建新的软件包。

▼ 克隆现有的资源类型的方法

请按照以下步骤克隆由 Agent Builder 生成的现有资源类型。

步骤 **1. 使用以下方法之一可将现有资源类型装入到 Agent Builder 中：**

- 在使用 Agent Builder 创建的现有资源类型的工作目录中启动 Agent Builder。确保工作目录中包含 `rtconfig` 文件。Agent Builder 将在“创建”和“配置”屏幕装入该资源类型的值。
- 使用“文件”下拉式菜单中的“装入资源类型”选项。

2. 在“创建”屏幕上更改工作目录。

必须使用“浏览”按钮来选择目录。仅键入新目录的名称是不够的。选择目录后，Agent Builder 将重新启用“创建”按钮。

3. 对现有资源类型进行所需更改。

您可能更改生成该资源类型的代码的类型。例如，如果您最初创建了资源类型的 Korn shell 版本，却发现超时并需要 C 版本，则可以进行以下操作：

- 装入现有的 Korn shell 资源类型。
- 将输出语言更改为 C。
- 单击“创建”以使 Agent Builder 生成该资源类型的 C 版本。

4. 创建克隆的资源类型。

a. 单击“创建”按钮创建该资源类型。

b. 单击“下一步”将显示“配置”屏幕。

c. 单击“配置”以配置资源类型，然后单击“取消”完成。

编辑已生成的源代码

为了简化创建资源类型的过程，Agent Builder 限制您可以指定的信息量，从而限制了生成的资源类型的范围。因此，要添加更多的复杂功能，您需要修改已生成的源代码或 RTR 文件。其他功能的示例包括添加其他属性的验证检查的代码，或调谐 Agent Builder 不显示的参数的代码。

源文件位于 *install-directory*/*rt-name*/src 目录中。Agent Builder 将注释嵌入在源代码中，您可以在这些注释中添加代码。这些注释的格式如下（针对 C 代码）：

```
/* User added code -- BEGIN vvvvvvvvvvvvvvv */
/* User added code -- END   ^^^^^^^^^^^^^^^ */
```

注 – 这些注释与 Korn shell 源代码中的相同，除了用注释标记 (#) 表示注释的开始。

例如，*rt-name*.h 声明了不同程序使用的所有实用程序函数。声明列表的结尾处是注释，通过这些注释您可以声明已添加到代码中的其他函数。

Agent Builder 还可以使用相应的目标在 *install_directory*/*rt_name*/src 目录中生成 make 程序的描述文件。使用 make 命令重新编译源代码。使用 make pkg 命令重新生成资源类型软件包。

RTR 文件位于 *install-directory*/*rt-name*/etc 目录中。您可以使用标准的文本编辑器编辑 RTR 文件。有关 RTR 文件的更多信息，请参见第 31 页中的 “设置资源和资源类型属性”。有关属性的信息，请参见附录 A。

▼ 使用 Agent Builder 的命令行版本的方法

Agent Builder 的命令行版本的使用过程与 GUI 相同。但是，与在 GUI 中键入信息不同，您需要把参数传送到 scdscreate 和 scdsconfig 命令。有关更多信息，请参见 scdscreate(1HA) 和 scdsconfig(1HA) 手册页。

请按照以下步骤使用 Agent Builder 的命令行版本。

步骤

1. **使用 `scdscreate` 来创建 Sun Cluster 资源类型模板，以便使应用程序具有高可用性和可伸缩性。**
2. **使用 `scdsconfig` 配置您用 `scdscreate` 创建的资源类型模板。**
   您可以指定属性变量。第 147 页中的 “使用属性变量”中介绍了属性变量。
3. **将目录更改为工作目录中的 `pkg` 子目录。**
4. **使用 `pkgadd` 命令来安装用 `scdscreate` 创建的软件包。**
   - **对于区环境中的 Solaris 10 OS，请做为全局区域中的全局管理员键入以下命令：**

     # **pkgadd -G -d .** *package-name*

如果软件包的内容不影响全局区域中与非全局区域共享的任何区域，则指定的软件包将被添加到全局区域中。

- **对于任何其他版本的 Solaris OS 或非区环境中的 Solaris 10 OS，请键入以下命令：**

  # **pkgadd -d .** *package-name*

5. **（可选的）编辑已生成的源代码**
6. **运行启动脚本。**

Agent Builder 创建的目录结构

Agent Builder 将创建一个目录结构，用于存储它为目标资源类型生成的所有文件。在“创建”屏幕上可以指定工作目录。必须为要开发的所有其他资源类型分别指定它们的安装目录。在工作目录下，Agent Builder 将创建一个子目录，其名称由供应商名称和资源类型名称串联而成。例如，如果您将 SUNW 指定为供应商的名称并创建了一个名为 ftp 的资源类型，则 Agent Builder 将在工作目录下创建一个名为 SUNWftp 的目录。

在此子目录下，Agent Builder 创建下表中列出的目录并在它们中分别放入应放入的内容。

目录名称	内容
bin	如果用 C 输出，则将包含编译源文件得到的二进制文件。对于 Korn shell 输出，包含的文件与 src 目录中的文件相同。
etc	包含 RTR 文件。Agent Builder 并置供应商名称和应用程序名称，两者之间用句点 (.) 进行分隔，从而构成 RTR 文件名。例如，如果供应商名为 SUNW 同时资源类型的名称为 ftp，则 RTR 文件的名称为 SUNW.ftp。
man	将包含 start、stop 和 remove 实用程序脚本的定制手册页，例如 startftp(1M)、stopftp(1M) 和 removeftp(1M)。 要查看这些手册页，请在指定相应路径时使用 man -M 选项。例如： **% man -M** *install-directory*/**SUNWftp/man removeftp**
pkg	包含最终的 Solaris 软件包，其中包括已创建的数据服务。
src	包含 Agent Builder 生成的源文件。
util	包含 Agent Builder 生成的 start、stop 和 remove 实用程序脚本。请参见第 153 页中的 “Sun Agent Builder 创建的实用程序脚本和手册页”。Agent Builder 将应用程序名称附加到以上各个脚本名称上；例如，startftp、stopftp 和 removeftp。

Agent Builder 的输出

源文件和二进制文件

用来管理群集上的资源组并最终管理资源的资源组管理器 (RGM)。RGM 基于回调模型进行工作。特定事件发生时（例如节点故障），RGM 将为在受影响的节点上运行的每个资源调用资源类型的方法。例如，RGM 将调用 `Stop` 方法来停止在受影响节点上运行的资源，然后再调用该资源的 `Start` 方法以在其它节点上启动该资源。有关此模型的更多信息，请参见第 20 页中的 “资源组管理器模型” 、第 23 页中的 “回调方法”和 `rt_callbacks`(1HA) 手册页。

为了支持该模型，Agent Builder 将在 *install-directory*`/`*rt-name*`/bin` 目录中生成八个可执行的 C 程序或 Korn shell 脚本。这些程序或 shell 脚本做为回调方法。

注 – 严格地说，实现故障监视器的 *rt-name*`_probe` 程序不是回调程序。RGM 并不直接调用 *rt-name*`_probe`，而是调用 *rt-name*`_monitor_start` 和 *rt-name*`_monitor_stop`。然后这两种方法通过调用 *rt-name*`_probe` 来启动和停止故障监视器。

下面是 Agent Builder 生成的八种方法：

- *rt-name*`_monitor_check`
- *rt-name*`_monitor_start`
- *rt-name*`_monitor_stop`
- *rt-name*`_probe`
- *rt-name*`_svc_start`
- *rt-name*`_svc_stop`
- *rt-name*`_update`
- *rt-name*`_validate`

有关每种方法的特定信息，请参见 `rt_callbacks`(1HA) 手册页。

在 *install-directory*`/`*rt-name*`/src` 目录（C 输出）中，Agent Builder 将生成以下文件：

- 头文件 (*rt-name*`.h`)
- 包含所有方法都通用的代码的源文件 (*rt-name*`.c`)
- 通用代码的对象文件 (*rt-name*`.o`)
- 每种方法的源文件 (`*.c`)
- 每种方法的对象文件 (`*.o`)

Agent Builder 将 *rt-name*`.o` 文件链接至每个方法 `.o` 文件以在 *install-directory*`/`*rt-name*`/bin` 目录中创建可执行文件。

对于 Korn shell 输出，*install-directory*/*rt-name*/bin 和 *install-directory*/*rt-name*/src 目录相同。每个目录都包含八个可执行脚本，分别对应七个回调方法和 Probe 方法。

注 – Korn shell 输出包括两个已编译的实用程序，gettime 和 gethostnames。特殊回调方法需要使用这些方法来获取时间和进行探测。

您可以编辑源代码并运行 make 命令来重新编译代码，并在完成时运行 make pkg 命令以生成新的软件包。为支持对源代码进行更改，Agent Builder 在源代码中的恰当位置嵌入了注释，您可以在这些注释中添加代码。请参见第 150 页中的 “编辑已生成的源代码”。

Sun Agent Builder 创建的实用程序脚本和手册页

生成资源类型并将其软件包安装到群集上之后，您还必须获取在群集上运行的该资源类型的实例（资源）。通常，可以使用管理命令或 SunPlex Manager 获取实例。但是为了方便起见，Agent Builder 将生成用于启动目标资源类型的资源的定制实用程序脚本，还将生成用于停止和删除该资源的脚本。位于 *install-directory*/*rt-name*/util 目录中的这三个脚本具有以下作用：

- **启动脚本。**注册资源类型和创建必要的资源组和资源。此脚本还可以创建网络地址资源（LogicalHostname 或 SharedAddress），使用该资源可以使应用程序与网络上的客户机进行通信。
- **停止脚本。** 停止资源。
- **删除脚本。**撤消启动脚本的工作。即，此脚本用来停止并删除系统中的资源、资源组和目标资源类型。

注 – 您只能使用与启动脚本启动的资源相对应的删除脚本，因为这些脚本使用了内部约定来命名资源和资源组。

Agent Builder 通过将应用程序名称附加到脚本名称来命名这些脚本。例如，如果应用程序名为 ftp，则脚本名分别为 startftp、stopftp 和 removeftp。

在 *install-directory*/*rt-name*/man/man1m 中，Agent Builder 为每个实用程序脚本都提供了手册页。启动这些脚本之前，应阅读这些手册页，由于它们记录了需要传送给脚本的参数。

要查看这些手册页，请通过使用 -M 选项和 man 命令来指定此 man 目录的路径。例如，如果 SUNW 为供应商，ftp 为应用程序名，则键入以下命令以查看 startftp(1M) 手册页：

```
% man -M install-directory/SUNWftp/man startftp
```

手册页公用程序脚本也可供群集管理员使用。如果 Agent Builder 生成的软件包已安装在群集上，则实用程序脚本的手册页将位于 `/opt/`*rt-name*`/man` 目录中。例如，键入以下命令可以查看 `startftp`(1M) 手册页：

```
% man -M /opt/SUNWftp/man startftp
```

Agent Builder 创建的支持文件

Agent Builder 将支持文件（例如 `pkginfo`、`postinstall`、`postremove` 和 `preremove`）置于 *install-directory*`/`*rt-name*`/etc` 目录中。此目录还包含资源类型注册 (RTR) 文件。RTR 文件将声明目标资源类型可用的资源和资源类型属性，并在群集中注册资源时初始化属性值。有关更多信息，请参见第 31 页中的 “设置资源和资源类型属性”。RTR 文件被命名为 *vendor-name*`.`*resource-type-name*，例如，`SUNW.ftp`。

您可以在不重新编译源代码的情况下使用标准的文本编辑器编辑此文件并进行更改，但是，您必须使用 `make pkg` 命令重建软件包。

Agent Builder 创建的软件包目录

install-directory`/`*rt-name*`/pkg` 目录包含 Solaris 软件包。软件包的名称是供应商名和应用程序名的串联，例如 `SUNWftp`。在 *install-directory*`/`*rt-name*`/src` 中的 make 程序的描述文件支持新软件包的创建。例如，如果您对源文件进行了更改并重新编译了其代码，或对软件包实用程序脚本进行了更改，则可以使用 `make pkg` 命令创建新软件包。

如果您尝试从多个节点同时运行 `pkgrm` 命令来从群集中删除一个软件包，则该命令将失败。您可以采用以下两种方法之一来解决此问题：

- 在群集的一个节点上运行 `remove` *rt-name* 脚本，然后在任意节点上运行 `pkgrm` 命令。
- 在群集的管理所有必要清除操作的那个节点上运行 `pkgrm` 命令。然后，在其余节点上同时（如果必要）运行 `pkgrm` 命令。

如果由于在多个节点上尝试同时运行 `pkgrm` 而使该命令失败，请在一个节点上再次运行该命令。然后，在其余节点运行该命令。

`rtconfig` 文件

如果在工作目录中生成 C 或 Korn shell 源代码，则 Agent Builder 将生成名为 `rtconfig` 的配置文件。此文件包含您在“创建”和“配置”屏幕上指定的信息。如果从现有资源类型的工作目录启动 Agent Builder，则 Agent Builder 将读取 `rtconfig` 文件。Agent Builder 将把您为现有资源类型提供的信息填充到“创建”和“配置”屏幕中。如果您通过从“文件”下拉式菜单中选择“装入资源类型”来装入现有资源类型，则 Agent Builder 将以相似的方式进行工作。当克隆现有资源类型时，此功能非常有用。请参见第 149 页中的 “重复使用使用 Agent Builder 创建的代码”。

Agent Builder 的 Cluster Agent 模块

Agent Builder 的 Cluster Agent 模块为 NetBeans™ 模块。此模块提供了 GUI，使用此 GUI 和 Sun Java Studio（以前的 Sun ONE Studio）产品可以为 Sun Cluster 软件创建资源类型。

注 – Sun Java Studio 文档中包含有关如何设置、安装和使用 Sun Java Studio 产品的信息。可以在 `http://wwws.sun.com/software/sundev/jde/documentation/index.html` Web 站点找到此文档。

▼ 安装和设置 Cluster Agent 模块的方法

安装 Sun Cluster 软件时会同时安装 Cluster Agent 模块。Sun Cluster 安装工具将 Cluster Agent 模块文件 `scdsbuilder.jar` 置于 `/usr/cluster/lib/scdsbuilder` 中。要将 Cluster Agent 模块与 Sun Java Studio 软件配套使用，您需要创建指向此文件的符号链接。

注 – 在要运行 Cluster Agent 模块的系统上，必须已安装 Sun Cluster 和 Sun Java Studio 产品以及 Java 1.4，并确保可用。

步骤 **1. 允许所有用户或只有您能够使用 Cluster Agent 模块。**

- 要使所有用户均能使用该模块，您需要成为超级用户或与此相当的身份，然后在全局模块目录中创建符号链接。

```
# cd /opt/s1studio/ee/modules
# ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

注 – 如果已将 Sun Java Studio 软件安装在 `/opt/s1studio/ee` 之外的目录，请用您所用目录的路径代替此目录路径。

- 如果希望只有您自己可以使用该模块，请在您的 `modules` 子目录中创建符号链接。

```
% cd ~your-home-dir/ffjuser40ee/modules
% ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

2. 停止并重新启动 Sun Java Studio 软件。

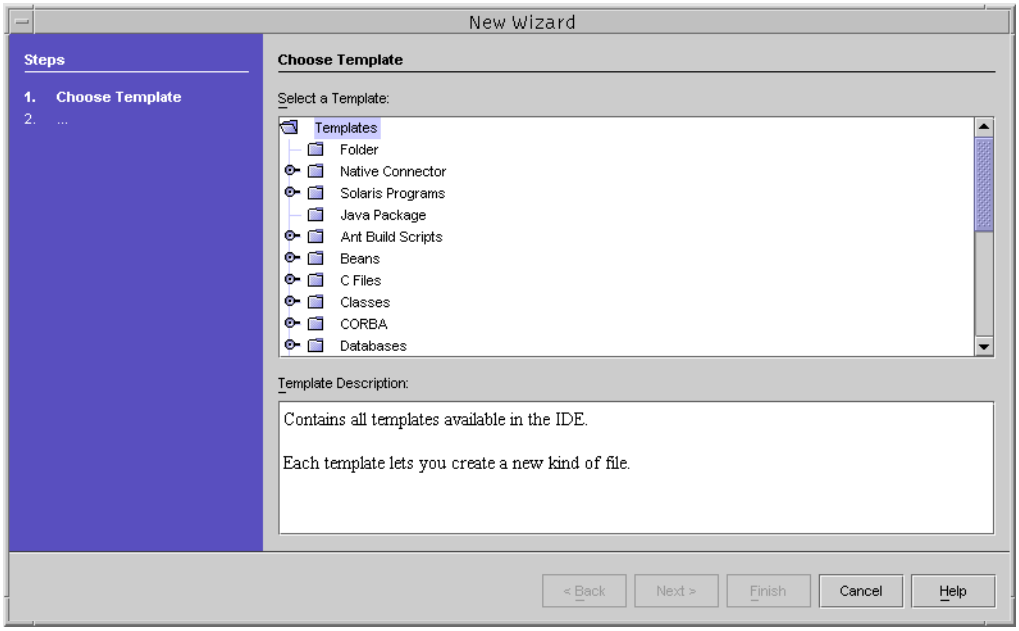
▼ 启动 Cluster Agent 模块的方法

以下各个步骤介绍了如何从 Sun Java Studio 软件启动 Cluster Agent 模块。

注 – Sun Java Studio 文档中包含有关如何设置、安装和使用 Sun Java Studio 产品的信息。可以在 `http://wwws.sun.com/software/sundev/jde/documentation/index.html` Web 站点找到此文档。

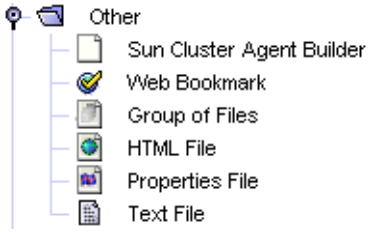
步骤 **1. 从 Sun Java Studio 的“文件”菜单中选择“新建”，或单击工具栏上相应的图标：**

将显示“新建向导”屏幕。



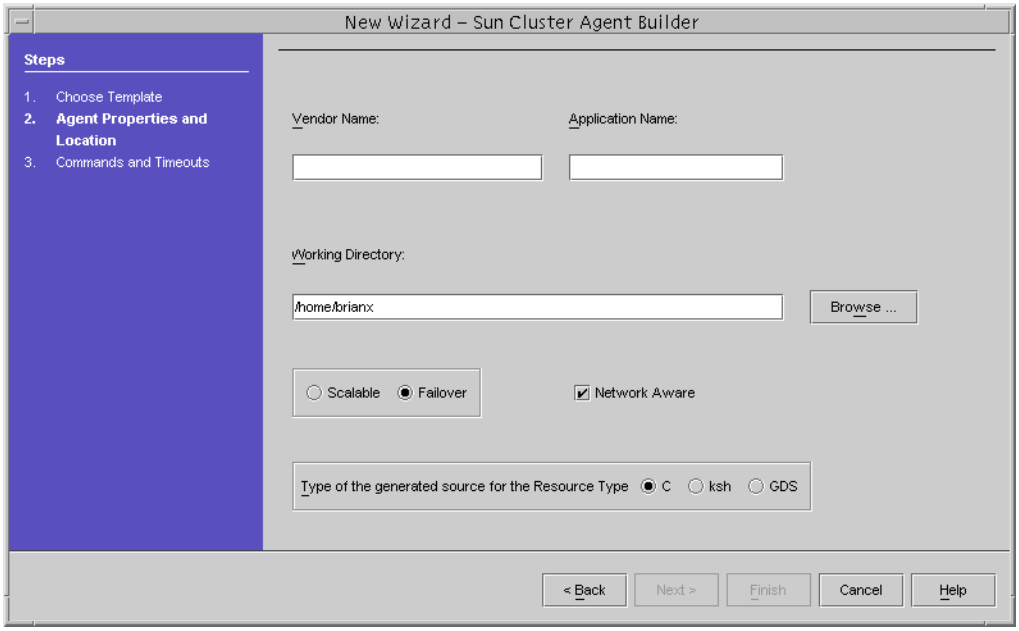
2. 在“选择模板”窗格中，向下滚动（如果必要）并单击“其他”文件夹旁边的钥匙符号。

将打开“其他”文件夹。



3. **从“其他”文件夹中选择 "Sun Cluster Agent Builder" 然后单击“下一步”。**

   Sun Java Studio 的 Cluster Agent 模块将启动。将显示第一个“新建向导 - Sun Cluster Agent Builder”屏幕。



使用 Cluster Agent 模块

Cluster Agent 模块的使用方法与 Agent Builder 软件的使用方法相同，而且它们的界面也是相同的。例如，以下各图分别显示了 Agent Builder 软件的“创建”屏幕和 Cluster Agent 模块中的第一个“新建向导 - Sun Cluster Agent Builder”屏幕，它们包含有相同的字段和选项。

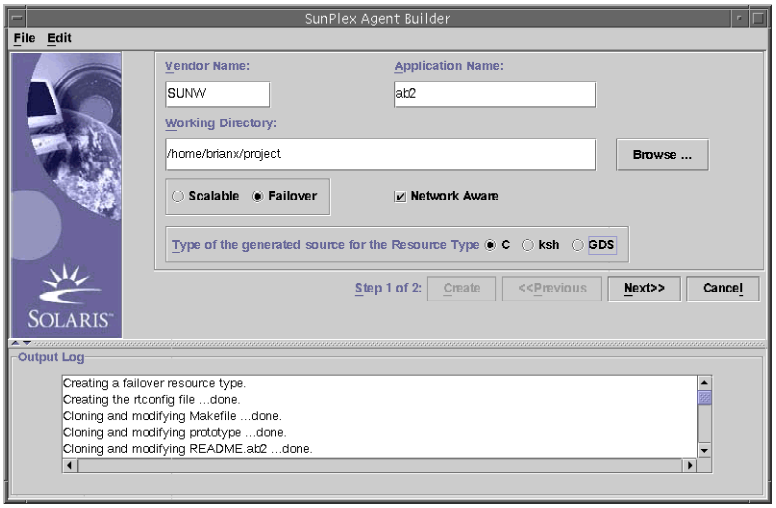


图 9–4 Agent Builder 软件中的“创建”屏幕

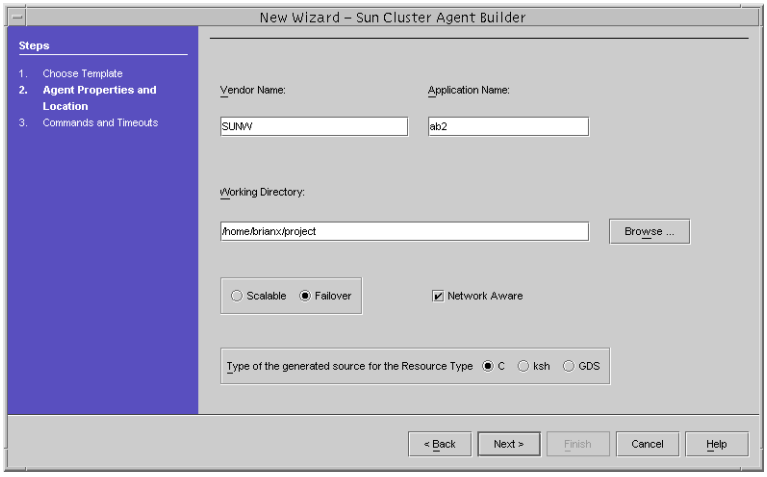


图 9–5 Cluster Agent 模块中的“新建向导 - Sun Cluster Agent Builder”屏幕

Cluster Agent 模块和 Agent Builder 之间的区别

虽然 Cluster Agent 模块和 Agent Builder 很相似，但是两者之间仍然存在微小的区别：

- 在 Cluster Agent 模块中，仅在您单击位于第二个“新建向导 - Sun Cluster Agent Builder”屏幕上的“完成”按钮之后才能创建和配置资源类型。当您单击位于第一个“新建向导 - Sun Cluster Agent Builder”屏幕上的“下一步”后，该资源类型实际上并**没有**创建。

  在 Agent Builder 中，在“创建”屏幕上单击“创建”时将立即创建资源类型。此外，在“配置”屏幕上单击“配置”时，将立即配置资源类型。
- 在 Sun Java Studio 产品中，显示在 Agent Builder 的“输出记录”区域中的信息将显示在一个单独的窗口中。

第 10 章

普通数据服务

本章提供有关普通数据服务 (GDS) 方面的信息和说明如何创建使用 GDS 的服务。您可以使用 SunPlex Agent Builder 或者 Sun Cluster 管理命令创建此服务。

本章包括以下主题：

普通数据服务概念

GDS 是一种机制，它通过将简单的网络可识别和非网络可识别应用程序插入 Sun Cluster 资源组管理 (RGM) 框架而使其具有高可用性或高可伸缩性。此机制无需对数据服务进行编码，而通常必须执行编码操作才能使应用程序具有高可用性或高可伸缩性。

GDS 是一项单独的、预编译的数据服务。您不能修改预编译的数据服务及其组件、回调方法 (`rt_callbacks`) 实现和资源类型注册文件 (`rt_reg`)。

本节包含以下主题：

- 预编译的资源类型
- 使用 GDS 的优点与不足
- 创建使用 GDS 的服务的方法
- GDS 记录事件的方式
- 必需的 GDS 属性
- 可选的 GDS 属性

预编译的资源类型

SUNWscgds 软件包中包含普通数据服务资源类型 SUNW.gds。scinstall 实用程序将在群集安装过程中安装此软件包。请参见 scinstall(1M) 手册页。SUNWscgds 软件包中包含下列文件：

```
# pkgchk -v SUNWscgds

/opt/SUNWscgds
/opt/SUNWscgds/bin
/opt/SUNWscgds/bin/gds_monitor_check
/opt/SUNWscgds/bin/gds_monitor_start
/opt/SUNWscgds/bin/gds_monitor_stop
/opt/SUNWscgds/bin/gds_probe
/opt/SUNWscgds/bin/gds_svc_start
/opt/SUNWscgds/bin/gds_svc_stop
/opt/SUNWscgds/bin/gds_update
/opt/SUNWscgds/bin/gds_validate
/opt/SUNWscgds/etc
/opt/SUNWscgds/etc/SUNW.gds
```

使用 GDS 的优点与不足

使用 GDS 与使用 Agent Builder 源代码（请参见 scdscreate(1HA) 手册页）或者 Sun Cluster 管理命令相比，具有以下优点：

- GDS 易于使用。
- GDS 及其方法都是预编译的，因此无法进行修改。
- 您可以使用 Agent Builder 为应用程序生成脚本。这些脚本都放在可以跨多个群集重复使用的 Solaris 软件包中。

虽然使用 GDS 有很多优点，但 GDS 机制**不能**用于以下实例：

- 所需的控制力度超出了使用预编译的资源类型所能达到的范围时（例如，需要添加扩展属性或者更改默认值时）
- 需要修改源代码以增加特殊功能时

创建使用 GDS 的服务的方法

创建使用 GDS 的服务的方法有两种：

- Agent Builder
- Sun Cluster 管理命令

GDS 和 Agent Builder

使用 Agent Builder 并选择 GDS 作为生成源代码的类型。使用用户输入的内容生成一组脚本，以便为给定的应用程序配置资源。

GDS 和 Sun Cluster 管理命令

此方法使用 SUNWscgds 中的预编译数据服务代码。但是，群集管理员必须使用 Sun Cluster 管理命令来创建和配置资源。请参见 scrgadm(1M) 和 scswitch(1M) 手册页。

选择创建基于 GDS 的服务时所应使用的方法

发布正确的 scrgadm 和 scswitch 命令需要键入大量内容。例如，请参见第 173 页中的 “如何使用 Sun Cluster 管理命令来创建使用 GDS 且具有高可用性的服务”和第 174 页中的 “如何使用 Sun Cluster 管理命令来创建使用 GDS 的可伸缩服务”。

将 GDS 与 Agent Builder 结合使用可以简化此进程，因为它将为您生成用于发布 scrgadm 和 scswitch 命令的脚本。

GDS 记录事件的方式

使用 GDS 可以记录 GDS 传递给 GDS 启动的脚本的相关信息。此信息包括启动、探测、停止方法以及属性变量的状态。您可以使用此信息诊断脚本中的问题或错误，或将其应用于其他目的。

使用第 166 页中的 “Log_level 属性”中介绍的 Log_level 属性来指定 GDS 要记录的消息级别或类型。可以指定 NONE、INFO 或 ERR。

GDS 日志文件

以下两个 GDS 日志文件放置于 /var/cluster/logs/DS/*resource-group-name*/*resource-name* 目录下：

- start_stop_log.txt，包含由资源启动和停止方法生成的消息
- probe_log.txt，包含由资源监视器生成的消息

以下示例显示了 start_stop_log.txt 包含的信息类型：

```
10/20/2005 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
10/20/2005 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

以下示例显示了 probe_log.txt 包含的信息类型：

```
10/20/2005 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
10/20/2005 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

必需的 GDS 属性

如果使用的是网络可识别应用程序，则必须提供 Start_command 扩展属性和 Port_list 属性。如果使用的是非网络可识别的应用程序，则只应提供 Start_command 扩展属性。

Start_command 扩展属性

您在 Start_command 扩展属性中指定的 start 命令将启动应用程序。此命令必须为可直接传递给 shell 启动应用程序的带参数的 UNIX 命令。

Port_list 属性

Port_list 属性将标识应用程序进行侦听的端口列表。Port_list 属性必须根据由 Agent Builder 创建的启动脚本指定，或是根据 scrgadm 命令指定（如果使用的是 Sun Cluster 管理命令）。

可选的 GDS 属性

可选的 GDS 属性包括**系统定义的属性**和**扩展属性**。系统定义的属性是 Sun Cluster 提供的一组标准属性。定义在 RTR 文件中的属性称为扩展属性。以下是可选的 GDS 属性：

- Network_resources_used 属性
- Stop_command 扩展属性
- Probe_command 扩展属性
- Start_timeout 属性
- Stop_timeout 属性
- Probe_timeout 扩展属性
- Child_mon_level 扩展属性（只与管理命令一起使用）
- Failover_enabled 扩展属性
- Stop_signal 扩展属性
- Log_level 扩展属性

Network_resources_used 属性

此属性的默认值为 Null。如果应用程序需要绑定到一个或多个特定的地址，则必须指定此属性。如果忽略此属性或将其指定为 Null，则应用程序会被认为是侦听所有地址。

创建 GDS 资源之前，必须已配置 LogicalHostname 或 SharedAddress 资源。有关如何配置 LogicalHostname 或 SharedAddress 资源的信息，请参见《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》。

要指定一个值，请指定一个或多个资源名称。每个资源名称可以包含一个或多个 LogicalHostname 资源或 SharedAddress 资源。有关详细信息，请参见 r_properties(5) 手册页。

Stop_command 属性

stop 命令必须停止应用程序并且只能在应用程序完全停止之后返回。此命令必须为可以直接传递给 shell 以停止应用程序的完整 UNIX 命令。

如果提供了 Stop_command 扩展属性，GDS 停止方法将在停止超时达到 80% 时启动 stop 命令。不管启动 stop 命令的输出结果是什么，GDS 停止方法都将在停止超时达到 15% 时发送 SIGKILL。剩余的 5% 的超时时间将用于内务处理开销。

如果忽略 stop 命令，GDS 将使用在 Stop_signal 中指定的信号尝试停止应用程序。

Probe_command 属性

probe 命令将定期检查给定应用程序的运行状况。此命令必须为可直接传递给 shell 以探测应用程序的带参数的 UNIX 命令。如果应用程序运行正常，probe 命令将以 0 退出状态返回。

probe 命令的退出状态用于确定应用程序失败的严重性。此退出状态（又称**探测状态**）必须是 0（表示成功）至 100（表示完全失败）之间的整数。探测状态也可以为特殊值 201，该值将导致应用程序立即故障转移，除非 Failover_enabled 设置为 FALSE。GDS 探测算法将使用探测状态决定是在本地重新启动应用程序还是将其故障转移。有关更多信息，请参见 scds_fm_action(3HA) 手册页。如果退出状态为 201，则应用程序将立即进行故障转移。

如果忽略 probe 命令，GDS 将提供它自己的简单探测。此探测将连接到源自 Network_resources_used 属性或 scds_get_netaddr_list() 函数的输出结果的 IP 地址集上的应用程序。有关更多信息，请参见 scds_get_netaddr_list(3HA) 手册页。如果连接成功，连接将立即断开。如果连接和断开连接均成功，则认为应用程序运行正常。

注 – GDS 提供的探测仅适于简单替换功能全面的应用程序特定的探测。

Start_timeout 属性

此属性用于指定 start 命令的启动超时。有关附加信息，请参见第 164 页中的“Start_command 扩展属性”。Start_timeout 的默认值为 300 秒。

Stop_timeout 属性

此属性用于指定 stop 命令的停止超时。有关附加信息，请参见第 164 页中的“Stop_command 属性”。Stop_timeout 的默认值为 300 秒。

Probe_timeout 属性

此属性用于指定 probe 命令的超时值。有关附加信息，请参见第 165 页中的“Probe_command 属性”。Probe_timeout 的默认值为 30 秒。

Child_mon_level 属性

注 – 如果使用 Sun Cluster 管理命令，则可以使用 Child_mon_level 属性。如果使用 Agent Builder，则不能使用此属性。

此属性对通过进程监视工具 (PMF) 监视的进程提供控制。此属性指示要达到监视交叉子进程的级别。此属性的作用类似于 pmfadm 命令的 -C 参数。请参见 pmfadm(1M) 手册页。

如果忽略此属性或将其设置为默认值 -1，所产生的效果将与忽略 pmfadm 命令的 -C 选项相同。即，所有子进程以及它们的子进程都将被监视。

Failover_enabled 属性

此 Boolean 扩展属性用于控制资源的故障转移行为。如果将该扩展属性设置为 TRUE，则当应用程序在 Retry_interval 秒内重新启动的次数超出 Retry_count 次，此应用程序就将故障转移。

如果将此扩展属性设置为 FALSE，则当应用程序在 Retry_interval 秒内重新启动的次数超出 Retry_count 次，此应用程序将不重新启动或故障转移至其他节点上。

您可以使用此属性禁止应用程序资源启动资源组故障转移。此属性的默认值为 TRUE。

Stop_signal 属性

GDS 使用整数扩展属性值来确定用于通过 PMF 停止应用程序的信号。有关可以指定的整数值列表，请参见 signal(3HEAD) 手册页。默认值为 15 (SIGTERM)。

Log_level 属性

此属性用于指定 GDS 所记录的诊断消息的级别或类型。可以为此属性指定 NONE、INFO 或 ERR。如果指定为 NONE，则 GDS 将不会记录诊断消息。如果指定为 INFO，则只记录信息型消息。如果指定为 ERR，则只记录错误消息。缺省情况下，GDS 不记录诊断消息 (NONE)。

使用 Agent Builder 创建使用 GDS 的服务

可以使用 Agent Builder 创建使用 GDS 的服务。第 9 章中对 Agent Builder 进行了详细介绍。

创建和配置基于 GDS 的脚本

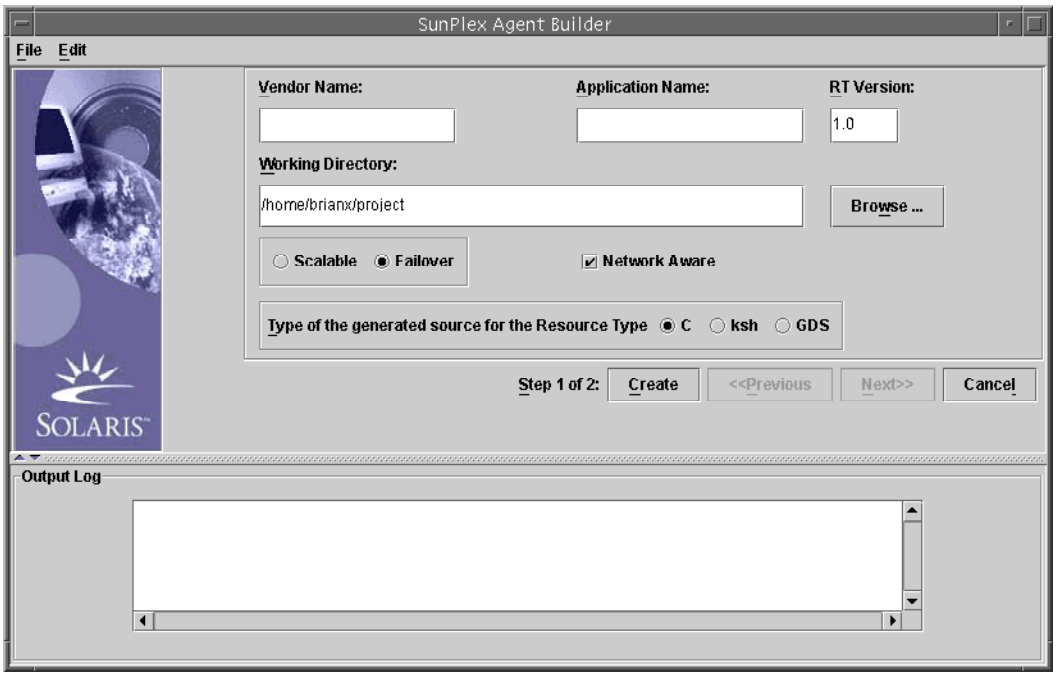
▼ 如何启动 Agent Builder 和创建脚本

步骤 **1. 成为超级用户或作为等效角色。**

2. 启动 Agent Builder。

```
# /usr/cluster/bin/scdsbuilder
```

3. 将显示 Agent Builder 的“创建”屏幕。



4. **键入供应商名称。**

5. **键入应用程序名称。**

注 – 自 Solaris 9 操作系统开始，供应商名称和应用程序名称的组合可超出九个字符。但是，如果使用的是早期版本的 Solaris 操作系统，则供应商名称和应用程序名称的组合不能超出九个字符。此组合用作脚本软件包的名称。

---

6. **转到工作目录。**
   可以使用“浏览”下拉式菜单选择目录而不是键入路径。

7. **选择该数据服务是可伸缩的，还是可进行故障转移的。**
   不必选择“网络可识别”，因为在创建 GDS 时该设置是默认值。

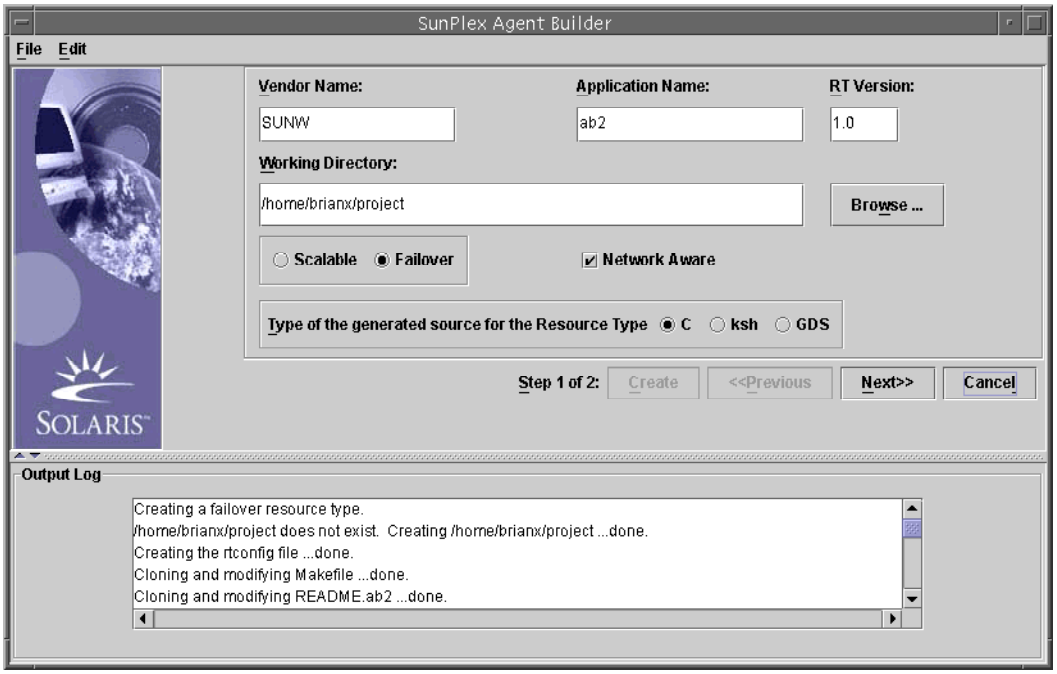
8. **选择“GDS”。**

9. **（可选的）更改 RT 版本的所示默认值。**

注 – 在“RT 版本”字段中不能使用以下字符：空格、制表符、斜杠 (/)、反斜杠 (\)、星号 (*)、问号 (?)、逗号 (,)、分号 (;)、左方括号 ([) 或右方括号 (])。

10. 单击“创建”。

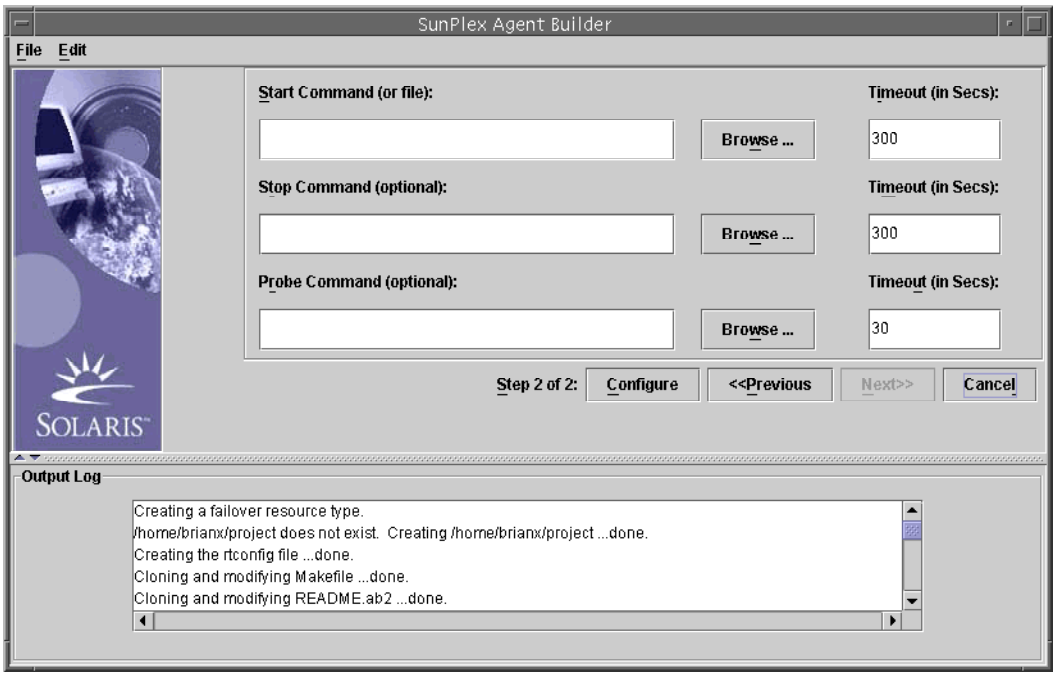
Agent Builder 将创建脚本。结果显示在“输出日志”区域中。



请注意，“创建”按钮将灰显。现在您就可以开始配置该脚本了。

11. 单击“下一步”。

将显示“配置”屏幕。



▼ 如何配置脚本

创建脚本后，需要配置新服务。

步骤 1. **键入 start 命令的位置，或单击“浏览”以找到 start 命令。**
您可以指定属性变量。第 147 页中的 “使用属性变量”介绍了属性变量。

2. **（可选的）键入 stop 命令的位置，或单击“浏览”以找到 stop 命令。**
您可以指定属性变量。第 147 页中的 “使用属性变量”介绍了属性变量。

3. **（可选的）键入 probe 命令的位置，或单击“浏览”以找到 probe 命令。**
您可以指定属性变量。第 147 页中的 “使用属性变量”介绍了属性变量。

4. **（可选的）为 start、stop 和 probe 命令指定新的超时值。**

5. **单击“配置”。**
Agent Builder 将配置脚本。

注 – Agent Builder 将供应商名称和应用程序名称组合在一起创建软件包名称。

将创建该脚本的软件包，并将其放置在以下目录中：

working-dir/*vendor-name-application*/pkg

例如，/export/wdir/NETapp/pkg。

6. **在群集的每个节点上，成为超级用户或假定一个等效的角色。**

7. **在群集的每个节点上，安装完整的软件包。**

   - **对于分区域环境中的 Solaris 10 OS，请以全局区域中的全局管理员身份键入以下命令：**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -G -d . NETapp
     ```

     指定的软件包将被添加到全局区域中，前提为软件包的内容不影响与非全局区域共享的全局区域的任何一部分。

     以下是 pkgadd 安装的文件：

     ```
     /opt/NETapp
     /opt/NETapp/README.app
     /opt/NETapp/man
     /opt/NETapp/man/man1m
     /opt/NETapp/man/man1m/removeapp.1m
     /opt/NETapp/man/man1m/startapp.1m
     /opt/NETapp/man/man1m/stopapp.1m
     /opt/NETapp/man/man1m/app_config.1m
     /opt/NETapp/util
     /opt/NETapp/util/removeapp
     /opt/NETapp/util/startapp
     /opt/NETapp/util/stopapp
     /opt/NETapp/util/app_config
     ```

   - **对于非区域环境中其他版本的 Solaris OS 或 Solaris 10 OS，请键入以下命令：**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -d . NETapp
     ```

     以下是 pkgadd 安装的文件：

     ```
     /opt/NETapp
     /opt/NETapp/README.app
     /opt/NETapp/man
     /opt/NETapp/man/man1m
     /opt/NETapp/man/man1m/removeapp.1m
     /opt/NETapp/man/man1m/startapp.1m
     /opt/NETapp/man/man1m/stopapp.1m
     /opt/NETapp/man/man1m/app_config.1m
     /opt/NETapp/util
     /opt/NETapp/util/removeapp
     /opt/NETapp/util/startapp
     /opt/NETapp/util/stopapp
     /opt/NETapp/util/app_config
     ```

注 – 手册页和脚本的名称对应于先前在"创建"屏幕中键入的应用程序名称，放在脚本名称之后（例如 `startapp`）。

8. **在群集的一个节点上，配置资源并启动应用程序。**

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

`startapp` 脚本的参数视资源类型的不同（故障转移或可伸缩）而有所变化。

注 – 要确定需要键入的命令行，请检查自定义的手册页或运行 `startapp` 脚本而不带任何参数以显示用法语句。

若要查看手册页，需要指定手册页的路径。例如，要查看 `startapp`(1M) 手册页，请键入：

```
# man -M /opt/NETapp/man startapp
```

要显示用法语句，请键入：

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be
 specified. For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup-adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

Agent Builder 的输出

Agent Builder 将根据创建软件包时提供的输入生成三个脚本和一个配置文件。配置文件用于指定资源组和资源类型的名称。

脚本如下：

- **启动脚本**。配置资源并启动处于 RGM 控制下的应用程序。
- **停止脚本**。停止应用程序并关闭资源和资源组。
- **删除脚本**。删除由启动脚本创建的资源和资源组。

这些脚本具有相同的接口，其功能与 Agent Builder 为基于非 GDS 的数据服务生成的实用程序脚本相同。这些脚本都放在可以跨多个群集重复使用的 Solaris 软件包中。

您可以自定义配置文件为资源组或其他参数提供自定义的名称，这些名称通常是在 `scrgadm` 命令中作为参数出现。如果不自定义脚本，Agent Builder 将提供 `scrgadm` 参数的默认值。

使用 Sun Cluster 管理命令创建使用 GDS 的服务

本节介绍了如何将参数输入到 GDS 中。使用现有的 Sun Cluster 管理命令（例如 `scrgadm` 和 `scswitch`）维护和管理 GDS。

如果脚本提供了足够的功能，则不必使用本节中所示的低级管理命令。但是，如果需要对基于 GDS 的资源进行较详细的控制，则可以使用低级管理命令。这些命令均由脚本执行。

▼ 如何使用 Sun Cluster 管理命令来创建使用 GDS 且具有高可用性的服务

步骤

1. **成为超级用户或作为等效角色。**
2. **注册资源类型 `SUNW.gds`。**

```
# scrgadm -a -t SUNW.gds
```

3. **创建资源组，其中包含 `LogicalHostname` 资源和故障转移服务本身。**

```
# scrgadm -a -g haapp_rg
```

4. **为 `LogicalHostname` 资源创建资源。**

```
# scrgadm -a -L -g haapp_rs -l hhead
```

5. **为故障转移服务本身创建资源。**

```
# scrgadm -a -j haapp_rs -g haapp_rg -t SUNW.gds \
          -y Scalable=false -y Start_timeout=120 \
          -y Stop_timeout=120 -x Probe_timeout=120 \
          -y Port_list="2222/tcp" \
          -x Start_command="/export/ha/appctl/start" \
          -x Stop_command="/export/ha/appctl/stop" \
          -x Probe_command="/export/app/bin/probe" \
          -x Child_mon_level=0 -y Network_resources_used=hhead \
          -x Failover_enabled=TRUE -x Stop_signal=9
```

6. **将资源组 `haapp_rg` 联机。**

```
# scswitch -Z -g haapp_rg
```

▼ 如何使用 Sun Cluster 管理命令来创建使用 GDS 的可伸缩服务

步骤 1. **成为超级用户或作为等效角色。**

2. **注册资源类型 `SUNW.gds`。**

```
# scrgadm -a -t SUNW.gds
```

3. **为 `SharedAddress` 资源创建资源组。**

```
# scrgadm -a -g sa_rg
```

4. **在 `sa_rg` 上创建 `SharedAddress` 资源。**

```
# scrgadm -a -S -g sa_rg -l hhead
```

5. **为可伸缩服务创建资源组。**

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

6. **为可伸缩服务创建资源。**

```
# scrgadm -a -j app_rs -g app_rg -t SUNW.gds \
       -y Scalable=TRUE -y Start_timeout=120 \
       -y Stop_timeout=120 -x Probe_timeout=120 \
       -y Port_list="2222/tcp" \
       -x Start_command="/export/app/bin/start" \
       -x Stop_command="/export/app/bin/stop" \
       -x Probe_command="/export/app/bin/probe" \
       -x Child_mon_level=0 -y Network_resource_used=hhead \
       -x Failover_enabled=TRUE -x Stop_signal=9
```

7. **将包含网络资源的资源组联机。**

```
 # scswitch -Z -g sa_rg
```

8. **将资源组 `app_rg` 联机。**

```
# scswitch -Z -g app_rg
```

Agent Builder 的命令行界面

Agent Builder 集成了一个命令行界面，该命令行界面与 GUI 提供的功能相同。此界面由 `scdscreate` 和 `scdsconfig` 命令组成。请参见 `scdscreate`(1HA) 和 `scdsconfig`(1HA) 手册页。

▼ 如何使用命令行版本的 Agent Builder 创建使用 GDS 的服务

本节介绍了如何使用命令行界面执行第 167 页中的 “使用 Agent Builder 创建使用 GDS 的服务”中所示的同一组步骤。

步骤 **1. 成为超级用户或作为等效角色。**

2. 创建服务。

- 对于故障转移服务，请键入：

```
# scdscreate -g -V NET -T app -d /export/wdir
```

- 对于可伸缩服务，请键入：

```
# scdscreate -g -s -V NET -T app -d /export/wdir
```

注 – `-d` 参数是可选的。如果不指定此参数，则当前目录将成为工作目录。

---

3. 配置服务。

```
# scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" \
-m "/export/app/bin/probe" -d /export/wdir
```

您可以指定属性变量。第 147 页中的 “使用属性变量”介绍了属性变量。

注 – 仅 `start` 命令是必需的。所有其他选项和参数都是可选的。

---

4. 在群集的每个节点上，安装完整的软件包。

- **对于分区域环境中的 Solaris 10 OS，请以全局区域中的全局管理员身份键入以下命令：**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -G -d . NETapp
```

指定的软件包将被添加到全局区域中，前提为软件包的内容不影响与非全局区域共享的全局区域的任何一部分。

以下是 `pkgadd` 安装的文件：

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
```

```
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

- **对于非区域环境中其他版本的 Solaris OS 或 Solaris 10 OS，请键入以下命令：**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

以下是 pkgadd 安装的文件：

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

注 – 手册页和脚本的名称对应于先前在"创建"屏幕上键入的应用程序名称，放在脚本名称之后（例如 startapp）。

---

5. **在群集的一个节点上，配置资源并启动应用程序。**

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

startapp 脚本的参数视资源类型的不同（故障转移或可伸缩）而有所变化。

注 – 要确定需要键入的命令行，请检查自定义的手册页或运行 `startapp` 脚本而不带任何参数以显示用法语句。

若要查看手册页，需要指定手册页的路径。例如，要查看 `startapp`(1M) 手册页，请键入：

```
# man -M /opt/NETapp/man startapp
```

要显示用法语句，请键入：

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be specified.
 For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup/adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

第 11 章

DSDL API 函数

本章列举并简单介绍了数据服务开发库 (DSDL) API 函数。有关每一个 DSDL 函数的完整介绍，请参阅各个 3HA 手册页。DSDL 仅提供 C 接口。基于脚本的 DSDL 接口不可用。

本章包括以下主题：

通用函数

本节中的函数可提供各种功能。使用这些函数您可以执行以下操作：

- 初始化 DSDL 环境
- 检索资源类型、资源、资源组名称和扩展属性值
- 故障转移和重新启动资源组，以及重新启动资源
- 将错误字符串转换成错误消息
- 在超时值定义的时间之内执行命令

初始化函数

以下函数用于初始化调用方法：

- `scds_initialize`(3HA) – 分配资源和初始化 DSDL 环境。
- `scds_close`(3HA) – 释放由 `scds_initialize()` 分配的资源。

检索函数

以下函数用于检索关于资源类型、资源、资源组和扩展属性的信息：

- `scds_get_resource_type_name`(3HA) – 检索用于调用程序的资源类型的名称。
- `scds_get_resource_name`(3HA) – 检索用于调用程序的资源的名称。
- `scds_get_resource_group_name`(3HA) – 检索用于调用程序的资源组的名称。
- `scds_get_ext_property`(3HA) – 检索指定扩展属性的值。
- `scds_free_ext_property`(3HA) – 释放由 `scds_get_ext_property()` 分配的内存。

以下函数用于检索关于资源所使用的 `SUNW.HAStoragePlus` 资源的状态信息：

`scds_hasp_check`(3HA) – 检索关于资源所使用的 `SUNW.HAStoragePlus` 资源的状态信息。此信息来自资源所依赖的所有 `SUNW.HAStoragePlus` 资源的状态（联机或脱机），是通过使用为该资源定义的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系统属性获得的。有关更多信息，请参见 `SUNW.HAStoragePlus`(5) 手册页。

故障转移和重新启动函数

以下函数用来进行故障转移或重新启动资源或资源组：

- `scds_failover_rg`(3HA) – 故障转移资源组。
- `scds_restart_rg`(3HA) – 重新启动资源组。
- `scds_restart_resource`(3HA) – 重新启动资源。

执行函数

以下函数在超时值定义的时间内执行命令，并将错误代码转换成错误消息：

- `scds_timerun`(3HA) – 在超时值定义的时间内执行命令。
- `scds_error_string`(3HA) – 将错误代码翻译为错误字符串。

属性函数

这些函数提供了公用 API，可用于访问相关资源类型、资源和资源组的具体属性，其中包括一些常用的扩展属性。DSDL 提供了 `scds_initialize()` 函数以解析命令行参数。该库用于**高速缓存**相关资源类型、资源和资源组的各种属性。

`scds_property_functions`(3HA) 手册页介绍了这些函数，包括：

- `scds_get_rt_`*property-name*

- `scds_get_rs_`*property-name*
- `scds_get_rg_`*property-name*
- `scds_get_ext_`*property-name*

网络资源访问函数

本节列出的函数用于检索、打印和释放由资源和资源组使用的网络资源。本节中的 `scds_get_` 函数提供了一种便捷的检索网络资源的方法，该方法无需使用 RMAPI 函数即可查询具体属性（例如 `Network_resources_used` 和 `Port_list`）。`scds_print_`*name*`()` 函数用于打印来自 `scds_get_`*name* `()` 函数所返回的数据结构的值。`scds_free_`*name* `()` 函数用于释放 `scds_get_` *name*`()` 函数所分配的内存。

主机名函数

以下函数用于处理主机名：

- `scds_get_rs_hostnames`(3HA) – 检索由资源使用的主机名列表。
- `scds_get_rg_hostnames`(3HA) – 检索由资源组中的网络资源使用的主机名列表。
- `scds_print_net_list`(3HA) – 打印由 `scds_get_rs_hostnames()` 函数或 `scds_get_rg_hostnames()` 函数返回的主机名列表的内容。
- `scds_free_net_list`(3HA) – 释放由 `scds_get_rs_hostnames()` 函数或 `scds_get_rg_hostnames()` 函数分配的内存。

端口列表函数

以下函数用于处理端口列表：

- `scds_get_port_list`(3HA) – 检索由资源使用的端口-协议对列表。
- `scds_print_port_list`(3HA) – 打印由 `scds_get_port_list()` 返回的端口-协议对列表的内容。
- `scds_free_port_list`(3HA) – 释放由 `scds_get_port_list()` 分配的内存。

网络地址函数

以下函数用于处理网络地址：

- `scds_get_netaddr_list`(3HA) – 检索由资源使用的网络地址列表。

- `scds_print_netaddr_list`(3HA) – 打印由 `scds_get_netaddr_list()` 返回的网络地址列表的内容。
- `scds_free_netaddr_list`(3HA) – 释放由 `scds_get_netaddr_list()` 分配的内存。

使用 TCP 连接进行故障监视

本节中的函数用于启用基于 TCP 的监视。通常，故障监视器使用这些函数来建立与服务之间的简单套接字连接、从服务中读取和向服务中写入数据以确定其状态，以及断开与服务的连接。

这些函数包括：

- `scds_fm_tcp_connect`(3HA) – 与仅使用 IPv4 寻址的进程建立 TCP 连接。
- `scds_fm_net_connect`(3HA) – 与使用 IPv4 或 IPv6 寻址的进程建立 TCP 连接。
- `scds_fm_tcp_read`(3HA) – 使用 TCP 连接从被监视的进程中读取数据。
- `scds_fm_tcp_write`(3HA) – 使用 TCP 连接向被监视的进程写入数据。
- `scds_simple_probe`(3HA) – 通过建立和终止与进程的 TCP 连接来探测进程。此函数只能处理 IPv4 地址。
- `scds_simple_net_probe`(3HA) – 通过建立和终止与进程的 TCP 连接来探测进程。此函数可以处理 IPv4 或 IPv6 地址。
- `scds_fm_tcp_disconnect`(3HA) – 终止与被监视进程的连接。此函数只能处理 IPv4 地址。
- `scds_fm_net_disconnect`(3HA) – 终止与被监视进程的连接。此函数可以处理 IPv4 或 IPv6 地址。

PMF 函数

这些函数封装了进程监视器工具 (PMF) 的功能。通过 PMF 进行监视的 DSDL 模型用于创建和使用 `pmfadm` 的隐含 *tag* 值。有关更多信息，请参见 `pmfadm`(1M) 手册页。

PMF 工具还使用 `Restart_interval`、`Retry_count` 和 `action_script`（`pmfadm` 的 `-t`、`-n` 和 `-a` 选项）的隐含值。最重要的是，DSDL 将由 PMF 确定的进程故障历史记录与由故障监视器检测到的应用程序故障历史记录相结合，来确定重新启动或故障转移决策。

此函数集包括以下函数：

- `scds_pmf_get_status`(3HA) – 确定指定实例是否正在 PMF 的控制下受到监视。
- `scds_pmf_restart_fm`(3HA) – 使用 PMF 来重新启动故障监视器。

- `scds_pmf_signal`(3HA) – 向在 PMF 的控制下运行的进程树发送指定信号。
- `scds_pmf_start`(3HA) – 执行在 PMF 的控制下的指定程序（包括故障监视器）。
- `scds_pmf_stop`(3HA) – 终止在 PMF 的控制下运行的进程。
- `scds_pmf_stop_monitoring`(3HA) – 停止监视在 PMF 的控制下运行的进程。

故障监视器函数

本节中的函数通过保留故障历史记录并将 `Retry_count` 和 `Retry_interval` 属性相结合对该历史记录进行评估，提供了预确定的故障监视模型。

此函数集包括以下函数：

- `scds_fm_sleep`(3HA) – 等待关于故障监视器控制套接字的消息。
- `scds_fm_action`(3HA) – 探测完成后采取措施。
- `scds_fm_print_probes`(3HA) – 向系统日志写入探测状态信息。

实用程序函数

使用以下函数，可以向系统日志写入消息和调试消息：

- `scds_syslog`(3HA) – 向系统日志写入消息。
- `scds_syslog_debug`(3HA) – 向系统日志写入调试消息。

第 12 章

群集重新配置通知协议

本章介绍群集重配置通知协议 (CRNP)。CRNP 使故障转移和可伸缩应用程序成为“群集可识别”的应用程序。更重要的是，CRNP 提供了一种机制，使应用程序能够注册和接收 Sun Cluster 重新配置事件的后续异步通知。群集内运行的数据服务以及群集外运行的应用程序都可以注册事件通知。当群集中的成员发生变化或者资源组或某个资源的状态发生变化时，都会生成事件。

注 – `SUNW.Event` 资源类型实现在 Sun Cluster 上提供了具有高可用性的 CRNP 服务。`SUNW.Event`(5) 手册页中对此资源类型的实现进行了详细介绍。

本章包括以下主题：

CRNP 概念

CRNP 定义了标准的七层开放式系统互连 (OSI) 协议栈的应用层、表示层和会话层。传输层必须为 TCP，网络层必须为 IP。CRNP 独立于数据链路层和物理层上。在 CRNP 中交换的所有应用层消息都基于 XML 1.0。

CRNP 的工作原理

CRNP 提供了一些机制和守护进程，用于生成群集重新配置事件、通过群集路由事件并将其发送给感兴趣的客户机。

`cl_apid` 守护进程与客户机交互。Sun Cluster 资源组管理器 (RGM) 生成群集重配置事件。此守护进程使用 `sysevent`d 来传送每个本地节点上的事件。`cl_apid` 守护进程使用可扩展标记语言 (XML) 通过 TCP/IP 与感兴趣的客户机进行通信。

下图显示了 CRNP 组件之间的事件流程。在该图中，一台客户机在群集节点 2 上运行，另一台客户机在不属于该群集的计算机上运行。

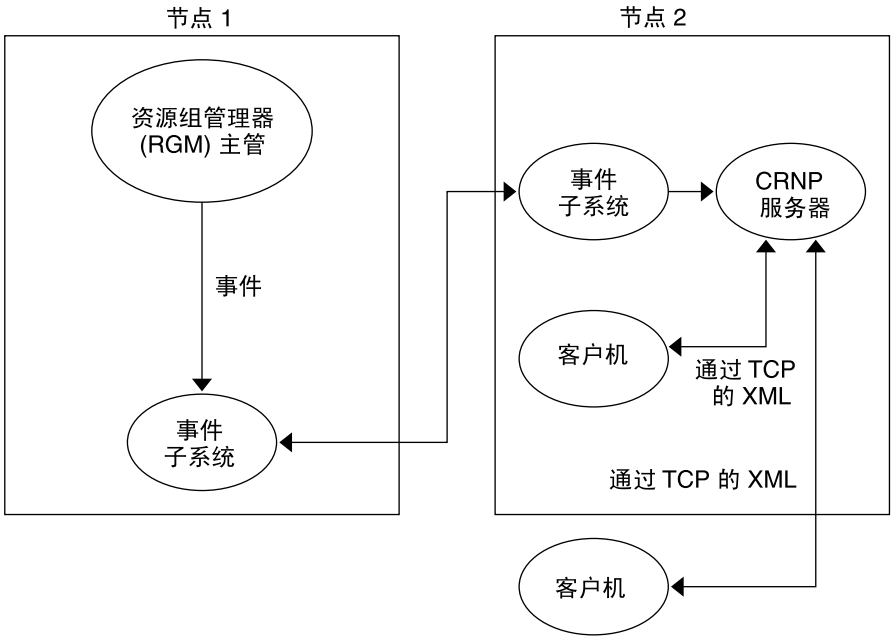


图 12–1 CRNP 组件之间的事件流程

CRNP 语义学

客户机通过向服务器发送一条注册消息 (`SC_CALLBACK_RG`) 来启动通信。此注册消息指定客户机要接收通知的事件类型，以及接收事件的端口。注册连接的源 IP 和指定的端口结合在一起形成回调地址。

当群集中生成某台客户机感兴趣的事件时，服务器将通过回调地址（IP 地址和端口号）联系该客户机，并将事件 (`SC_EVENT`) 传送给该客户机。在群集中运行的服务器具有高度的可用性。服务器将客户机注册存储在存储器中，即使重新引导群集，注册信息也会保留在存储器中。

客户机通过向服务器发送一条注册消息（SC_CALLBACK_RG，它包含 REMOVE_CLIENT 消息）进行取消注册。客户机接收到来自服务器的 SC_REPLY 消息后，将关闭连接。

下图显示了客户机和服务器之间的通信流程。

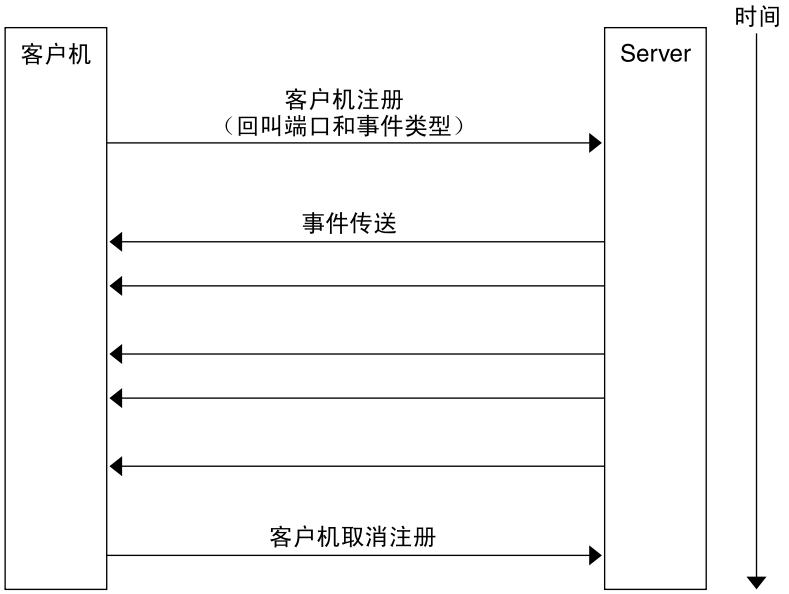


图 12–2 客户机和服务器之间的通信流程

CRNP 消息类型

CRNP 使用三种基于 XML 的消息。下表介绍了这些消息的用法。本章后面将对这些消息类型进行详细介绍，

CRNP 消息类型	说明
SC_CALLBACK_REG	此类消息采用四种格式：ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS 和 REMOVE_EVENTS。每种格式都包含以下信息： ■ 协议版本 ■ ASCII 格式（而不是二进制格式）的回调端口 ADD_CLIENT、ADD_EVENTS 和 REMOVE_EVENTS 还包含无界限的事件类型列表，每个事件类型都包括以下信息： ■ 事件类 ■ 事件子类（可选） ■ 名称和值对列表（可选） 由事件类和事件子类共同定义唯一的“事件类型”。生成 SC_CALLBACK_REG 类的文档类型定义 (DTD) 为 SC_CALLBACK_REG。附录 F 中对此 DTD 进行了详细介绍。
SC_REPLY	此类消息包含以下信息： ■ 协议版本 ■ 错误代码 ■ 错误消息 生成 SC_REPLY 类的 DTD 是 SC_REPLY。附录 F 中对此 DID 进行了详细介绍。

SC_EVENT	此类消息包含以下信息：
	■ 协议版本
	■ 事件类
	■ 事件子类
	■ 供应商
	■ 发行商
	■ 名称和值对列表（0 或多个名称和值对数据结构）
	  ■ 名称（字符串）
	  ■ 值（字符串或字符串数组）
	
	SC_EVENT 中的值未定义类型。生成 SC_EVENT 类的 DTD 是 SC_EVENT。附录 F 中对此 DID 进行了详细介绍。

客户机如何向服务器进行注册

本节介绍群集管理员如何设置服务器、如何标识客户机以及如何通过应用层和会话层发送信息，还介绍了错误状态。

管理员设置服务器的前提

群集管理员必须使用具有高可用性的 IP 地址（尚未连接到群集中特定计算机的一个 IP 地址）和端口号来配置服务器。群集管理员必须向预期的客户机发布此网络地址。CRNP 没有定义客户机获得该服务器名称的方式。群集管理员可以使用命名服务，使 客户机能够动态查找服务器的网络地址，也可以将网络名称添加到配置文件中以供客户机读取。服务器在群集中作为故障转移资源类型运行。

服务器如何标识客户机

每台客户机均由其回调地址（即其 IP 地址和端口号）唯一标识。端口是在 SC_CALLBACK_REG 消息中指定的，IP 地址是从 TCP 注册连接中获得的。CRNP 假定使用相同回调地址的后续 SC_CALLBACK_REG 消息来自同一台客户机，即使发送消息的源端口不同。

如何在客户机和服务器之间传送 SC_CALLBACK_REG 消息

客户机通过打开与服务器的 IP 地址和端口号的 TCP 连接启动注册。建立 TCP 连接并做好写入准备后，客户机必须发送其注册消息。注册消息必须是一条格式正确的 SC_CALLBACK_REG 消息，消息前后不能包含其他字节。

所有字节均已写入数据流后，该客户机必须使连接保持打开状态，以便接收来自服务器的应答。如果客户机没有正确格式化消息，服务器将不注册客户机，并向客户机发送一条出错应答。但是，如果客户机在服务器发出回复之前关闭套接字连接，服务器将照常注册客户机。

客户机可以随时联系服务器。客户机每次联系服务器时都必须发送一条 SC_CALLBACK_REG 消息。如果服务器发送的消息格式不正确、次序有误或者无效，服务器将向该客户机发送一条出错应答。

客户机必须先发送 ADD_CLIENT 消息，再发送 ADD_EVENTS、REMOVE_EVENTS 或 REMOVE_CLIENT 消息。客户机必须先发送 ADD_CLIENT 消息，再发送 REMOVE_CLIENT 消息。

如果某台客户机发送一条 ADD_CLIENT 消息，但该客户机已经注册，那么服务器可能会接收这条消息。这种情况下，服务器将使用第二条 ADD_CLIENT 消息中指定的新客户机注册替换旧的客户机注册，替换时不会发出提示。

大多数情况下，客户机在启动时通过发送一条 ADD_CLIENT 消息向服务器注册一次，通过向服务器发送 REMOVE_CLIENT 消息可取消注册一次客户机。然而，CRNP 为那些需要动态修改其事件类型列表的客户机提供了更大的灵活性。

SC_CALLBACK_REG 消息的内容

每个 ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS 和 REMOVE_EVENTS 消息都包含一个事件列表。下表描述了 CRNP 接受的事件类型，包括所需的名称和值对。

如果客户机执行以下操作之一，服务器将在不提示的情况下忽略这些消息：

- 发送 REMOVE_EVENTS 消息，用于指定客户机先前未注册的一个或多个事件类型
- 为同一事件类型注册了两次

类和子类	名称和值对	说明
EC_Cluster ESC_cluster_membership	必需：无 可选：无	注册所有群集成员更改事件（节点断开或连接）
EC_Cluster ESC_cluster_rg_state	一个是必需的，如下所示： rg_*name* 值类型：字符串 可选：无	注册资源组 *name* 的所有状态更改事件
EC_Cluster ESC_cluster_r_state	一个是必需的，如下所示： r_*name* 值类型：字符串 可选：无	注册资源 *name* 的所有状态更改事件
EC_Cluster 无	必需：无 可选：无	注册所有 Sun Cluster 事件

服务器如何对客户机进行应答

处理注册之后，收到注册请求的服务器将在客户机打开的 TCP 连接上发送 SC_REPLY 消息。服务器将关闭连接。客户机必须保持 TCP 连接为打开状态，直到接收到来自服务器的 SC_REPLY 消息。

例如，客户机将执行以下操作：

1. 打开服务器 TCP 连接
2. 等候连接进入“可写入”状态
3. 发送 SC_CALLBACK_REG 消息（其中包含 ADD_CLIENT 消息）
4. 等待来自服务器的 SC_REPLY 消息
5. 收到来自服务器的 SC_REPLY 消息
6. 接收服务器已关闭连接（从套接字读取 0 字节）的指示
7. 关闭连接

稍后，客户机将执行以下操作：

1. 打开服务器 TCP 连接
2. 等候连接进入“可写入”状态
3. 发送 SC_CALLBACK_REG 消息（其中包含 REMOVE_CLIENT 消息）
4. 等待来自服务器的 SC_REPLY 消息
5. 收到来自服务器的 SC_REPLY 消息
6. 接收服务器已关闭连接（从套接字读取 0 字节）的指示
7. 关闭连接

服务器每次接收到来自客户机的 SC_CALLBACK_REG 消息时，都会通过同一个打开的连接发送一条 SC_REPLY 消息。此消息用于指明该操作是否成功。第 304 页中的“SC_REPLY XML DTD” 包含 SC_REPLY 消息的 XML 文档类型定义，以及此消息可以包括的可能错误消息。

SC_REPLY 消息的内容

SC_REPLY 消息用于指定操作是成功还是失败。此消息包含 CRNP 消息的版本、状态码和状态消息（其中详细介绍了状态码）。下表描述了状态码可能具有的值。

状态码	说明
OK	已成功处理消息。
RETRY	由于瞬态错误，客户机的注册已被服务器拒绝。客户机应当使用其他参数重新尝试注册。
LOW_RESOURCE	群集资源不足，并且客户机只能稍后重新尝试。群集的群集管理员也可以增加群集中的资源。
SYSTEM_ERROR	发生严重问题。请与群集的群集管理员联系。
FAIL	授权失败，或者其他问题导致注册失败。
MALFORMED	XML 请求的格式不正确，无法进行分析。
INVALID	XML 请求无效，即 XML 请求不符合 XML 规范。
VERSION_TOO_HIGH	消息的版本过高，无法成功处理该消息。
VERSION_TOO_LOW	消息的版本太低，无法成功处理该消息。

客户机如何处理错误状态

在通常情况下，发送 SC_CALLBACK_REG 消息的客户机将收到指明注册是成功还是失败的回复。

但是，服务器可能会在客户机进行注册时遇到错误状态，该错误状态禁止服务器向客户机发送 SC_REPLY 消息。在这种情况下，注册可能已经在发生错误之前成功完成，也可能已经失败，还可能尚未进行。

由于服务器必须在群集中起故障转移或具有高可用性的服务器的作用，因此，这种错误状态并不意味着服务的结束。实际上，服务器可以很快开始向新注册的客户机发送事件。

要摆脱这些状态，客户机应执行以下操作：

- 在等待 SC_REPLY 消息的注册连接上强加应用级超时值，超出此超时值后客户机必须重新尝试注册。
- 在注册事件回调之前，开始在其回调 IP 地址和端口号上侦听事件传送。客户机应当同时等候注册确认消息和事件传送。如果客户机在接收到确认消息之前就开始接收事件，客户机将静默关闭注册连接。

服务器如何向客户机传送事件

事件在群集内生成时，CRNP 服务器会将其发送给已请求这些类型的事件的每个客户机。发送过程包括向客户机的回调地址发送一条 SC_EVENT 消息。每个事件都是通过一个新的 TCP 连接传送的。

客户机注册事件类型后，服务器立即通过一条包含 ADD_CLIENT 或 ADD_EVENT 消息的 SC_CALLBACK_REG 消息向客户机发送该类型的最新事件。客户机可以确定后续事件的源系统的当前状态。

当服务器启动客户机 TCP 连接时，服务器将通过该连接发送一条 SC_EVENT 消息，服务器将发出全双工关闭。

例如，客户机将执行以下操作：

1. 等候服务器启动 TCP 连接
2. 接受来自服务器的传入连接
3. 等待来自服务器的 SC_EVENT 消息
4. 读取来自服务器的 SC_EVENT 消息
5. 接收服务器已关闭连接（从套接字读取 0 字节）的指示
6. 关闭连接

所有客户机都注册后，必须始终在各自的回调地址（IP 地址和端口号）上侦听传入的事件传送连接。

如果服务器未能联系客户机以传送事件，将按照您指定的次数和时间间隔重新尝试传送事件。如果所有尝试均未成功，将从该服务器的客户机列表中删除此客户机。客户机还必须通过发送包含 ADD_CLIENT 消息的另一条 SC_CALLBACK_REG 消息重新注册之后，才能接收更多事件。

如何保障事件的传送

在群集中有一个事件生成的总排序，它是按照传送给每个客户机的顺序保留的。换言之，如果事件 A 在群集中比事件 B 生成得早，则客户机 X 将先收到事件 A，然后再收到事件 B。但是，**不**保留传送给**所有**客户机的事件的总排序。也就是说，客户机 Y 可以在客户机 X 接收到事件 A 之前接收事件 A 和事件 B。这样，速度慢的客户机将无法容纳传送到所有客户机的事件。

服务器传送的所有事件（子类的第一个事件和出现服务器错误后的事件除外）都是作为响应群集实际生成的事件而发生的，除非服务器遇到错误，导致丢失群集生成的事件。这种情况下，服务器将为表示系统当前状态的每个事件类型生成一个事件。每个事件都被发送到注册为对该事件类型感兴趣的客户机。

事件传送遵循“至少一次”的规则。即，服务器可以向客户机多次发送同一个事件。当服务器暂时出现故障，然后从故障中恢复后无法确定客户机是否已收到最新信息的情况下，这种宽容的规则是比要的。

SC_EVENT 消息的内容

SC_EVENT 消息包含在群集中生成的实际消息，此消息已翻译为符合 SC_EVENT XML 消息格式的消息。下表描述了 CRNP 传送的事件类型，包括名称和值对、发行商和供应商。

注 – state_list 中数组元素的位置与 node_list 中的位置同步。即，在 node_list 数据中先列出的节点状态将在 state_list 数组中先列出。

可能会有以 ev_ 为开头的其他名称及其关联值，但它们不供客户机使用。

类和子类	发行商和供应商	名称和值对
EC_Cluster	发行商：rgm	名称:node_list
ESC_cluster_membership	卖主:SUNW	值类型：字符串数组
		名称:state_list
		state_list 仅包含用 ASCII 表示的数字。每个数字都表示该节点在群集中的当前象征数字。如果该数字与上一个消息中接收到的数字相同，则说明该节点与群集之间的关系（脱离、连接或重新连接）尚未改变。如果象征数字是 –1，则说明该节点不是该群集的成员。如果象征数字非负，则说明该节点是该群集的成员。
		值类型：字符串数组
EC_Cluster	发行商：rgm	名称:rg_name
ESC_cluster_rg_state	卖主:SUNW	值类型：字符串
		名称:node_list
		值类型：字符串数组
		名称:state_list
		state_list 包含资源组状态的字符串表示。有效值为可用 scha_cmds(1HA) 命令检索到的值。
		值类型：字符串数组
EC_Cluster	发行商：rgm	名称:r_name
ESC_cluster_r_state	卖主:SUNW	值类型：字符串
		名称:node_list
		值类型：字符串数组
		名称:state_list
		state_list 包含资源状态的字符串表示。有效值为可用 scha_cmds(1HA) 命令检索到的值。
		值类型：字符串数组

CRNP 如何鉴别客户机和服务器

服务器将使用 TCP 封装程序的形式验证客户机。注册消息的源 IP 地址（也用作传送事件的回调 IP 地址）必须在服务器允许的客户机列表中。拒绝的客户机列表中不能包含源 IP 地址和注册消息。如果源 IP 地址和注册不在列表中，服务器将拒绝请求，并向客户机发送一个出错应答。

当服务器收到 SC_CALLBACK_REG ADD_CLIENT 消息时，该客户机的后续 SC_CALLBACK_REG 消息必须包含第一条消息中包含的同一个源 IP 地址。如果 CRNP 服务器收到不满足此要求的 SC_CALLBACK_REG，服务器将执行以下操作之一：

- 忽略请求并向客户机发送一个错误回复
- 假定根据 SC_CALLBACK_REG 消息的内容，该请求来自新的客户机

此安全机制有助于防止拒绝服务攻击，即防止有人尝试撤销合法客户机的注册。

客户机需要以同样的方式鉴别服务器。客户机只需接收其源 IP 地址和端口号与该客户机使用的注册 IP 地址和端口号相同的服务器上的事件传送。

由于 CRNP 服务的客户机必须位于保护群集的防火墙内，因此 CRNP 不包含其他安全机制。

创建使用 CRNP 的 Java 应用程序示例

以下示例说明了如何开发使用 CRNP 的简单 Java 应用程序 CrnpClient。应用程序将向群集中的 CRNP 服务器注册事件回调，侦听事件回调并通过打印其内容来处理事件。应用程序终止前将取消注册事件回调请求。

查看此示例时，请注意以下几点：

- 样例应用程序将使用 JAXP（用于 XML 处理的 Java API）生成和解析 XML。此示例不显示如何使用 JAXP。http://java.sun.com/xml/jaxp/index.html 中对 JAXP 进行了详细介绍。
- 此示例提供了部分应用程序，可在附录 G 中找到完整的应用程序。为了更有效地说明特殊概念，本章中的示例与附录 G 中提供的完整应用程序略有不同。
- 为简短起见，本章中的样例代码不包括注释。附录 G 中的完整应用程序包括注释。
- 此实例中的应用程序只是通过退出应用程序来处理大多数错误状态。实际的应用程序需要更有效地处理错误。

▼ 如何设置环境

步骤 **1. 下载并安装 JAXP 以及 Java 编译器和虚拟机的正确版本。**

可在 http://java.sun.com/xml/jaxp/index.html 中找到说明。

注 – 此示例至少要使用 Java 1.3.1。

---

2. **在资源文件所在的目录中键入：**

```
% javac -classpath jaxp-root/dom.jar:jaxp-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xsltc.jar -sourcepath . source-filename.java
```

其中，*jaxp-root* 为 JAXP jar 文件所在目录的绝对路径或相对路径，*source-filename* 为 Java 资源文件的名称。

编译命令行中的 `classpath` 用于确保编译器可以找到 JAXP 类。

3. **运行应用程序时，请指定 `classpath` 以使应用程序可以加载正确的 JAXP 类文件（注意，`classpath` 中的第一个路径是当前目录）：**

```
% java -cp .:jaxp-root/dom.jar:jaxp-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xsltc.jar source-filename arguments
```

现在已完成环境配置，可以开发应用程序了。

▼ 如何开始开发应用程序

在示例的这一部分中，通过使用解析命令行参数和构造 `CrnpClient` 对象的主方法，创建名为 `CrnpClient` 的基本类。此对象将命令行参数传递给类，等待用户终止应用程序，对 `CrnpClient` 调用 `shutdown` 命令，然后退出。

`CrnpClient` 类的构造函数需要执行以下任务：

- 设置 XML 处理对象。
- 创建一个侦听事件回调的线程。
- 联系 CRNP 服务器并注册事件回调。

步骤 ● **创建实现上述逻辑的 Java 代码。**

以下实例显示了 `CrnpClient` 类的骨架代码。本章稍后将列出在构造函数和关闭方法中引用的四个帮助器方法的实现。请注意，将列出导入所需的所有软件包的代码。

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;

class CrnpClient
{
        public static void main(String []args)
        {
                InetAddress regIp = null;
```

```
                int regPort = 0, localPort = 0;
                try {
                        regIp = InetAddress.getByName(args[0]);
                        regPort = (new Integer(args[1])).intValue();
                        localPort = (new Integer(args[2])).intValue();
                } catch (UnknownHostException e) {
                        System.out.println(e);
                        System.exit(1);
                }
                CrnpClient client = new CrnpClient(regIp, regPort,
                    localPort, args);
                System.out.println("Hit return to terminate demo...");
                try {
                        System.in.read();
                } catch (IOException e) {
                        System.out.println(e.toString());
                }
                client.shutdown();
                System.exit(0);
        }

        public CrnpClient(InetAddress regIpIn, int regPortIn,
            int localPortIn, String []clArgs)
        {
                try {
                        regIp = regIpIn;
                        regPort = regPortIn;
                        localPort = localPortIn;
                        regs = clArgs;
                        setupXmlProcessing();
                        createEvtRecepThr();
                        registerCallbacks();
                } catch (Exception e) {
                        System.out.println(e.toString());
                        System.exit(1);
                }
        }

        public void shutdown()
        {
                try {
                        unregister();
                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        private InetAddress regIp;
        private int regPort;
        private EventReceptionThread evtThr;
        private String regs[];
        public int localPort;
        public DocumentBuilderFactory dbf;
}
```

本章稍后将详细说明成员变量。

▼ 如何解析命令行参数

步骤 ● **要解析命令行参数，请参见附录 G 中的代码。**

▼ 如何定义事件接收线程

在代码中，必须确保事件接收是在单独的线程中执行，这样应用程序才能在事件线程阻塞和等待事件回调时继续执行其他工作。

注 – 本章稍后将讨论如何设置 XML。

步骤 **1. 在代码中，将定义名为 `EventReceptionThread` 的 `Thread` 子类，它用于创建 `ServerSocket` 和等待事件到达套接字。**

在实例代码的这一部分，既没有读取事件也没有处理事件。本章稍后将讨论如何读取和处理事件。`EventReceptionThread` 用于在通配符网络互联协议地址上创建 `ServerSocket`。`EventReceptionThread` 还引用了 `CrnpClient` 对象，以使 `EventReceptionThread` 能够向 `CrnpClient` 对象发送要处理的事件。

```
class EventReceptionThread extends Thread
{
        public EventReceptionThread(CrnpClient clientIn) throws IOException
        {
                client = clientIn;
                listeningSock = new ServerSocket(client.localPort, 50,
                    InetAddress.getLocalHost());
        }

        public void run()
        {
                try {
                        DocumentBuilder db = client.dbf.newDocumentBuilder();
                        db.setErrorHandler(new DefaultHandler());

                        while(true) {
                                Socket sock = listeningSock.accept();
                                // Construct event from the sock stream and process it
                                sock.close();
                        }
                        // UNREACHABLE

                } catch (Exception e) {
                        System.out.println(e);
```

```
                System.exit(1);
            }
        }

        /* private member variables */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. **构造 `createEvtRecepThr` 对象。**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
}
```

▼ 如何注册和取消注册回调

注册任务涉及以下操作：

- 向注册网络互联协议和端口打开一个基本 TCP 套接字
- 构造 XML 注册消息
- 通过套接字发送 XML 注册消息
- 脱离套接字并读取 XML 应答消息
- 关闭套接字

步骤 1. **创建实现上述逻辑的 Java 代码。**

以下示例代码显示了 `CrnpClient` 类（由 `CrnpClient` 构造函数调用）的 `registerCallbacks` 方法的实现。本章稍后将对 `createRegistrationString()` 和 `readRegistrationReply()` 的调用进行详细介绍。

`regIp` 和 `regPort` 是由构造函数设置的对象成员。

```
private void registerCallbacks() throws Exception
{
        Socket sock = new Socket(regIp, regPort);
        String xmlStr = createRegistrationString();
        PrintStream ps = new
                PrintStream(sock.getOutputStream());
        ps.print(xmlStr);
        readRegistrationReply(sock.getInputStream();
        sock.close();
}
```

2. **实现 `unregister` 方法。**

此方法由 `CrnpClient` 的 `shutdown` 方法调用。本章稍后将对 `createUnregistrationString` 的实现进行详细介绍。

```
private void unregister() throws Exception
{
```

```
            Socket sock = new Socket(regIp, regPort);
            String xmlStr = createUnregistrationString();
            PrintStream ps = new PrintStream(sock.getOutputStream());
            ps.print(xmlStr);
            readRegistrationReply(sock.getInputStream());
            sock.close();
    }
```

▼ 如何生成 XML

至此，已设置应用程序的结构并写入了所有联网代码，您需要写入生成和解析 XML 的代码。开始先写入生成 SC_CALLBACK_REG XML 注册消息的代码。

SC_CALLBACK_REG 消息包括注册类型（ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS 或 REMOVE_EVENTS）、回调端口和感兴趣的事件列表。每个事件都包括一个类和一个子类，后跟一个名称和值对列表。

在实例的这一部分，需要编写一个存储注册类型、回调端口和注册事件列表的 CallbackReg 类。此类还可将自身序列化为 SC_CALLBACK_REG XML 消息。

此类中有一个有趣的方法，即 convertToXml 方法，它可以从类成员中创建一个 SC_CALLBACK_REG XML 消息字符串。http://java.sun.com/xml/jaxp/index.html 中的 JAXP 文档详细说明了此方法中的代码。

以下示例代码中显示了 Event 类的实现。请注意，CallbackReg 类使用了 Event 类，该类用于存储一个事件并可将该事件转换为 XML Element。

步骤　1. 创建实现上述逻辑的 Java 代码。

```
class CallbackReg
{
        public static final int ADD_CLIENT = 0;
        public static final int ADD_EVENTS = 1;
        public static final int REMOVE_EVENTS = 2;
        public static final int REMOVE_CLIENT = 3;

        public CallbackReg()
        {
                port = null;
                regType = null;
                regEvents = new Vector();
        }

        public void setPort(String portIn)
        {
                port = portIn;
        }

        public void setRegType(int regTypeIn)
```

```
{
        switch (regTypeIn) {
        case ADD_CLIENT:
                regType = "ADD_CLIENT";
                break;
        case ADD_EVENTS:
                regType = "ADD_EVENTS";
                break;
        case REMOVE_CLIENT:
                regType = "REMOVE_CLIENT";
                break;
        case REMOVE_EVENTS:
                regType = "REMOVE_EVENTS";
                break;
        default:
                System.out.println("Error, invalid regType " +
                    regTypeIn);
                regType = "ADD_CLIENT";
                break;
        }
}

public void addRegEvent(Event regEvent)
{
        regEvents.add(regEvent);
}

public String convertToXml()
{
        Document document = null;
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                document = builder.newDocument();
        } catch (ParserConfigurationException pce) {
                // Parser with specified options can't be built
                pce.printStackTrace();
                System.exit(1);
        }

        // Create the root element
        Element root = (Element) document.createElement("SC_CALLBACK_REG");

        // Add the attributes
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Add the events
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(document));
        }
```

```
                document.appendChild(root);

                // Convert the whole thing to a string
                DOMSource domSource = new DOMSource(document);
                StringWriter strWrite = new StringWriter();
                StreamResult streamResult = new StreamResult(strWrite);
                TransformerFactory tf = TransformerFactory.newInstance();
                try {
                        Transformer transformer = tf.newTransformer();
                        transformer.transform(domSource, streamResult);
                } catch (TransformerException e) {
                        System.out.println(e.toString());
                        return ("");
                }
                return (strWrite.toString());
        }

        private String port;
        private String regType;
        private Vector regEvents;
}
```

2. **实现 `Event` 和 `NVPair` 类。**

   请注意，`CallbackReg` 类使用了 `Event` 类，而该类自身使用了 `NVPair` 类。

```
class Event
{

        public Event()
        {
                regClass = regSubclass = null;
                nvpairs = new Vector();
        }

        public void setClass(String classIn)
        {
                regClass = classIn;
        }

        public void setSubclass(String subclassIn)
        {
                regSubclass = subclassIn;
        }

        public void addNvpair(NVPair nvpair)
        {
                nvpairs.add(nvpair);
        }

        public Element createXmlElement(Document doc)
        {
                Element event = (Element)
                    doc.createElement("SC_EVENT_REG");
                event.setAttribute("CLASS", regClass);
                if (regSubclass != null) {
```

```
                        event.setAttribute("SUBCLASS", regSubclass);
                }
                for (int i = 0; i < nvpairs.size(); i++) {
                     NVPair tempNv = (NVPair)
                          (nvpairs.elementAt(i));
                     event.appendChild(tempNv.createXmlElement(doc));
                }
                return (event);
        }

        private String regClass, regSubclass;
        private Vector nvpairs;
}

class NVPair
{
        public NVPair()
        {
                name = value = null;
        }

        public void setName(String nameIn)
        {
                name = nameIn;
        }

        public void setValue(String valueIn)
        {
                value = valueIn;
        }

        public Element createXmlElement(Document doc)
        {
                Element nvpair = (Element)
                    doc.createElement("NVPAIR");
                Element eName = doc.createElement("NAME");
                Node nameData = doc.createCDATASection(name);
                eName.appendChild(nameData);
                nvpair.appendChild(eName);
                Element eValue = doc.createElement("VALUE");
                Node valueData = doc.createCDATASection(value);
                eValue.appendChild(valueData);
                nvpair.appendChild(eValue);

                return (nvpair);
        }

        private String name, value;
}
```

▼ 如何创建注册消息和取消注册消息

请注意，创建生成 XML 消息的帮助器类之后，您可以向 `createRegistrationString` 方法的实现写入数据。此方法由 `registerCallbacks` 方法（第 199 页中的 “如何注册和取消注册回调”中介绍了该方法）调用。

`createRegistrationString` 构造一个 `CallbackReg` 对象并设置其注册类型和端口。然后，`createRegistrationString` 通过使用 `createAllEvent`、`createMembershipEvent`、`createRgEvent` 和 `createREvent` 帮助器方法构造各种事件。创建 `CallbackReg` 对象后，每个事件都被添加到该对象中。最后，`createRegistrationString` 对 `CallbackReg` 对象调用 `convertToXml` 方法，以便在 `String` 表单中检索 XML 消息。

请注意，`regs` 成员变量用于存储用户提供给应用程序的命令行参数。第五个参数及后续参数用于指定应用程序应该注册的事件。第四个参数用于指定注册的类型，但本实例中忽略了该参数。附录 G 中的完整代码显示了如何使用第四个参数。

步骤　1. 创建实现上述逻辑的 Java 代码。

```
private String createRegistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);

        cbReg.setRegType(CallbackReg.ADD_CLIENT);

        // add the events
        for (int i = 4; i < regs.length; i++) {
                if (regs[i].equals("M")) {
                        cbReg.addRegEvent(createMembershipEvent());
                } else if (regs[i].equals("A")) {
                        cbReg.addRegEvent(createAllEvent());
                } else if (regs[i].substring(0,2).equals("RG")) {
                        cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
                } else if (regs[i].substring(0,1).equals("R")) {
                        cbReg.addRegEvent(createREvent(regs[i].substring(2)));
                }
        }

        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}

private Event createAllEvent()
{
        Event allEvent = new Event();
        allEvent.setClass("EC_Cluster");
        return (allEvent);
}

private Event createMembershipEvent()
```

```
{
        Event membershipEvent = new Event();
        membershipEvent.setClass("EC_Cluster");
        membershipEvent.setSubclass("ESC_cluster_membership");
        return (membershipEvent);
}

private Event createRgEvent(String rgname)
{
        Event rgStateEvent = new Event();
        rgStateEvent.setClass("EC_Cluster");
        rgStateEvent.setSubclass("ESC_cluster_rg_state");

        NVPair rgNvpair = new NVPair();
        rgNvpair.setName("rg_name");
        rgNvpair.setValue(rgname);
        rgStateEvent.addNvpair(rgNvpair);

        return (rgStateEvent);
}

private Event createREvent(String rname)
{
        Event rStateEvent = new Event();
        rStateEvent.setClass("EC_Cluster");
        rStateEvent.setSubclass("ESC_cluster_r_state");

        NVPair rNvpair = new NVPair();
        rNvpair.setName("r_name");
        rNvpair.setValue(rname);
        rStateEvent.addNvpair(rNvpair);

        return (rStateEvent);
}
```

2. **创建撤销注册字符串。**

   创建取消注册字符串比创建注册字符串容易，因为您不必提供事件。

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}
```

▼ 如何设置 XML 解析器

现在已经为应用程序创建了联网和 XML 生成代码。`CrnpClient` 构造函数将调用 `setupXmlProcessing` 方法。此方法将创建 `DocumentBuilderFactory` 对象，并在该对象上设置各种解析属性。JAXP 文档详细介绍了此方法。请参见 `http://java.sun.com/xml/jaxp/index.html`。

步骤 ● **创建实现上述逻辑的 Java 代码。**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        // We don't need to bother validating
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // We want to ignore comments and whitespace
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Coalesce CDATA sections into TEXT nodes.
        dbf.setCoalescing(true);
}
```

▼ 如何解析注册回复

要解析 SC_REPLY XML 消息（CRNP 服务器发送的响应注册或取消注册消息），需要使用 RegReply 帮助器类。可以从 XML 文档构造此类。此类提供了状态代码和状态消息的访问程序。要解析来自服务器的 XML 流，需要创建新的 XML 文档并使用该文档的解析方法。http://java.sun.com/xml/jaxp/index.html 中的 JAXP 文档详细介绍了此方法。

步骤 **1. 创建实现上述逻辑的 Java 代码。**

请注意，readRegistrationReply 方法使用新的 RegReply 类。

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();
        db.setErrorHandler(new DefaultHandler());

        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
}
```

2. 实现 RegReply 类。

请注意，retrieveValues 方法将检索 XML 文档中的 DOM 树，并提取状态代码和状态消息。http://java.sun.com/xml/jaxp/index.html 中的 JAXP 文档包含详细说明。

```
class RegReply
{
        public RegReply(Document doc)
```

```
{
        retrieveValues(doc);
}

public String getStatusCode()
{
        return (statusCode);
}

public String getStatusMsg()
{
        return (statusMsg);
}
public void print(PrintStream out)
{
        out.println(statusCode + ": " +
            (statusMsg != null ? statusMsg : ""));
}

private void retrieveValues(Document doc)
{
        Node n;
        NodeList nl;
        String nodeName;

        // Find the SC_REPLY element.
        nl = doc.getElementsByTagName("SC_REPLY");
        if (nl.getLength() != 1) {
                System.out.println("Error in parsing: can't find "
                    + "SC_REPLY node.");
                return;
        }

        n = nl.item(0);

        // Retrieve the value of the statusCode attribute
        statusCode = ((Element)n).getAttribute("STATUS_CODE");

        // Find the SC_STATUS_MSG element
        nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
        if (nl.getLength() != 1) {
                System.out.println("Error in parsing: can't find "
                    + "SC_STATUS_MSG node.");
                return;
        }
        // Get the TEXT section, if there is one.
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        // Not an error if there isn't one, so we just silently return.
                return;
        }

        // Retrieve the value
        statusMsg = n.getNodeValue();
}
```

```
        private String statusCode;
        private String statusMsg;
}
```

▼ 如何解析回调事件

最后一步是解析和处理实际的回调事件。要帮助完成此任务，需要修改在“第 200 页中的 “如何生成 XML””中创建的 `Event` 类，以使该类能够从 XML 文档构建 `Event` 并创建 XML `Element`。此更改需要使用其他构造函数（获取 XML 文档）、`retrieveValues` 方法、添加两个成员变量（`vendor` 和 `publisher`）、所有字段的访问程序方法，最后，还需要打印方法。

步骤 **1. 创建实现上述逻辑的 Java 代码。**

请注意，此代码与“第 206 页中的 “如何解析注册回复””中说明的 `RegReply` 类的代码类似。

```
public Event(Document doc)
        {
                nvpairs = new Vector();
                retrieveValues(doc);
        }
        public void print(PrintStream out)
        {
                out.println("\tCLASS=" + regClass);
                out.println("\tSUBCLASS=" + regSubclass);
                out.println("\tVENDOR=" + vendor);
                out.println("\tPUBLISHER=" + publisher);
                for (int i = 0; i < nvpairs.size(); i++) {
                        NVPair tempNv = (NVPair)
                            (nvpairs.elementAt(i));
                        out.print("\t\t");
                        tempNv.print(out);
                }
        }

        private void retrieveValues(Document doc)
        {
                Node n;
                NodeList nl;
                String nodeName;

                // Find the SC_EVENT element.
                nl = doc.getElementsByTagName("SC_EVENT");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "SC_EVENT node.");
                   return;
                }
```

```
            n = nl.item(0);

            //
            // Retrieve the values of the CLASS, SUBCLASS,
            // VENDOR and PUBLISHER attributes.
            //
            regClass = ((Element)n).getAttribute("CLASS");
            regSubclass = ((Element)n).getAttribute("SUBCLASS");
            publisher = ((Element)n).getAttribute("PUBLISHER");
            vendor = ((Element)n).getAttribute("VENDOR");

            // Retrieve all the nv pairs
            for (Node child = n.getFirstChild(); child != null;
                child = child.getNextSibling())
            {
                   nvpairs.add(new NVPair((Element)child));
            }
    }

    public String getRegClass()
    {
            return (regClass);
    }

    public String getSubclass()
    {
            return (regSubclass);
    }

    public String getVendor()
    {
            return (vendor);
    }

    public String getPublisher()
    {
            return (publisher);
    }

    public Vector getNvpairs()
    {
            return (nvpairs);
    }

    private String vendor, publisher;
```

2. **实现用于支持 XML 解析的 `NVPair` 类的其他构造函数和方法。**

   对 `Event` 类的更改（如步骤 1 所述）需要对 `NVPair` 类进行类似的更改。

```
public NVPair(Element elem)
        {
                retrieveValues(elem);
        }
        public void print(PrintStream out)
        {
```

```
                out.println("NAME=" + name + " VALUE=" + value);
        }
        private void retrieveValues(Element elem)
        {
                Node n;
                NodeList nl;
                String nodeName;

                // Find the NAME element
                nl = elem.getElementsByTagName("NAME");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "NAME node.");
                   return;
                }
                // Get the TEXT section
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                   System.out.println("Error in parsing: can't find "
                       + "TEXT section.");
                   return;
                }

                // Retrieve the value
                name = n.getNodeValue();

                // Now get the value element
                nl = elem.getElementsByTagName("VALUE");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "VALUE node.");
                   return;
                }
                // Get the TEXT section
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                System.out.println("Error in parsing: can't find "
                            + "TEXT section.");
                        return;
                }

                // Retrieve the value
                value = n.getNodeValue();
                }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3. **实现 `EventReceptionThread` 中的 `while` 循环，它将等待事件回调。**

   “第 198 页中的 “如何定义事件接收线程””中介绍了 `EventReceptionThread`。

```
while(true) {
                Socket sock = listeningSock.accept();
                Document doc = db.parse(sock.getInputStream());
                Event event = new Event(doc);
                client.processEvent(event);
                sock.close();
        }
```

▼ 如何运行应用程序

步骤 1. **成为超级用户或作为等效角色。**

2. **运行应用程序。**

```
# java CrnpClient crnpHost crnpPort localPort ...
```

   附录 G 中列出了 `CrnpClient` 应用程序的完整代码。

附录 A

标准属性

本附录介绍了标准资源类型、资源组和资源的属性。本附录还介绍了可用于更改系统定义的属性以及创建扩展属性的资源特性属性。

注 – 资源类型、资源和资源组的属性名称**不**区分大小写。指定属性名称时，您可以使用大写和小写字母的任意组合。

本附录包括以下主题：

资源类型属性

以下信息介绍了由 Sun Cluster 软件定义的资源类型属性。属性值分为以下几类：

- **必需**。此属性要求资源类型注册 (RTR) 文件中有明确的值。否则，将无法创建属性所属的对象。此值不能为空格或空字符串。
- **有条件的**。必须在 RTR 文件中声明此属性，此属性才存在。否则，RGM 不会创建该属性，并且该属性将不能用于管理实用程序。允许使用空格或空字符串。如果在 RTR 文件中声明属性但未指定值，RGM 将提供缺省值。
- **有条件的或显式**。必须在 RTR 文件中声明此属性的明确值，此属性才存在。否则，RGM 不会创建该属性，并且该属性将不能用于管理实用程序。不允许使用空格或空字符串。
- **可选**。可在 RTR 文件中声明此属性。如果未在 RTR 文件中声明属性，则 RGM 将创建属性并提供一个缺省值。如果在 RTR 文件中声明了属性但未指定值，则 RGM 将提供相同的默认值，就像没在 RTR 文件中声明该属性一样。

- **仅限于查询** - 不能通过管理工具直接进行设置。

不能通过管理实用程序更新资源类型属性，`Installed_nodes` 和 `RT_system` 除外，这两个资源类型属性不能在 RTR 文件中声明并且必须由群集管理员设置。

首先列出的是属性名称，后面是对该属性的说明。

注 – 资源类型属性名称（例如 `API_version` 和 `Boot`）**不**区分大小写。指定属性名称时可以使用大小写字母的任意组合。

`API_version` (`integer`)
支持此资源类型实现所需的资源管理 API 的最低版本。

以下信息汇总了 Sun Cluster 的每个版本所支持的最高 `API_version`。

3.1 版或更低版本	`2`
3.1 10/03	`3`
3.1 4/04	`4`
3.1 9/04	`5`
3.1 8/05	`6`

在 RTR 文件中为 `API_version` 声明一个大于 `2` 的值将可防止资源类型被安装到支持低于最高版本的 Sun Cluster 版本上。例如，如果您为某个资源类型声明了 `API_version=5`，该资源类型将不能安装在 3.1 9/04 之前发行的任何版本的 Sun Cluster 上。

注 – 如果不将此属性声明或设置为默认值 (`2`)，则可将数据服务安装到 Sun Cluster 3.0 以后的任何版本的 Sun Cluster 上。

类别： 可选

缺省值： `2`

可调： 无

`Boot` (`string`)
一种可选的回调方法：节点连接或重新连接群集时，RGM 将对此节点调用的程序的路径（此类型的资源处于被管理状态时）。此方法初始化此类资源的方法与 `Init` 方法相同。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

Failover (boolean)
True 用于指明不能在可同时在多个节点上联机的任何组中配置此类资源。

下表显示了如何将此资源类型属性与 Scalable 资源属性结合使用。

Failover 资源类型的值	Scalable 资源的值	说明
TRUE	TRUE	请不要指定这个不合逻辑的组合。
TRUE	FALSE	可为故障转移服务指定此组合。
FALSE	TRUE	可为使用 SharedAddress 资源进行网络负载平衡的可伸缩服务指定此组合。 《Sun Cluster 概念指南（适用于 Solaris OS）》详细介绍了 SharedAddress。
FALSE	FALSE	虽然这不是常见组合，但您可以使用此组合选择不使用网络负载平衡的多主服务。

r_properties(5) 手册页中的对 Scalable 的说明和《Sun Cluster 概念指南（适用于 Solaris OS）》中的第 3 章 "适用于系统管理员和应用程序开发者的关键概念"包含附加信息。

类别： 可选

缺省值： FALSE

可调： 无

Fini (string)
一种可选的回调方法：从 RGM 管理中删除此类资源时，RGM 所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

Init (string)
一种可选的回调方法：当此类资源开始处于 RGM 的管理下时，RGM 所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

Init_nodes (enum)
指明 RGM 对哪些节点调用 Init、Fini、Boot 和 Validate 方法。值可为 RG_PRIMARIES（仅为可以控制资源的节点）或 RT_INSTALLED_NODES（安装了该资源类型的所有节点）。

类别： 可选

缺省值： RG_PRIMARIES

可调： 无

Installed_nodes (string_array)
可运行资源类型的群集节点名称的列表。RGM 将自动创建此属性。群集管理员可以设置此值。不能在 RTR 文件中声明该属性。

类别： 可以由群集管理员进行配置

缺省值： 所有群集节点

可调： ANYTIME

Is_logical_hostname (boolean)
TRUE 表明此资源类型是用来管理故障转移 Internet 协议 (IP) 地址的 LogicalHostname 资源类型的某个版本。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

Is_shared_address (boolean)
TRUE 表明该资源类型是管理故障转移 Internet 协议 (IP) 地址的 SharedAddress 资源类型的某个版本。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

Monitor_check (string)
一种可选的回调方法：在对此类资源执行监视器请求的故障转移之前，RGM 所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

Monitor_start (string)
一种可选的回调方法：RGM 为启动此类资源的故障监视器所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

Monitor_stop (string)
设置 Monitor_start 后需要的回调方法：RGM 为停止此类资源的故障监视器所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Pkglist` (`string_array`)
包含在资源类型安装中的软件包的可选列表。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Postnet_stop` (`string`)
一种可选的回调方法：在调用此类资源所依赖的任何网络地址资源的 `Stop` 方法后，RGM 运行的程序的路径。在将网络接口配置为关闭之后，此方法必须执行 `Stop` 操作。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Prenet_start` (`string`)
一种可选的回调方法：在 RGM 调用此类资源所依赖的任何网络地址资源的 `Start` 方法之前，RGM 运行的程序的路径。此方法将执行在配置网络接口之前必须执行的 `Start` 操作。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Resource_list` (`string_array`)
资源类型的所有资源的列表。群集管理员不直接设置此属性。而是当群集管理员在任何资源组中添加或删除此类资源时，由 RGM 更新此属性。

类别： 仅限于查询

缺省值： 空列表

可调： 无

`Resource_type` (`string`)
资源类型的名称。要查看当前已注册的资源类型的名称，请使用：

`scrgadm` `-p`

在 Sun Cluster 3.1 及更高版本中，资源类型名称必须包含版本：

vendor-id`.`*resource-type*`:`*rt-version*

资源类型名称的三个组成部分是在 RTR 文件中指定为 *vendor-id*、*resource-type* 和 *rt-version* 的属性。`scrgadm` 命令用于插入句点 (.) 和冒号 (:)分界符。资源类型名称的 *rt-version* 后缀的值与 `RT_version` 属性的值相同。要确保 *vendor-id* 唯一，请使用要创建资源类型的公司的股票代号。在 Sun Cluster 3.1 之前的版本中创建的资源类型名称继续采用以下语法：

vendor-id`.`*resource-type*

类别： 必需的

缺省值： 空字符串

可调： 无

`RT_basedir` (`string`)
用于完成回调方法的相对路径的目录路径。必须将此路径设置为资源类型软件包的安装目录。路径必须为完整路径，即必须以正斜杠 (/) 开头。

类别： 必需的，除非所有方法路径名称均为绝对路径

缺省值： 无缺省值

可调： 无

`RT_description` (`string`)
资源类型的简单说明。

类别： 有条件的

缺省值： 空字符串

可调： 无

`RT_system` (`boolean`)
如果资源类型的 `RT_system` 属性为 `TRUE`，则不能删除该资源类型 (`scrgadm -r -t` *resource-type-name*)。此属性用于防止意外删除资源资源类型，例如用于支持群集基础结构的 `LogicalHostname`。但是，您可以将 `RT_system` 属性应用于任何资源类型。

要删除 `RT_system` 属性设置为 `TRUE` 的资源类型，必须先将此属性设置为 `FALSE`。删除其资源支持群集服务的资源类型时需谨慎。

类别： 可选

缺省值： `FALSE`

可调： `ANYTIME`

`RT_version` (`string`)
从 Sun Cluster 3.1 开始，实施该资源类型必需版本字符串。`RT_version` 是完整资源类型名称的后缀部分。曾在 Sun Cluster 3.0 中为可选属性的 `RT_version` 属性在 Sun Cluster 3.1 及更高版本中是必需属性。

类别： 有条件的/显式或必需的

缺省值： 无缺省值

可调： 无

`Single_instance` (`boolean`)
如果将其设置为 `TRUE`，则表明在群集中只能存在一个此类型的资源。RGM 仅允许在群集中一次只运行此类型的一个资源。

类别： 可选

缺省值： `FALSE`

可调： 无

`Start` (`string`)
一种回调方法：RGM 为启动此类资源所运行的程序的路径。

类别： 必需的（除非在 RTR 文件中声明了 `Prenet_start` 方法）

缺省值： 无缺省值

可调： 无

`Stop` (`string`)
一种回调方法：RGM 为停止此类资源所运行的程序的路径。

类别： 必需的（除非在 RTR 文件中声明了 `Postnet_stop` 方法）

缺省值： 无缺省值

可调： 无

`Update` (`string`)
一种可选的回调方法：当更改此类正在运行的资源的属性时，RGM 所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Validate` (`string`)
一种可选的回调方法：RGM 为检查此类资源的属性值所运行的程序的路径。

类别： 有条件的/显式

缺省值： 无缺省值

可调： 无

`Vendor_ID` (`string`)
请参见 `Resource_type` 属性。

类别： 有条件的

缺省值： 无缺省值

可调： 无

资源属性

此部分介绍了由 Sun Cluster 软件定义的资源属性。属性值分为以下几类：

- **必需**。群集管理员在使用管理实用程序创建资源时必须指定值。
- **可选**。如果群集管理员在创建资源组时未指定值，系统将会提供一个默认值。
- **有条件的**。仅当属性已在 RTR 文件中声明后，RGM 才能创建属性。否则，属性将不存在且对群集管理员不可用。RTR 文件中声明的条件属性是可选的还是必需的取决于是否在 RTR 文件中指定了缺省值。有关详细信息，请参见对每个条件的属性的说明。
- **仅限于查询**。不能通过管理工具直接设置。

第 240 页中的 “资源特性属性”中介绍的 `Tunable` 属性列出了您是否可以和何时可以更新资源属性，如下所示：

`FALSE` 或 `NONE`	永远不
`TRUE` 或 `ANYTIME`	任何时候
`AT_CREATION`	在将资源添加到群集时
`WHEN_DISABLED`	当资源被禁止时

首先列出的是属性名称，后面是对该属性的说明。

`Affinity_timeout` (`integer`)
: 以秒表示的时间长度，在此期间，对于资源中的任何服务，来自给定客户机IP 地址的连接均被发送到同一服务器节点。

  仅当 `Load_balancing_policy` 的值为 `Lb_sticky` 或 `Lb_sticky_wild` 时，此属性才适用。另外，必须将 `Weak_affinity` 设置为 `FALSE`。

  此属性只用于可伸缩服务。

  类别： 可选

  缺省值： 无缺省值

  可调： `ANYTIME`

类型中的每个回调方法的 `Boot_timeout` (`integer`)
: 以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

  类别： 有条件的/可选

  缺省值： `3600`（一小时），如果在 RTR 文件中声明了此方法本身

  可调： `ANYTIME`

Cheap_probe_interval (integer)
在两次资源故障快速探测的调用之间的秒数。该属性由 RGM 创建，并且仅当在 RTR 文件中声明此属性后，群集管理员才可以使用此属性。如果在 RTR 文件中指定了缺省值，则此属性是可选的。

如果在 RTR 文件中未指定 Tunable 属性，则该属性的 Tunable 值为 WHEN_DISABLED。

类别： 有条件的

缺省值： 无缺省值

可调： WHEN_DISABLED

扩展属性
在资源类型的 RTR 文件中声明的扩展属性。资源类型的实现中定义了这些属性。第 240 页中的 “资源特性属性”中包含关于可为扩展属性设置的单个属性的信息。

类别： 有条件的

缺省值： 无缺省值

可调： 取决于特定的属性

Failover_mode (enum)
修改在出现以下情况时 RGM 所采取的恢复操作：当资源无法成功启动或停止时，或者在资源监视器发现资源运行不正常并因而请求重新启动或故障转移时。

NONE、SOFT 或 HARD（方法失败）

当启动或停止方法（Prenet_start、Start、Monitor_stop、Stop、Postnet_stop）失败时，这些设置仅影响故障转移行为。成功启动资源之后，NONE、SOFT 和 HARD 将不会对资源监视器使用 scha_control 命令或 scha_control() 函数启动的后续资源重新启动或停止行为有任何影响。请参见 scha_control(1HA) 和 scha_control(3HA) 手册页。NONE 表示当先前列出的某个启动或停止方法失败时，RGM 将不执行任何恢复操作。SOFT 或 HARD 表示如果 Start 或 Prenet_start 方法失败，RGM 会将资源的组重定位到其他节点上。对于 Start 或 Prenet_start 失败，SOFT 与 HARD 相同。

对于 stop 方法（Monitor_stop、Stop 或 Postnet_stop）的失败，SOFT 与 NONE 相同。如果 Failover_mode 设置为 HARD，当这些停止方法之一失败时，RGM 将重新引导节点以强制资源组脱机。然后，RGM 可能尝试在其他节点上启动该组。

RESTART_ONLY 或 LOG_ONLY

与 NONE、SOFT 和 HARD 不同，它们影响的是在启动或停止方法失败时的故障转移行为，而 RESTART_ONLY 和 LOG_ONLY 将影响所有故障转移行为。故障转移行为包括由监视器启动的 (scha_control) 资源和资源组重新启动，以及由资源监视器 (scha_control) 启动的停止。RESTART_ONLY 表示监视器可以运行 scha_control 来重新启动资源或资源组。RGM 允许在 Retry_interval 指定的

时间间隔内重新启动 Retry_count 次。如果超出 Retry_count 次，则不允许再进行重新启动。如果 Failover_mode 设置为 LOG_ONLY，则不允许进行资源重新启动或停止。将 Failover_mode 设置为 LOG_ONLY 相当于将 Failover_mode 设置为 RESTART_ONLY 并将 Retry_count 设置为零。

RESTART_ONLY 或 LOG_ONLY（方法失败）

如果 Prenet_start、Start、Monitor_stop、Stop 或 Postnet_stop 方法失败，则 RESTART_ONLY 和 LOG_ONLY 与 NONE 相同。即，既不故障转移也不重新引导节点。

Failover_mode 设置对数据服务的影响

Failover_mode 的每个设置对数据服务的影响取决于数据服务是受监视还是不受监视，数据服务是否基于数据服务开发库 (DSDL)。

- 如果数据服务实现了 Monitor_start 方法并且对资源的监视已启用，将对该数据服务进行监视。RGM 通过在启动资源自身之后执行 Monitor_start 方法来启动资源监视器。资源监视器将探测资源的运行状况。如果探测失败，资源监视器可能通过调用 scha_control() 函数请求重新启动或故障转移。对于基于 DSDL 的资源，探测可能会表明数据服务的部分失败（降级）或完全失败。重复的部分失败将累积为完全失败。
- 如果数据服务不提供 Monitor_start 方法或者对资源的监视已禁用，则不对数据服务进行监视。
- 基于 DSDL 的数据服务包括通过 GDS 使用 Agent Builder 开发的数据服务，或者通过使用 DSDL 直接开发的数据服务。某些数据服务（例如 HA Oracle）不是使用 DSDL 开发的。

NONE、SOFT 或 HARD（探测失败）

如果将 Failover_mode 设置为 NONE、SOFT 或 HARD 并且数据服务是基于 DSDL 的受监视服务，则当探测完全失败时，监视器将调用 scha_control() 函数以请求重新启动资源。如果探测持续失败，将在 Retry_interval 内将资源最多重新启动 Retry_count 次。如果在重新启动 Retry_count 次之后探测再次失败，监视器将请求把资源的组故障转移到其他节点上。

如果将 Failover_mode 设置为 NONE、SOFT 或 HARD 并且数据服务是基于 DSDL 的不受监视服务，则仅检测唯一一种失败— —资源的进程树中止。如果资源的进程树中止，则重新启动资源。

如果数据服务不是基于 DSDL 的服务，则重新启动或故障转移行为取决于资源监视器如何编码。例如，Oracle 资源监视器将通过重新启动资源或资源组，或者通过故障转移资源组进行恢复。

RESTART_ONLY（探测失败）

如果将 Failover_mode 设置为 RESTART_ONLY 并且数据服务是基于 DSDL 的受监视服务，则当探测完全失败时，将在 Retry_interval 内将资源重新启动 Retry_count 次。但是，如果超出 Retry_count 次，则资源监视器将退出、将资源状态设置为 FAULTED 并生成状态消息“应用程序有问题，但未重新启动。探测正在退出。”此时，虽然监视仍处于启用状态，但是在群集管理员修复并重新启动资源之前，资源将一直处于不受监视的状态。

如果将 Failover_mode 设置为 RESTART_ONLY 并且数据服务是基于 DSDL 的不受监视服务，则当进程树中止时，将不重新启动资源。

如果受监视的数据服务不是基于 DSDL 的数据服务，则恢复行为取决于如何对资源监视器进行编码。如果将 Failover_mode 设置为 RESTART_ONLY，则可以通过在 Retry_interval 内对 scha_control() 函数进行 Retry_count 次调用来重新启动资源或资源组。如果资源监视器超出 Retry_count 次，则尝试重新启动失败。如果监视器调用了 scha_control() 函数以请求故障转移，则该请求也将失败。

LOG_ONLY（探测失败）

如果对于任何数据服务都将 Failover_mode 设置为 LOG_ONLY，则所有的 scha_control() 都将请求重新启动资源或资源组，或者请求将已排除的组故障转移。如果数据服务是基于 DSDL 的数据服务，则在探测完全失败是将记录一条消息，但不重新启动资源。如果在 Retry_interval 内探测完全失败的次数超出 Retry_count 次，则资源监视器将退出、将资源状态设置为 FAULTED 并生成状态消息“应用程序有问题，但未重新启动。探测正在退出。”此时，虽然监视仍处于启用状态，但是在群集管理员修复并重新启动资源之前，资源将一直处于不受监视的状态。

如果将 Failover_mode 设置为 LOG_ONLY 并且数据服务是基于 DSDL 的不受监视的服务，则当进程树中止时，将记录一条消息但不重新启动资源。

如果受监视的数据服务不是基于 DSDL 的数据服务，则恢复行为取决于如何对资源监视器进行编码。如果将 Failover_mode 设置为 LOG_ONLY，则所有的 scha_control() 将请求重新启动资源或资源组，或者请求将失败的组故障转移。

类别： 可选

缺省值： 无

可调： ANYTIME

类型中的每个回调方法的 Fini_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

类型中的每个回调方法的 Init_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果方法本身已在 RTR 文件中声明

可调： ANYTIME

`Load_balancing_policy` (string)
定义所使用的负载平衡策略的字符串。此属性仅用于可伸缩服务。如果在 RTR 文件中声明了 `Scalable` 属性，则 RGM 将自动创建此属性。
`Load_balancing_policy` 可以取以下值：

`Lb_weighted`（缺省值）。根据在 `Load_balancing_weights` 属性中设置的权重在各个节点间分配负载。

`Lb_sticky`。可伸缩服务的给定客户机（由客户机的 IP 地址标识）总是发送到同一群集节点。

`Lb_sticky_wild`。连接到用通配符表示的粘滞服务的 IP 地址的给定客户机 IP 地址总是被发送到相同的群集节点，而忽略此 IP 地址的目标端口号。

类别： 有条件的/可选

缺省值： `Lb_weighted`

可调： `AT_CREATION`

`Load_balancing_weights` (`string_array`)
只用于可缩放资源。如果在 RTR 文件中声明了 `Scalable` 属性，则 RGM 将自动创建此属性。格式为 *weight@node,weight@node*，其中 *weight* 是反映分布到指定 *node* 的相对负载部分的整数。分布到某个节点的负载部分是此节点的权重除以所有权重之和。例如，`1@1,3@2` 指定了节点 1 接收负载的四分之一，节点 2 接收负载的四分之三。空字符串 (“”) 为缺省值，它设置均匀的分配。未明确指定权重的任何节点的缺省权重为 1。

如果在 RTR 文件中未指定 `Tunable` 属性，则此属性的 `Tunable` 值为 `ANYTIME`。更改该属性将仅改变新连接的分配。

类别： 有条件的/可选

缺省值： 空字符串 ("")

可调： `ANYTIME`

类型中的每个回调方法的 `Monitor_check_timeout` (`integer`)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： `ANYTIME`

类型中的每个回调方法的 `Monitor_start_timeout` (`integer`)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

类型中的每个回调方法的 Monitor_stop_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

Monitored_switch (enum)
如果群集管理员通过管理公用程序启用或禁止监视器，则 RGM 将把此属性设置为 Enabled 或 Disabled。如果设置为 Disabled，那么虽然资源本身仍保持联机，但将停止对资源的监视。仅当重新启用监视之后，才会调用 Monitor_start 方法。如果资源没有监视器回调方法，则此属性不存在。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

Network_resources_used (string_array)
资源所使用的逻辑主机名或共享地址网络资源的列表。对于可缩放服务，此属性必须是指存在于单独资源组的共享地址资源。对于失败转移服务，此属性是指存在于相同的资源组的逻辑主机名或共享地址资源。如果在 RTR 文件中声明了 Scalable 属性，则 RGM 将自动创建此属性。如果在 RTR 文件中没有声明 Scalable，Network_resources_used 将不可用，除非在 RTR 文件中显式声明。

如果在 RTR 文件中未指定 Tunable 属性，则此属性的 Tunable 值为 AT_CREATION。

注 – SUNW.Event(5) 手册页介绍了如何为 CRNP 设置此属性。

类别： 有条件的/必需的

缺省值： 无缺省值

可调： AT_CREATION

每个群集节点上的 Num_resource_restarts (integer)
不能直接设置此属性。RGM 将把此属性设置为在过去的 *n* 秒内对此节点上的此资源执行 scha_control、Resource_restart 或 Resource_is_restarted 调用的次数。*n* 为资源的 Retry_interval 属性的值。只要此资源执行了 scha_control 停止，无论停止尝试成功与否，RGM 都会将资源重新启动计数器重置为零 (0)。

如果某个资源类型不声明 Retry_interval 属性，则该类资源的 Num_resource_restarts 属性不可用。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

每个群集节点上的 Num_rg_restarts (integer)
不能直接设置此属性。RGM 将把此属性设置为在过去的 *n* 秒内资源已对其在此节点上的包含资源组执行 scha_control Restart 调用的次数。*n* 为此资源的 Retry_interval 属性的值。如果资源类型不声明 Retry_interval 属性，则该类资源的 Num_rg_restarts 属性不可用。

类别： 请参阅说明

缺省值： 无缺省值

可调： 无

On_off_switch (enum)
如果群集管理员通过管理公用程序启用或禁止资源，则 RGM 将把此属性设置成 Enabled 或 Disabled。如果设置为 Disabled，资源将被脱机并且仅在重新启用该资源后才能运行回调。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

Port_list (string_array)
端口（服务器在其上进行侦听）号列表。每个端口号后附加斜杠 (/)，斜杠后接该端口所使用的协议，例如 Port_list=80/tcp 或 Port_list=80/tcp6,40/udp6。您可以指定以下协议值：

- tcp，用于 TCP IPv4
- tcp6（适用于 TCP IPv6）
- udp，用于 UDP IPv4
- udp6（适用于 UDP IPv6）

如果在 RTR 文件中声明了 Scalable 属性，则 RGM 将自动创建 Port_list。否则，此属性将不可用，除非在 RTR 文件中显式声明。

《Sun Cluster Data Service for Apache Guide for Solaris OS》中介绍了如何为 Apache 设置此属性。

类别： 有条件的/必需的

缺省值： 无缺省值

可调： ANYTIME

类型中的每个回调方法的 Postnet_stop_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

类型中的每个回调方法的 Prenet_start_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

R_description (string)
资源的简单说明。

类别： 可选

缺省值： 空字符串

可调： ANYTIME

Resource_dependencies (string_array)
在同一组或不同组中与 Resource_dependencies 资源具有强依赖性的资源的列表。如果该列表中有任一资源处于脱机状态，则无法启动此资源。如果此资源和列表中的某个资源同时启动，则 RGM 将一直等到列表中的资源启动之后才启动此资源。如果此资源的 Resource_dependencies 列表中的资源不启动，则此资源将保持脱机状态。此资源的列表中的资源可能不启动的原因在于：列表中资源的资源组仍保持脱机状态或处于 START_FAILED 状态。如果此资源由于与无法启动的其他资源组中的资源具有依赖性而保持脱机状态，则此资源的组将进入 PENDING_ONLINE_BLOCKED 状态。

如果此资源与该列表中的资源同时进入脱机状态，则将在停止该列表中的资源之前先停止此资源。但是，即使此资源仍处于联机状态或无法停止，该列表中属于另一不同资源组的资源照样也会停止。除非先禁用此资源，否则无法禁用列表中的资源。

缺省情况下，在资源组中，应用程序资源对网络地址资源具有固有的强依赖性。第 233 页中的 “资源组属性”中的 Implicit_network_dependencies 包含更多信息。

在资源组中，按照依赖顺序，Prenet_start 方法将先于 Start 方法运行。按照依赖顺序，Postnet_stop 方法将在运行 Stop 方法后运行。在不同的资源组中，依赖资源将等待被依赖资源完成 Prenet_start 和 Start 之后再运行 Prenet_start。被依赖资源将等待依赖资源完成 Stop 和 Postnet_stop 之后再运行 Stop。

类别： 可选

缺省值： 空列表

可调： ANYTIME

Resource_dependencies_restart (string_array)
在同一组或不同组中与 Resource_dependencies_restart 资源具有重新启动依赖性的资源的列表。

此属性的功能与 Resource_dependencies 相同，以下情况除外：如果重新启动依赖性列表中的任何资源被重新启动，则重新启动此资源。RGM 将在列表中的资源重新联机之后重新启动此资源。

类别： 可选

缺省值： 空列表

可调： ANYTIME

Resource_dependencies_weak (string_array)
在同一组或不同组中与 Resource_dependencies_weak 资源具有弱依赖性的资源的列表。弱依赖性用于确定方法调用的顺序。RGM 将先调用此列表中资源的 Start 方法，再调用此资源的 Start 方法。RGM 将先调用此资源的 Stop 方法，再调用列表中那些资源的 Stop 方法。如果列表中的资源启动失败或保持脱机状态，该资源将仍然可以启动。

如果此资源与其 Resource_dependencies_weak 列表中的资源同时启动，则 RGM 将一直等到该列表中的资源启动之后再启动此资源。如果列表中的资源不启动（例如列表中资源的资源组仍然保持脱机状态或者列表中的资源处于 START_FAILED 状态时），此资源将启动。当此资源的 Resource_dependencies_weak 列表中的资源启动时，此资源的资源组可能会临时进入 PENDING_ONLINE_BLOCKED 状态。当所有资源已启动或无法启动时，此资源将启动并且此资源的组将重新进入 PENDING_ONLINE 状态。

如果将该资源与列表中的资源同时脱机，则该资源将在列表中的资源停止之前停止。如果该资源保持联机或无法停止，列表中的资源仍会停止。您不能禁用列表中的资源，除非先禁用此资源。

在资源组中，按照依赖顺序，Prenet_start 方法将先于 Start 方法运行。按照依赖顺序，Postnet_stop 方法将在运行 Stop 方法后运行。在不同的资源组中，依赖资源将等待被依赖资源完成 Prenet_start 和 Start 之后再运行 Prenet_start。被依赖资源将等待依赖资源完成 Stop 和 Postnet_stop 之后再运行 Stop。

类别： 可选

缺省值： 空列表

可调： ANYTIME

Resource_name (string)
资源实例的名称。在群集配置中此名称必须唯一，并且在资源创建后该名称无法进行更改。

类别： 必需的

缺省值： 无缺省值

可调： 无

`Resource_project_name` (`string`)
与资源关联的 Solaris 项目名称。使用该属性可以将 Solaris 资源的管理功能（例如 CPU 共享和资源池）应用于群集数据服务。当 RGM 使资源联机时，它将启动此项目名称下的相关进程。如果未指定此属性，将通过包含资源的资源组的 `RG_project_name` 属性获取项目名称（请参见 `rg_properties`(5) 手册页）。如果也未指定 RG_project_name 属性，RGM 将使用预定义的项目名称 `default`。指定的项目名称必须已在项目数据库中（请参见 `projects`(1) 手册页和《System Administration Guide: Solaris Containers-Resource Management and Solaris Zones》）。

仅从 Solaris 9 开始支持此属性。

注 – 对此属性的更改将在下次资源被启动时生效。

类别： 可选

缺省值： 空

可调： `ANYTIME`

每个群集节点上的 `Resource_state` (`enum`)
每个群集节点上由 RGM 确定的资源状态。可能的状态包括 `ONLINE`、`OFFLINE`、`START_FAILED`、`STOP_FAILED`、`MONITOR_FAILED`、`ONLINE_NOT_MONITORED`、`STARTING` 和 `STOPPING`。

不能配置该属性。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

`Retry_count` (`integer`)
如果资源失败，监视器尝试重新启动该资源的次数。如果超出 `Retry_count`，则根据特定数据服务和 `Failover_mode` 属性的设置，监视器可能执行以下操作之一：

- 允许资源组保留在当前的主节点上，即使资源处于有问题的状态
- 请求将资源组故障转移到其他节点上

该属性由 RGM 创建，并且仅当在 RTR 文件中声明此属性后，群集管理员才可以使用此属性。如果在 RTR 文件中指定了缺省值，则此属性是可选的。

如果在 RTR 文件中未指定 `Tunable` 属性，则此属性的 `Tunable` 值为 `WHEN_DISABLED`。

注 – 如果为此属性指定了负值，监视器将无限次尝试重新启动资源。

类别： 有条件的

缺省值： 参见上述说明

可调： WHEN_DISABLED

Retry_interval (integer)
尝试重新启动失败的资源前计算的秒数。资源监视器将此属性与 Retry_count 一起使用。该属性由 RGM 创建，并且仅当在 RTR 文件中声明此属性后，群集管理员才可以使用此属性。如果在 RTR 文件中指定了缺省值，则此属性是可选的。

如果在 RTR 文件中未指定 Tunable 属性，则此属性的 Tunable 值为 When_disabled。

类别： 有条件的

缺省值： 无默认值（参见上述说明）

可调： WHEN_DISABLED

Scalable (boolean)
指明资源是不是可伸缩资源，即资源是否使用 Sun Cluster 软件的网络负载平衡功能。

如果在 RTR 文件中声明了此属性，则 RGM 将为该类型的资源自动创建以下可伸缩服务属性：Affinity_timeout、Load_balancing_policy、Load_balancing_weights、Network_resources_used、Port_list、UDP_affinity 和 Weak_affinity。这些属性具有缺省值，除非在 RTR 文件中对它们进行了显式声明。Scalable 的默认值（在 RTR 文件中声明时）为 TRUE。

如果在 RTR 文件中声明此属性，则不能其指定给除了 AT_CREATION 以外的任何 Tunable 属性。

如果未在 RTR 文件中声明此属性，该资源将不具有可伸缩性，您不能调节此属性，并且 RGM 也不会设置任何可伸缩服务属性。但是，您可以在 RTR 文件中显式声明 Network_resources_used 和 Port_list 属性。这些属性对不可伸缩服务和可伸缩服务都会非常有用。

r_properties(5) 手册页中详细介绍了如何将此资源属性与 Failover 资源类型属性结合使用。

类别： 可选

缺省值： 无缺省值

可调： AT_CREATION

类型中的每个回调方法的 Start_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

每个群集节点上的 Status (enum)
由资源监视器使用 scha_resource_setstatus 命令或 scha_resource_setstatus() 函数设置。可能的值包括 OK、degraded、faulted、unknown 和 offline。当资源进入联机或脱机状态时，如果资源的监视器或方法未设置 Status 值，则 RGM 将自动设置 Status 值。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

每个群集节点上的 Status_msg (string)
由资源监视器在设置 Status 属性时同时设置。当资源进入联机或脱机状态时，如果资源调用的方法未设置此属性，则 RGM 将自动把此属性的值复位为空字符串。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

类型中的每个回调方法的 Stop_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

Thorough_probe_interval (integer)
在两次资源高开销故障探测的调用之间的秒数。该属性由 RGM 创建，并且仅当在 RTR 文件中声明此属性后，群集管理员才可以使用此属性。如果在 RTR 文件中指定了缺省值，则此属性是可选的。

如果在 RTR 文件中未指定 Tunable 属性，则该属性的 Tunable 值为 When_disabled。

类别： 有条件的

缺省值： 无缺省值

可调： WHEN_DISABLED

Type (string)
资源类型，此资源是该资源类型的一个实例。

类别： 必需的

缺省值： 无缺省值

可调： 无

Type_version (string)
指定当前与此资源关联的资源类型的版本。RGM 将自动创建此属性，该属性不能在 RTR 文件中声明。此属性的值与资源类型的 RT_version 属性一样。创建资源时，

并不明确指定 Type_version 属性，尽管它可能显示为资源类型名称的后缀。编辑资源时，可以将 Type_version 属性更改为新值。

此属性的可调性源自以下源：

- 资源类型的当前版本
- RTR 文件中的 #$upgrade_from 指令

类别： 请参阅说明

缺省值： 无缺省值

可调： 请参阅说明

UDP_affinity (boolean)
如果将此属性设置为 TRUE，则将来自给定客户机的所有 UDP 通信流量都发送给当前为该客户机处理所有 TCP 通信流量的同一个服务器节点。

仅当 Load_balancing_policy 的值为 Lb_sticky 或 Lb_sticky_wild 时，此属性才适用。另外，必须将 Weak_affinity 设置为 FALSE。

此属性只用于可伸缩服务。

类别： 可选

缺省值： 无缺省值

可调： WHEN_DISABLED

类型中的每个回调方法的 Update_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

类型中的每个回调方法的 Validate_timeout (integer)
以秒计算的时间段，这段时间过后，RGM 将认为对该方法的调用已失败。对于给定的资源类型，仅为在 RTR 文件中声明的那些方法定义超时属性。

类别： 有条件的/可选

缺省值： 3600（一小时），如果在 RTR 文件中声明了此方法本身

可调： ANYTIME

Weak_affinity (boolean)
如果将此属性设置为 TRUE，此属性将启用客户机的弱关联形式。较弱形式的客户机关联允许将来自给定客户机的连接发送到同一个服务器节点，以下情况除外：

- 服务器侦听器为响应以下操作而启动：故障监视器的重新启动、资源的故障转移或切换转移、节点的重新加入群集等
- 由于群集管理员执行了管理操作，可伸缩资源的 Load_balancing_weights 发生更改

就内存消耗和处理器周期而言，不采用缺省形式而启用较弱的关联性所使用的系统开销较低。

仅当 Load_balancing_policy 的值为 Lb_sticky 或 Lb_sticky_wild 时，此属性才适用。

此属性只用于可伸缩服务。

类别： 可选

缺省值： 无缺省值

可调： WHEN_DISABLED

资源组属性

以下信息介绍了由 Sun Cluster 软件定义的资源组属性。属性值分为以下几类：

- **必需**。群集管理员在使用管理实用程序创建资源组时必须指定值。
- **可选**。如果群集管理员在创建资源组时未指定值，系统将会提供一个默认值。
- **仅限于查询**。不能通过管理工具直接设置。

首先列出的是属性名称，后面是对该属性的说明。

Auto_start_on_new_cluster (boolean)
在形成新群集时，此属性用于控制资源组管理器 (RGM) 是否自动启动资源组。默认值为 TRUE。

如果设置为 TRUE，则同时重新引导群集的所有节点时，RGM 将尝试自动启动资源组以实现 Desired_primaries。

如果设置为 FALSE，则在重新引导群集时资源组将不自动启动。在使用 scswitch 命令或等价的 GUI 指令第一次将资源组手动切换为联机之前，资源组将一直保持脱机状态。联机后，资源组将恢复正常的故障转移行为。

类别： 可选

缺省值： TRUE

可调： ANYTIME

Desired_primaries (integer)
可同时运行组的首选节点数目。

默认值为 1。如果 RG_mode 属性为 Failover，则此属性的值必须不大于 1。如果 RG_mode 属性为 Scalable，则值大于 1 是允许的。

类别： 可选

缺省值： 1

可调： ANYTIME

`Failback` (boolean)
一个布尔值，指示是否在群集成员资格更改时重新计算组处于联机状态的节点集。重新计算将造成资源组管理器使组在优先级较低的节点上脱机，并在优先级较高的节点上联机。

类别： 可选

缺省值： FALSE

可调： ANYTIME

`Global_resources_used` (`string_array`)
表明群集文件系统是否由此资源组中的任何资源使用。群集管理员可以指定的合法值为星号 (*)（用于表示所有全局资源）和空字符串 (“”)（用于表示无全局资源）。

类别： 可选

缺省值： 所有全局资源

可调： ANYTIME

`Implicit_network_dependencies` (`boolean`)
一个布尔值，指示在设置为 `True` 时 RGM 应当强制非网络地址资源对组内的网络地址资源具有隐含强依赖性。这意味着 RGM 将在组内所有其他资源启动之前启动所有网络地址资源，在组内所有其他资源停止之后停止网络地址资源。网络地址资源包括逻辑主机名和共享地址资源类型。

在可伸缩资源组中，此属性不起作用，因为可伸缩资源组不包含任何网络地址资源。

类别： 可选

缺省值： TRUE

可调： ANYTIME

`Maximum_primaries` (`integer`)
组一次可以联机的最大节点数。

如果 `RG_mode` 属性为 `Failover`，则此属性的值必须不大于 1。如果 `RG_mode` 属性为 `Scalable`，则值大于 1 是允许的。

类别： 可选

缺省值： 1

可调： ANYTIME

`Nodelist` (`string_array`)
群集节点列表，在这些节点上可以按优先顺序使组联机。这些节点被称为资源组的潜在主节点或主控节点。

类别： 可选

缺省值： 任意顺序的所有群集节点的列表

可调： ANYTIME

Pathprefix (string)
群集文件系统中，组资源可以编写所需管理文件的目录。某些资源可能需要此属性。使 Pathprefix 对每个资源组都是唯一的。

类别： 可选

缺省值： 空字符串

可调： ANYTIME

Pingpong_interval (integer)
非负整数值（以秒为单位），RGM 使用此属性值确定使以下实例中的资源组联机到何处：

- 在重新配置的情况下
- 导致执行 scha_control GIVEOVER 命令或函数

在重新配置的情况下，资源组在过去 Pingpong_interval 秒内在特定节点上多次联机失败。此失败的原因在于资源的 Start 或 Prenet_start 方法在非零状态下或超时后退出。因此，此节点被认为不适于托管资源组，RGM 将查找其他主控节点。

如果资源在给定节点上执行 scha_control 命令或 scha_control GIVEOVER 命令，因而导致其资源组故障转移到其他节点上，则第一个节点（在其上运行过 scha_control）不能作为同一资源执行另一个 scha_control GIVEOVER 的目标节点，直至经过 Pingpong_interval 秒之后。

类别： 可选

缺省值： 3600（一小时）

可调： ANYTIME

Resource_list (string_array)
组中包含的资源的列表。群集管理员不直接设置此属性。而是当群集管理员在资源组中添加或删除资源时，由 RGM 更新此属性。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

RG_affinities (string)
RGM 将尝试找到其他给定资源组的当前主控节点上的资源组（正关联），或尝试找到给定资源组的非当前主控节点上的资源组（负关联）。

您可以将 RG_affinities 设置为以下字符串：

- ++，即较强正关联
- +，即较弱正关联
- -，即较弱负关联
- --，即强负关联
- +++，带故障转移托管的较强正关联

例如，RG_affinities=+RG2,--RG3 表示此资源组具有与 RG2 的较弱正关联和 RG3 的较强负关联。

《Sun Cluster Data Services Planning and Administration Guide for Solaris OS》中的第 2 章 “Administering Data Service Resources”中介绍了如何使用 RG_affinities。

类别： 可选

缺省值： 空字符串

可调： ANYTIME

RG_dependencies (string_array)
可选资源组列表，用于指明在同一节点上使其他组联机或脱机的优先顺序。所有强 RG_dependencies 和 RG_affinities（正负）的图形都不允许包含循环。

例如，假定资源组 RG1 的 RG_dependencies 列表中列有资源组 RG2，即 RG1 与 RG2 具有资源组依赖性。以下列表汇总了此资源组依赖性的影响：

- 当某个节点加入群集时，必须等该节点上的所有 Boot 方法在 RG2 中的资源上运行完之后，才能在 RG1 中的资源上运行该节点的 Boot 方法。
- 如果 RG1 和 RG2 同时在同一个节点上处于 PENDING_ONLINE 状态，则在 RG2 中的所有资源运行完其启动方法之后，RG1 中的资源才运行启动方法（Prenet_start 或 Start）。
- 如果 RG1 和 RG2 同时在同一个节点上处于 PENDING_OFFLINE 状态，则在 RG1 中的所有资源运行完其停止方法之后，RG2 中的资源才运行停止方法（Stop 或 Postnet_stop）。
- 如果切换主节点会使 RG1 在任意一个节点上保持联机而使 RG2 在所有节点上都保持脱机，则尝试切换 RG1 或 RG2 的主节点将会失败。scswitch(1M) 和 scsetup(1M) 手册页包含更多信息。
- 如果在 RG2 上将 Desired_primaries 设置为零，则不允许在 RG1 上将 Desired_primaries 属性设置为大于零的值。
- 如果在 RG2 上将 Auto_start_on_new_cluster 属性设置为 FALSE，则不允许在 RG1 上将 Auto_start_on_new_cluster 属性设置为 TRUE。

类别： 可选

缺省值： 空列表

可调： ANYTIME

RG_description (string)
资源组的简单说明。

类别： 可选

缺省值： 空字符串

可调： ANYTIME

RG_is_frozen (boolean)
一个布尔值，指示是否正在切换转移资源组所依赖的全局设备。如果将此属性设置为 TRUE，则将切换转移全局设备。如果将此属性设置为 FALSE，则表明没有切换转移全局设备。资源组依赖于其 Global_resources_used 属性中指定的全局设备。

请不要直接设置 RG_is_frozen 属性。RGM 将在全局设备的状态发生更改时更新 RG_is_frozen 属性。

类别： 可选

缺省值： 无缺省值

可调： 无

RG_mode (enum)
指明资源组是故障转移资源组还是可伸缩资源组。如果值为 Failover，则 RGM 将把该组的 Maximum_primaries 属性设置为 1 并将资源组限制为由单个节点控制。

如果此属性的值为 Scalable，则 RGM 允许为 Maximum_primaries 属性设置大于 1 的值，因此，该组可同时受多个节点控制。RGM 不允许将 Failover 属性为 True 的资源添加到 RG_mode 为 Scalable 的资源组中。

如果 Maximum_primaries 为 1，则默认值为 Failover。如果 Maximum_primaries 的值大于 1，则缺省值为 Scalable。

类别： 可选

缺省值： 取决于 Maximum_primaries 的值。

可调： 无

RG_name (string)
资源组的名称。此属性是必需的，并且在群集中必须唯一。

类别： 必需的

缺省值： 无缺省值

可调： 无

RG_project_name (string)
与资源组相关的 Solaris 项目名称（请参见 projects(1) 手册页）。使用该属性可以将 Solaris 资源的管理功能（例如 CPU 共享和资源池）应用于群集数据服务。RGM 使资源组联机时，将为不具有 Resource_project_name 属性集的资源启动此项目名称下的相关进程（请参见 r_properties(5) 手册页）。指定的项目名称必须已在项目数据库中（请参见 projects(1) 手册页和《System Administration Guide: Solaris Containers-Resource Management and Solaris Zones》）。

仅从 Solaris 9 开始支持此属性。

注 – 对此属性的更改将在下次资源被启动时生效。

类别： 可选

缺省值： 文本字符串 "default"

可调： ANYTIME

每个群集节点上的 RG_state (enum)
由 RGM 将其设置为 UNMANAGED、ONLINE、OFFLINE、PENDING_ONLINE、PENDING_OFFLINE、ERROR_STOP_FAILED、ONLINE_FAULTED 或 PENDING_ONLINE_BLOCKED，以说明每个群集节点上的组的状态。

不能配置该属性。不过，您可以通过运行 scswitch 命令或者使用等效的 scsetup 或 SunPlex Manager 命令间接设置此属性。如果组不在 RGM 的控制下，可以处于 UNMANAGED 状态。

以下说明汇总了所有状态。

注 – 状态仅适于单个节点，UNMANAGED 状态除外，该状态适于所有节点。例如，资源组可能在节点 A 上 OFFLINE，但在节点 B 上 PENDING_ONLINE。

UNMANAGED	新建资源组的初始状态，或者先前受管理的资源组的状态。不是尚未对组中的资源运行 Init 方法，就是已经对组中的资源运行了 Fini 方法。 该组不受 RGM 管理。
ONLINE	已在节点上启动资源组。换言之，适于每个资源的启动方法 Prenet_start、Start 和 Monitor_start 已在组中所有已启用的资源上成功执行。
OFFLINE	已在节点上停止资源组。换言之，适于每个资源的停止方法 Monitor_stop、Stop 和 Postnet_stop 已在组中所有已启用的资源上成功执行。此状态还适于在节点上首次启动资源组之前。
PENDING_ONLINE	正在节点上启动资源组。适于每个资源的启动方法 Prenet_start、Start 和 Monitor_start 正在组中已启用的资源上执行。
PENDING_OFFLINE	正在节点上停止资源组。适于每个资源的停止方法 Monitor_stop、Stop 和 Postnet_stop 正在组中已启用的资源上执行。
ERROR_STOP_FAILED	资源组内的一个或多个资源无法成功停止，并且处于 Stop_failed 状态。组中的其他资源可能保持联机状态，也可能保持脱机状态。在 ERROR_STOP_FAILED 状态被清除之前，不允许在任何节点上启动资源组。

	必须使用管理命令（例如 scswitch -c）手动中止 Stop_failed 资源并将其状态重置为 OFFLINE。
ONLINE_FAULTED	资源组曾处于 PENDING_ONLINE 状态，并已在此节点上启动。但是，一个或多个资源在 Start_failed 状态或 Faulted 状态下停止。
PENDING_ONLINE_BLOCKED	资源组无法完全启动，因为该资源组内的一个或多个资源与其他资源组中的资源具有未满足的强资源依赖性。此类资源将保持 OFFLINE 状态。满足资源依赖性之后，资源组将自动返回到 PENDING_ONLINE 状态。

类别： 仅限于查询

缺省值： 无缺省值

可调： 无

RG_system (boolean)

如果资源组的 RG_system 属性为 TRUE，将限制对该资源组和资源组包含的资源执行的特定操作。此限制旨在防止意外修改或删除重要的资源组和资源。此属性仅影响 scrgadm 和 scswitch 命令。对 scha_control(1HA) 和 scha_control(3HA) 的操作不受影响。

在资源组（或资源组的资源）上执行受限制的操作之前，您必须先将该资源组的 RG_system 属性设置为 FALSE。当修改或删除支持群集服务的资源组时，或者当修改或删除此类资源组中包含的资源时，请慎用此属性。

操作	示例
删除资源组	**scrgadm -r -g RG1**
编辑资源组属性（RG_system 除外）	**scrgadm -c -t RG1 -y nodelist=...**
将资源添加到资源组	**scrgadm -a -j R1 -g RG1**
从资源组中删除资源	**scrgadm -r -j R1 -g RG1**
编辑资源组中某个资源的属性	**scrgadm -c -j R1**
使资源组脱机	**scswitch -F -g RG1**
管理资源组	**scswitch -o -g RG1**
取消管理资源组	**scswitch -u -g RG1**
启用资源	**scswitch -e -j R1**
启用对资源的监视	**scswitch -e -M -j R1**

操作	示例
禁用资源	**scswitch -n -j R1**
禁用对资源的监视	**scswitch -n -M -j R1**

如果资源组的 RG_system 属性为 TRUE，则能编辑的该资源组属性只有 RG_system 属性本身。换言之，编辑 RG_system 属性永不受限制。

类别： 可选

缺省值： FALSE

可调： ANYTIME

资源特性属性

本节介绍了可用来更改系统定义的属性或用来创建扩展属性的资源属性。

注意 – 您不能将 Null 或空字符串 ("") 指定为 boolean、enum 或 int 类型的默认值。

首先列出的是属性名称，后面是对该属性的说明。

Array_maxsize
对于 stringarray 类型所允许的数组元素的最大数。

Array_minsize
对于 stringarray 类型所允许的数组元素的最小数。

Default
表明属性的缺省值。

Description
一个字符串注释，用于对属性作简单说明。对于系统定义的属性，不能在 RTR 文件中设置 Description 属性。

Enumlist
对于 enum 类型，为该属性所允许的一组字符串值。

Extension
如果使用该属性，则表明 RTR 文件条目声明了一个由资源类型的实现所定义的扩展属性，否则，此项为系统定义的属性。

Max
对于 int 类型所允许的属性的最大值。

Maxlength
对于 string 和 stringarray 类型所允许的最大字符串长度。

Min
对于 int 类型所允许的属性的最小值。

Minlength
对于 string 和 stringarray 类型所允许的最小字符串长度。

Property
资源属性的名称。

Tunable
表明群集管理员何时可以设置某个资源中的此属性值。设置为 NONE 或 FALSE 可以防止群集管理员设置属性。允许群集管理员调节属性的值为 TRUE、ANYTIME（随时）、AT_CREATION（仅在创建资源时）或 WHEN_DISABLED（禁用资源时）。要建立其他条件（例如“禁用监视时”或“脱机时”），请将此属性设置为 ANYTIME 并在 Validate 方法中验证资源的状态。

不同的标准资源属性有不同的默认值，如下面的条目所示。如果没有在 RTR 文件中进行指定，则调节扩展属性的默认设置为 TRUE (ANYTIME)。

属性的类型
允许的类型为 string、boolean、integer、enum 和 stringarray。对于系统定义的属性，您不能在 RTR 文件条目中设置类型属性。类型决定了可接受的属性值和在 RTR 文件项中允许的特定于类型的属性。enum 类型是一组字符串值。

附录 B

数据服务样例代码列表

本附录提供数据服务样例中的每个方法的完整代码，还列出了资源类型注册 (RTR) 文件的内容。

本附录包括以下主题：

资源类型注册文件列表

RTR 文件包含资源和资源类型属性声明，这些声明用于定义当群集管理员注册数据服务时数据服务的初始配置。

示例 B–1 SUNW.Sample RTR 文件

```
#
# Copyright (c) 1998-2005 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#
```

示例 **B–1** SUNW.Sample RTR 文件 （续）

```
#pragma ident   "@(#)SUNW.sample   1.1   00/05/24 SMI"

Resource_type = "sample";
Vendor_id = SUNW;
RT_description = "Domain Name Service on Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;

RT_basedir=/opt/SUNWsample/bin;
Pkglist = SUNWsample;

Start              = dns_svc_start;
Stop               = dns_svc_stop;

Validate           = dns_validate;
Update             = dns_update;

Monitor_start      = dns_monitor_start;
Monitor_stop       = dns_monitor_stop;
Monitor_check      = dns_monitor_check;

# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.
#
# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.

# The MIN value for all method timeouts is set to 60 seconds. This
# prevents administrators from setting shorter timeouts, which do not
# improve switchover/failover performance, and can lead to undesired
# RGM actions (false failovers, node reboot, or moving the resource group
# to ERROR_STOP_FAILED state, requiring operator intervention). Setting
# too-short method timeouts leads to a *decrease* in overall availability
# of the data service.
{
        PROPERTY = Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
        DEFAULT=300;
}
{
```

示例 **B–1** SUNW.Sample RTR 文件 （续）

```
        PROPERTY = Update_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Thorough_Probe_Interval;
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Set Retry_interval as a multiple of 60 since it is converted from seconds
# to minutes, rounding up. For example, a value of 50 (seconds)
# is converted to 1 minute. Use this property to time the number of
# retries (Retry_count).
{
        PROPERTY = Retry_interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

{
        PROPERTY = Network_resources_used;
        TUNABLE = AT_CREATION;
        DEFAULT = "";
}

#
# Extension Properties
#
```

示例 B–1 SUNW.Sample RTR 文件 （续）

```
# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, DNS, specify the path of the DNS configuration file on
# PXFS (typically named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "The Configuration Directory Path";
}

# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 30;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Start 方法代码列表

当包含数据服务资源的资源组在群集节点上被联机后或者该资源被启用时，RGM 将在该群集节点上运行 Start 方法。在样例应用程序中，Start 方法激活了该节点上的 in.named (DNS) 守护进程。

示例 B–2 dns_svc_start 方法

```
#!/bin/ksh
#
# Start Method for HA-DNS.
#
# This method starts the data service under the control of PMF. Before starting
# the in.named process for DNS, it performs some sanity checks. The PMF tag for
# the data service is $RESOURCE_NAME.named. PMF tries to start the service a
# specified number of times (Retry_count) and if the number of attempts exceeds
# this value within a specified interval (Retry_interval) PMF reports a failure
# to start the service. Retry_count and Retry_interval are both properties of the
# resource set in the RTR file.


#pragma ident   "@(#)dns_svc_start   1.1   00/05/24 SMI"

################################################################################
```

示例 **B–2** dns_svc_start 方法 （续）

```
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}



##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
```

示例 **B–2** dns_svc_start 方法 （续）

```
# Get the value of the Confdir property of the resource in order to start
# DNS. Using the resource name and the resource group entered, find the value of
# Confdir value set by the cluster administrator when adding theresource.
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get returns the "type" as well as the "value" for the extension
# properties. Get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Check if $CONFIG_DIR is accessible.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} Directory $CONFIG_DIR missing or not mounted"
   exit 1
fi

# Change to the $CONFIG_DIR directory in case there are relative
# path names in the data files.
cd $CONFIG_DIR

# Check that the named.conf file is present in the $CONFIG_DIR directory.
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} File $CONFIG_DIR/named.conf is missing or empty"
   exit 1
fi

# Get the value for Retry_count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the value for Retry_interval from the RTR file. Convert this value, which is in
# seconds, to minutes for passing to pmfadm. Note that this is a conversion with
# round-up, for example, 50 seconds rounds up to one minute.
((RETRY_INTRVAL = `scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM? 60))

# Start the in.named daemon under the control of PMF. Let it crash and restart
# up to $RETRY_COUNT times in a period of $RETRY_INTERVAL; if it crashes
# more often than that, PMF will cease trying to restart it. If there is a
# process already registered under the tag <$PMF_TAG>, then, PMF sends out
# an alert message that the process is already running.
echo "Retry interval is "$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Log a message indicating that HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Stop 方法代码列表

当包含 HA-DNS 资源的资源组在群集节点上被脱机后或者该资源被禁用时，RGM 将在该群集节点上运行 Stop 方法。此方法将停止该节点上的 in.named (DNS) 守护进程。

示例 B–3 dns_svc_stop 方法

```
#!/bin/ksh
#
# Stop method for HA-DNS
#
# Stop the data service using PMF. If the service is not running the
# method exits with status 0 as returning any other value puts the resource
# in STOP_FAILED state.


#pragma ident   "@(#)dns_svc_stop   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done

}
```

示例 **B–3** dns_svc_stop 方法 （续）

```
#############################################################################
# MAIN
#
#############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Obtain the Stop_timeout value from the RTR file.
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAM?

# Attempt to stop the data service in an orderly manner using a SIGTERM
# signal through PMF. Wait for up to 80% of the Stop_timeout value to
# see if SIGTERM is successful in stopping the data service. If not, send SIGKILL
# to stop the data service. Use up to 15% of the Stop_timeout value to see
# if SIGKILL is successful. If not, there is a failure and the method exits with
# non-zero status. The remaining 5% of the Stop_timeout is for other uses.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))

# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG.named; then
   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"

      # Since the data service did not stop with a SIGTERM signal, use
      # SIGKILL now and wait for another 15% of the total timeout value.
      pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESSFUL"

          exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
```

示例 B–3 dns_svc_stop 方法 （续）

```
   logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
           "HA-DNS is not started"

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service in STOP_FAILED State.
   exit 0
fi

# Successfully stopped DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

gettime 实用程序代码列表

gettime 实用程序是由 PROBE 程序使用的 C 程序，用于跟踪探测重新启动之间的已过时间。您必须编译此程序并将其放入回调方法所在的目录中，即 RT_basedir 属性指向的目录。

示例 B–4 gettime.c 实用程序

```
# This utility program, used by the probe method of the data service, tracks
# the elapsed time in seconds from a known reference point (epoch point). It
# must be compiled and placed in the same directory as the data service callback
# methods (RT_basedir).

#pragma ident   "@(#)gettime.c   1.1   00/05/24 SMI"

#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
{
    printf("%d\n", time(0));
    exit(0);
}
```

PROBE 程序代码列表

PROBE 程序使用 nslookup 命令检查数据服务的可用性（请参见 nslookup(1M) 手册页）。Monitor_start 回调方法用于启动此程序，Monitor_stop 回调方法用于停止此程序。

示例 B–5 dns_probe 程序

```
#!/bin/ksh
#pragma ident   "@(#)dns_probe   1.1   00/04/19 SMI"
#
# Probe method for HA-DNS.
#
# This program checks the availability of the data service using nslookup, which
# queries the DNS server to look for the DNS server itself. If the server
# does not respond or if the query is replied to by some other server,
# then the probe concludes that there is some problem with the data service
# and fails the service over to another node in the cluster. Probing is done
# at a specific interval set by THOROUGH_PROBE_INTERVAL in the RTR file.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
```

示例 **B–5** dns_probe 程序 （续）

```
        done
}

#############################################################################
# restart_service ()
#
# This function tries to restart the data service by calling the Stop method
# followed by the Start method of the dataservice. If the dataservice has
# already died and no tag is registered for the dataservice under PMF,
# then this function fails the service over to another node in the cluster.
#
function restart_service
{
        # To restart the dataservice, first, verify that the
        # dataservice itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the dataservice is still registered under
                # PMF, first stop the dataservice and start it back up again.
                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                STOP_METHOD=`scha_resource_get -O STOP \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                                "${ARGV0} Stop method failed."
                        return 1
                fi

                # Obtain the Start method name and the START_TIMEOUT value for
                # this resource.
                START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                START_METHOD=`scha_resource_get -O START \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                                "${ARGV0} Start method failed."
                        return 1
                fi

        else
                # The absence of the TAG for the dataservice
```

示例 **B–5** dns_probe 程序　（续）

```
                # implies that the dataservice has already
                # exceeded the maximum retries allowed under PMF.
                # Therefore, do not attempt to restart the
                # dataservice again, but try to failover
                # to another node in the cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}

#############################################################################
# decide_restart_or_failover ()
#
# This function decides the action to be taken upon the failure of a probe:
# restart the data service locally or fail over to another node in the cluster.
#
function decide_restart_or_failover
{

   # Check if this is the first restart attempt.
   if [ $retries -eq 0 ]; then
         # This is the first failure. Note the time of
         # this first attempt.
         start_time=`$RT_BASEDIR/gettimè
         retries=`expr $retries + 1`
         # Because this is the first failure, attempt to restart
         # the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart data service."
            exit 1
         fi
   else
      # This is not the first failure
      current_time=`$RT_BASEDIR/gettimè
      time_diff=`expr $current_time - $start_timè
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # This failure happened after the time window
         # elapsed, so reset the retries counter,
         # slide the window, and do a retry.
         retries=1
         start_time=$current_time
         # Because the previous failure occurred more than
         # Retry_interval ago, attempt to restart the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
```

示例 **B–5** dns_probe 程序 （续）

```
      elif [ $retries -ge $RETRY_COUNT ]; then
         # Still within the time window,
         # and the retry counter expired, so fail over.
         retries=0
         scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
             -R $RESOURCE_NAME
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failover attempt failed."
            exit 1
         fi
      else
         # Still within the time window,
         # and the retry counter has not expired,
         # so do another retry.
         retries=`expr $retries + 1`
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      fi
fi
}

##############################################################################
# MAIN
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# The interval at which probing is to be done is set in the system defined
# property THOROUGH_PROBE_INTERVAL. Obtain the value of this property with
# scha_resource_get
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?

# Obtain the timeout value allowed for the probe, which is set in the
# PROBE_TIMEOUT extension property in the RTR file. The default timeout for
# nslookup is 1.5 minutes.
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Probe_timeout`
PROBE_TIMEOUT=`echo $probe_timeout_info | awk '{print $2}'`
```

示例 **B–5** dns_probe 程序 （续）

```
# Identify the server on which DNS is serving by obtaining the value
# of the NETWORK_RESOURCES_USED property of the resource.
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the retry count value from the system defined property Retry_count
RETRY_COUNT =`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the retry interval value from the system defined property
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the full path for the gettime utility from the
# RT_basedir property of the resource type.
RT_BASEDIR=scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# The probe runs in an infinite loop, trying nslookup commands.
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep for a
   # duration of <THOROUGH_PROBE_INTERVAL>
   sleep $PROBE_INTERVAL

   # Run the probe, which queries the IP address on
   # which DNS is serving.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
           > $DNSPROBEFILE 2>&1

   retcode=$?
        if [ retcode -ne 0 ]; then
                probefail=1
        fi

   # Make sure that the reply to nslookup command comes from the HA-DNS
   # server and not from another name server listed in the
   # /etc/resolv.conf file.
   if [ $probefail -eq 0 ]; then
      # Get the name of the server that replied to the nslookup query.
                 SERVER=` awk ' $1=="Server:" {print $2 }' \
                 $DNSPROBEFILE | awk -F. ' { print $1 } ' `
              if [ -z "$SERVER" ];
                  then
                      probefail=1
```

示例 **B–5** dns_probe 程序　（续）

```
                else
                        if [ $SERVER != $DNS_HOST ]; then
                                probefail=1
                        fi
                fi
        fi

   # If the probefail variable is not set to 0, either the nslookup command
   # timed out or the reply to the query was came from another server
   # (specified in the /etc/resolv.conf file). In either case, the DNS server is
   # not responding and the method calls decide_restart_or_failover,
   # which evaluates whether to restart the data service or to fail it over
   # to another node.

   if [ $probefail -ne 0 ]; then
         decide_restart_or_failover
   else
         logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
         "${ARGV0} Probe for resource HA-DNS successful"
   fi
done
```

Monitor_start 方法代码列表

此方法用来启动数据服务的 PROBE 程序。

示例 **B–6** dns_monitor_start 方法

```
#!/bin/ksh
#
# Monitor start Method for HA-DNS.
#
# This method starts the monitor (probe) for the data service under the
# control of PMF. The monitor is a process that probes the data service
# at periodic intervals and if there is a problem restarts it on the same node
# or fails it over to another node in the cluster. The PMF tag for the
# monitor is $RESOURCE_NAME.monitor.

#pragma ident   "@(#)dns_monitor_start   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt
```

示例 **B–6** dns_monitor_start 方法 （续）

```
        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
          logger -p ${SYSLOG_FACILITY}.err \
                   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done

}

##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the data service.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Start the probe for the data service under PMF. Use the infinite retries
# option to start the probe. Pass the resource name, group, and type to the
# probe method.
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
```

示例 B–6 dns_monitor_start 方法 （续）

```
    -T $RESOURCETYPE_NAME

# Log a message indicating that the monitor for HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} Monitor for HA-DNS successfully started"
fi
exit 0
```

Monitor_stop 方法代码列表

此方法用来停止数据服务的 PROBE 程序。

示例 B–7 dns_monitor_stop 方法

```
#!/bin/ksh
# Monitor stop method for HA-DNS
# Stops the monitor that is running using PMF.

#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
```

示例 B–7 dns_monitor_stop 方法 （续）

```
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done
}

#############################################################################
# MAIN
#
#############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
          "${ARGV0} Could not stop monitor for resource " \
          $RESOURCE_NAME
           exit 1
   else
      # Could successfully stop the monitor. Log a message.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
          "${ARGV0} Monitor for resource " $RESOURCE_NAME \
          " successfully stopped"
   fi
fi
exit 0
```

Monitor_check 方法代码列表

此方法用于验证 Confdir 属性指向的目录是否存在。只要 PROBE 方法将数据服务故障转移到新节点上，以及要检查节点是不是潜在主控节点时，RGM 都将调用 Monitor_check。

示例 B–8 dns_monitor_check 方法

```
#!/bin/ksh#
# Monitor check  Method for DNS.
```

示例 **B–8** dns_monitor_check 方法 （续）

```
#
# The RGM calls this method whenever the fault monitor fails the data service
# over to a new node. Monitor_check calls the Validate method to verify
# that the configuration directory and files are available on the new node.

#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
   typeset opt

   while getopts 'R:G:T:' opt
   do
      case "$opt" in

      R)
      # Name of the DNS resource.
      RESOURCE_NAME=$OPTARG
      ;;

      G)
      # Name of the resource group in which the resource is
      # configured.
      RESOURCEGROUP_NAME=$OPTARG
      ;;

      T)
      # Name of the resource type.
      RESOURCETYPE_NAME=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
      "ERROR: Option $OPTARG unknown"
      exit 1
      ;;
      esac
   done

}

################################################################################
# MAIN
################################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
```

示例 **B–8** dns_monitor_check 方法 （续）

```
# Parse the arguments that have been passed to this method.
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Obtain the full path for the Validate method from
# the RT_basedir property of the resource type.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the name of the Validate method for this resource.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the value of the Confdir property in order to start the
# data service. Use the resource name and the resource group entered to
# obtain the Confdir value set at the time of adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the type as well as the value for extension
# properties. Use awk to get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Call the validate method so that the dataservice can be failed over
# successfully to the new node.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
-T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

# Log a message indicating that monitor check was successful.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS successful."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS not successful."
   exit 1
fi
```

Validate 方法代码列表

此方法用于验证 Confdir 属性指向的目录是否存在。创建数据服务时，RGM 将调用此方法。群集管理员更新数据服务属性时，RGM 也将调用此方法。只要故障监视器将数据服务故障转移到新节点上，Monitor_check 方法就将调用此方法。

示例 B–9 dns_validate 方法

```
#!/bin/ksh
# Validate method for HA-DNS.
# This method validates the Confdir property of the resource. The Validate
# method gets called in two scenarios. When the resource is being created and
# when a resource property is getting updated. When the resource is being
# created, this method gets called with the -c flag and all the system-defined
# and extension properties are passed as command-line arguments. When a resource
# property is being updated, the Validate method gets called with the -u flag,
# and only the property/value pair of the property being updated is passed as a
# command-line argument.
#
# ex: When the resource is being created command args will be
#
# dns_validate -c -R <..> -G <...> -T <..> -r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# when the resource property is being updated
#
# dns_validate -u -R <..> -G <...> -T <..> -r <sys-prop_being_updated=value>
#   OR
# dns_validate -u -R <..> -G <...> -T <..> -x <extn-prop_being_updated=value>

#pragma ident   "@(#)dns_validate   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
   typeset opt

   while getopts 'cur:x:g:R:T:G:' opt
   do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                r)
                        #The method is not accessing any system defined
                        #properties, so this is a no-op.
                        ;;
                g)
                        # The method is not accessing any resource group
```

示例 **B–9** dns_validate 方法　（续）

```
                        # properties, so this is a no-op.
                        ;;
                c)
                        # Indicates the Validate method is being called while
                        # creating the resource, so this flag is a no-op.
                        ;;
                u)
                        # Indicates the updating of a property when the
                        # resource already exists. If the update is to the
                        # Confdir property then Confdir should appear in the
                        # command-line arguments. If it does not, the method must
                        # look for it specifically using scha_resource_get.
                        UPDATE_PROPERTY=1
                        ;;
                x)
                        # Extension property list. Separate the property and
                        # value pairs using "=" as the separator.
                        PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                        VAL=echo $OPTARG | awk -F= '{print $2}'`

                        # If the Confdir extension property is found on the
                        # command line, note its value.
                        if [ $PROPERTY == "Confdir" ];
                        then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                        fi
                        ;;
                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        "ERROR: Option $OPTARG unknown"
                        exit 1
                        ;;
                esac
   done
}

##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Set the Value of CONFDIR to null. Later, this method retrieves the value
# of the Confdir property from the command line or using scha_resource_get.
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0
```

示例 **B–9** dns_validate 方法 （续）

```
# Parse the arguments that have been passed to this method.
parse_args "$@"

# If the validate method is being called due to the updating of properties
# try to retrieve the value of the Confdir extension property from the command
# line. Otherwise, obtain the value of Confdir using scha_resource_get.
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
   config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
       -G $RESOURCEGROUP_NAME Confdir`
   CONFDIR=`echo $config_info | awk '{print $2}'`
fi

# Verify that the Confdir property has a value. If not there is a failure
# and exit with status 1.
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
       "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
   exit 1
fi

# Now validate the actual Confdir property value.

# Check if $CONFDIR is accessible.
if [ ! -d $CONFDIR ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
            "${ARGV0} Directory $CONFDIR missing or not mounted"
        exit 1
fi

# Check that the named.conf file is present in the Confdir directory.
if [ ! -s $CONFDIR/named.conf ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
        exit 1
fi

# Log a message indicating that the Validate method was successful.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

Update 方法代码列表

RGM 调用 Update 方法通知正在运行的资源其属性已更改。

示例 B–10 dns_update 方法

```
#!/bin/ksh
# Update method for HA-DNS.
# The actual updates to properties are done by the RGM. Updates affect only
# the fault monitor so this method must restart the fault monitor.

#pragma ident   "@(#)dns_update   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done
}
################################################################################
# MAIN
################################################################################
export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
```

示例 **B–10** dns_update 方法　（续）

```
# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the resource.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# When the Update method is called, the RGM updates the value of the property
# being updated. This method must check if the fault monitor (probe)
# is running, and if so, kill it and then restart it.
if pmfadm -q $PMF_TAG.monitor; then

# Kill the monitor that is running already
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                    "${ARGV0} Could not stop the monitor"
                exit 1
        else
                # Could successfully stop DNS. Log a message.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully stopped"
        fi
   # Restart the monitor.
   pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
      -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCETYPE_NAME
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                  "${ARGV0} Could not restart monitor for HA-DNS "
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully restarted"
   fi
fi
exit 0
```

附录 C

DSDL 样例资源类型代码列表

本附录列出了 SUNW.xfnts 资源类型中的每个方法的完整代码。它包括 xfnts.c 的列表，该列表包含由回调方法调用的子例行程序的代码。第 8 章详细介绍了资源类型样例 SUNW.xfnts。

本附录包括以下主题：

xfnts.c 文件列表

此文件用于实现由 SUNW.xfnts 方法调用的子例行程序。

示例 C–1 xfnts.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
 * to probe the health of the data service. The probe just returns either
```

示例 **C–1** xfnts.c （续）

```
 * success or failure. Action is taken based on this returned value in the
 * method found in the file xfnts_probe.c
 *
 */

#pragma ident "@(#)xfnts.c 1.47 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <scha.h>
#include <rgm/libdsdev.h>
#include <errno.h>
#include "xfnts.h"

/*
 * The initial timeout allowed for the HAXFS data service to
 * be fully up and running. We will wait for 3 % (SVC_WAIT_PCT)
 * of the start_timeout time before probing the service.
 */
#define   SVC_WAIT_PCT      3

/*
 * We need to use 95% of probe_timeout to connect to the port and the
 * remaining time is used to disconnect from port in the svc_probe function.
 */
#define   SVC_CONNECT_TIMEOUT_PCT      95

/*
 * SVC_WAIT_TIME is used only during starting in svc_wait().
 * In svc_wait() we need to be sure that the service is up
 * before returning, thus we need to call svc_probe() to
 * monitor the service. SVC_WAIT_TIME is the time between
 * such probes.
 */
#define   SVC_WAIT_TIME      5

/*
 * This value will be used as disconnect timeout, if there is no
 * time left from the probe_timeout.
 */
#define   SVC_DISCONNECT_TIMEOUT_SECONDS      2


/*
 * svc_validate():
 *
```

示例 **C–1** xfnts.c （续）

```
 * Do HA-XFS specific validation of the resource configuration.
 *
 * svc_validate will check for the following
 * 1. Confdir_list extension property
 * 2. fontserver.cfg file
 * 3. xfs binary
 * 4. port_list property
 * 5. network resources
 * 6. other extension properties
 *
 * If any of the above validation fails then, Return > 0 otherwise return 0 for
 * success
 */

int
svc_validate(scds_handle_t scds_handle)
{
   char   xfnts_conf[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_net_resource_list_t *snrlp;
   int rc;
   struct stat statbuf;
   scds_port_list_t   *portlist;
   scha_err_t   err;

   /*
    * Get the configuration directory for the XFS dataservice from the
    * confdir_list extension property.
    */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   /* Return an error if there is no confdir_list extension property */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }

   /*
    * Construct the path to the configuration file from the extension
    * property confdir_list. Since HA-XFS has only one configuration
    * we will need to use the first entry of the confdir_list property.
    */
   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /*
    * Check to see if the HA-XFS configuration file is in the right place.
    * Try to access the HA-XFS configuration file and make sure the
    * permissions are set properly
    */
   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suppress lint error because errno.h prototype
```

示例 **C–1** xfnts.c （续）

```
        * is missing void arg
        */
       scds_syslog(LOG_ERR,
           "Failed to access file <%s> : <%s>",
           xfnts_conf, strerror(errno));   /*lint !e746 */
       return (1);
   }

   /*
    * Make sure that xfs binary exists and that the permissions
    * are correct. The XFS binary are assumed to be on the local
    * File system and not on the Global File System
    */
   if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
       scds_syslog(LOG_ERR,
           "Cannot access XFS binary : <%s> ", strerror(errno));
       return (1);
   }

   /* HA-XFS will have only port */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
       scds_syslog(LOG_ERR,
           "Could not access property Port_list: %s.",
          scds_error_string(err));
       return (1); /* Validation Failure */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
       scds_syslog(LOG_ERR,
           "Property Port_list must have only one value.");
       scds_free_port_list(portlist);
       return (1); /* Validation Failure */
   }
#endif

   /*
    * Return an error if there is an error when trying to get the
    * available network address resources for this resource
    */
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
       != SCHA_ERR_NOERR) {
       scds_syslog(LOG_ERR,
           "No network address resource in resource group: %s.",
          scds_error_string(err));
       return (1); /* Validation Failure */
   }

   /* Return an error if there are no network address resources */
   if (snrlp == NULL || snrlp->num_netresources == 0) {
       scds_syslog(LOG_ERR,
           "No network address resource in resource group.");
```

示例 **C–1** xfnts.c （续）

```
      rc = 1;
      goto finished;
   }

   /* Check to make sure other important extension props are set */
   if (scds_get_ext_monitor_retry_count(scds_handle) <= 0)
   {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_count is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }
   if (scds_get_ext_monitor_retry_interval(scds_handle) <= 0) {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_interval is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }

   /* All validation checks were successful */
   scds_syslog(LOG_INFO, "Successful validation.");
   rc = 0;

finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* return result of validation */
}

/*
 * svc_start():
 *
 * Start up the X font server
 * Return 0 on success, > 0 on failures.
 *
 * The XFS service will be started by running the command
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS will be started under PMF. XFS will be started as a single instance
 * service. The PMF tag for the data service will be of the form
 * <resourcegroupname,resourcename,instance_number.svc>. In case of XFS, since
 * there will be only one instance the instance_number in the tag will be 0.
 */

int
svc_start(scds_handle_t scds_handle)
{
   char    xfnts_conf[SCDS_ARRAY_SIZE];
   char   cmd[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_port_list_t    *portlist;
   scha_err_t   err;
```

示例 **C–1** xfnts.c （续）

```
   /* get the configuration directory from the confdir_list property */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /* obtain the port to be used by XFS from the Port_list property */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }

   /*
    * Construct the command to start HA-XFS.
    * NOTE: XFS daemon prints the following message while stopping the XFS
    * "/usr/openwin/bin/xfs notice: terminating"
    * In order to suppress the daemon message,
    * the output is redirected to /dev/null.
    */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);

   /*
    * Start HA-XFS under PMF. Note that HA-XFS is started as a single
    * instance service. The last argument to the scds_pmf_start function
    * denotes the level of children to be monitored. A value of -1 for
    * this parameter means that all the children along with the original
    * process are to be monitored.
    */
   scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

   if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
   }

   scds_free_port_list(portlist);
   return (err); /* return Success/failure status */
}


/*
 * svc_stop():
 *
 * Stop the XFS server
 * Return 0 on success, > 0 on failures.
```

示例 C–1 xfnts.c （续）

```
 *
 * svc_stop will stop the server by calling the toolkit function:
 * scds_pmf_stop.
 */
int
svc_stop(scds_handle_t scds_handle)
{
   scha_err_t   err;

   /*
    * The timeout value for the stop method to succeed is set in the
    * Stop_Timeout (system defined) property
    */
   scds_syslog(LOG_ERR, "Issuing a stop request.");
   err = scds_pmf_stop(scds_handle,
       SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
       scds_get_rs_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Successfully stopped HA-XFS.");
   return (SCHA_ERR_NOERR); /* Successfully stopped */
}

/*
 * svc_wait():
 *
 * wait for the data service to start up fully and make sure it is running
 * healthy
 */

int
svc_wait(scds_handle_t scds_handle)
{
   int rc, svc_start_timeout, probe_timeout;
   scds_netaddr_list_t   *netaddr;

   /* obtain the network resource to use for probing */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
```

示例 **C–1** xfnts.c （续）

```
      return (1);
   }

   /*
    * Get the Start method timeout, port number on which to probe,
    * the Probe timeout value
    */
   svc_start_timeout = scds_get_rs_start_timeout(scds_handle);
   probe_timeout = scds_get_ext_probe_timeout(scds_handle);

   /*
    * sleep for SVC_WAIT_PCT percentage of start_timeout time
    * before actually probing the dataservice. This is to allow
    * the dataservice to be fully up in order to reply to the
    * probe. NOTE: the value for SVC_WAIT_PCT could be different
    * for different data services.
    * Instead of calling sleep(),
    * call scds_svc_wait() so that if service fails too
    * many times, we give up and return early.
    */
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

   do {
      /*
       * probe the data service on the IP address of the
       * network resource and the portname
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Success. Free up resources and return */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

      /*
       * Dataservice is still trying to come up. Sleep for a while
       * before probing again. Instead of calling sleep(),
       * call scds_svc_wait() so that if service fails too
       * many times, we give up and return early.
       */
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }
```

示例 C–1 xfnts.c （续）

```
   /* We rely on RGM to timeout and terminate the program */
   } while (1);

}

/*
 * This function starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as <RG-name,RS-name,instance_number.mon>. The restart option
 * of PMF is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

int
mon_start(scds_handle_t scds_handle)
{
   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling MONITOR_START method for resource <%s>.",
      scds_get_resource_name(scds_handle));

   /*
    * The probe xfnts_probe is assumed to be available in the same
    * subdirectory where the other callback methods for the RT are
    * installed. The last parameter to scds_pmf_start denotes the
    * child monitor level. Since we are starting the probe under PMF
    * we need to monitor the probe process only and hence we are using
    * a value of 0.
    */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Started the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully started Monitor */
}


/*
 * This function stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on <RG-name_RS-name,instance_number.mon>.
 */

int
```

示例 **C–1** xfnts.c （续）

```
mon_stop(scds_handle_t scds_handle)
{

   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");

   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully stopped monitor */
}

/*
 * svc_probe(): Do data service specific probing. Return a float value
 * between 0 (success) and 100(complete failure).
 *
 * The probe does a simple socket connection to the XFS server on the specified
 * port which is configured as the resource extension property (Port_list) and
 * pings the dataservice. If the probe fails to connect to the port, we return
 * a value of 100 indicating that there is a total failure. If the connection
 * goes through and the disconnect to the port fails, then a value of 50 is
 * returned indicating a partial failure.
 */
int
svc_probe(scds_handle_t scds_handle, char *hostname, int port, int
timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
   char   testcmd[2048];
   int    time_used, time_remaining;
   time_t      connect_timeout;


   /*
    * probe the dataservice by doing a socket connection to the port
    * specified in the port_list property to the host that is
    * serving the XFS dataservice. If the XFS service which is configured
    * to listen on the specified port, replies to the connection, then
    * the probe is successful. Else we will wait for a time period set
```

示例 C–1 xfnts.c （续）

```
 * in probe_timeout property before concluding that the probe failed.
 */

/*
 * Use the SVC_CONNECT_TIMEOUT_PCT percentage of timeout
 * to connect to the port
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "Failed to connect to port <%d> of resource <%s>.",
       port, scds_get_resource_name(scds_handle));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Compute the actual time it took to connect. This should be less than
 * or equal to connect_timeout, the time allocated to connect.
 * If the connect uses all the time that is allocated for it,
 * then the remaining value from the probe_timeout that is passed to
 * this function will be used as disconnect timeout. Otherwise, the
 * the remaining time from the connect call will also be added to
 * the disconnect timeout.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Use the remaining time(timeout - time_took_to_connect) to disconnect
 */

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
       "%d seconds during connect operation and exceeded the "
```

示例 **C–1** xfnts.c （续）

```
            "timeout by %d seconds. Attempting disconnect with timeout"
            " %d ",
            connect_timeout,
            abs(time_used),
            SVC_DISCONNECT_TIMEOUT_SECONDS);

        time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
    }

    /*
     * Return partial failure in case of disconnection failure.
     * Reason: The connect call is successful, which means
     * the application is alive. A disconnection failure
     * could happen due to a hung application or heavy load.
     * If it is the later case, don't declare the application
     * as dead by returning complete failure. Instead, declare
     * it as partial failure. If this situation persists, the
     * disconnect call will fail again and the application will be
     * restarted.
     */
    rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
    if (rc != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "Failed to disconnect to port %d of resource %s.",
            port, scds_get_resource_name(scds_handle));
        /* this is a partial failure */
        return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }

    t2 = (hrtime_t)(gethrtime()/1E9);
    time_used = (int)(t2 - t1);
    time_remaining = timeout - time_used;

    /*
     * If there is no time left, don't do the full test with
     * fsinfo. Return SCDS_PROBE_COMPLETE_FAILURE/2
     * instead. This will make sure that if this timeout
     * persists, server will be restarted.
     */
    if (time_remaining <= 0) {
        scds_syslog(LOG_ERR, "Probe timed out.");
        return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }

    /*
     * The connection and disconnection to port is successful,
     * Run the fsinfo command to perform a full check of
     * server health.
     * Redirect stdout, otherwise the output from fsinfo
     * ends up on the console.
     */
    (void) sprintf(testcmd,
        "/usr/openwin/bin/fsinfo -server %s:%d > /dev/null",
```

示例 C–1 xfnts.c （续）

```
        hostname, port);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Checking the server status with %s.", testcmd);
   if (scds_timerun(scds_handle, testcmd, time_remaining,
      SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

      scds_syslog(LOG_ERR,
         "Failed to check server status with command <%s>",
         testcmd);
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }
   return (0);
}
```

xfnts_monitor_check 方法代码列表

此方法用来检验基本资源类型的配置是否有效。

示例 C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_check.c - Monitor Check method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * just make a simple validate check on the service
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
```

示例 C–2 xfnts_monitor_check.c （续）

```
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "monitor_check method "
       "was called and returned <%d>.", rc);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method run as part of monitor check */
   return (rc);
}
```

xfnts_monitor_start 方法代码列表

此方法用于启动 xfnts_probe 方法。

示例 C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_start.c - Monitor Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as RG-name,RS-name.mon. The restart option of PMF
 * is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;
```

示例 C–3 xfnts_monitor_start.c （续）

```
   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = mon_start(scds_handle);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of monitor_start method */
   return (rc);
}
```

xfnts_monitor_stop 方法代码列表

此方法用来停止 xfnts_probe 方法。

示例 C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_stop.c - Monitor Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on RG-name_RS-name.mon.
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int    rc;
```

示例 C–4 xfnts_monitor_stop.c （续）

```
   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = mon_stop(scds_handle);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of monitor stop method */
   return (rc);
}
```

xfnts_probe 方法代码列表

xfnts_probe 方法用于检查应用程序的可用性并确定是故障转移还是重新启动数据服务。xfnts_monitor_start 回调方法用于启动此程序，xfnts_monitor_stop 回调方法用于停止此程序。

示例 C–5 xfnts_probe.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_probe.c - Probe for HA-XFS
 */

#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"


/*
 * main():
```

示例 **C–5** xfnts_probe.c （续）

```
 * Just an infinite loop which sleep()s for sometime, waiting for
 * the PMF action script to interrupt the sleep(). When interrupted
 * It calls the start method for HA-XFS to restart it.
 *
 */

int
main(int argc, char *argv[])
{
   int         timeout;
   int         port, ip, probe_result;
   scds_handle_t      scds_handle;

   hrtime_t      ht1, ht2;
   unsigned long      dt;

   scds_netaddr_list_t *netaddr;
   char   *hostname;

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Get the ip addresses available for this resource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }

   /*
    * Set the timeout from the X props. This means that each probe
    * iteration will get a full timeout on each network resource
    * without chopping up the timeout between all of the network
    * resources configured for this resource.
    */
   timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {

      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
```

示例 **C–5** xfnts_probe.c （续）

```
        */
       (void) scds_fm_sleep(scds_handle,
           scds_get_rs_thorough_probe_interval(scds_handle));

       /*
        * Now probe all ipaddress we use. Loop over
        * 1. All net resources we use.
        * 2. All ipaddresses in a given resource.
        * For each of the ipaddress that is probed,
        * compute the failure history.
        */
       probe_result = 0;
       /*
        * Iterate through the all resources to get each
        * IP address to use for calling svc_probe()
        */
       for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
          /*
           * Grab the hostname and port on which the
           * health has to be monitored.
           */
          hostname = netaddr->netaddrs[ip].hostname;
          port = netaddr->netaddrs[ip].port_proto.port;
          /*
           * HA-XFS supports only one port and
           * hence obtain the port value from the
           * first entry in the array of ports.
           */
          ht1 = gethrtime(); /* Latch probe start time */
          scds_syslog(LOG_INFO, "Probing the service on "
              "port: %d.", port);

          probe_result =
          svc_probe(scds_handle, hostname, port, timeout);

          /*
           * Update service probe history,
           * take action if necessary.
           * Latch probe end time.
           */
          ht2 = gethrtime();

          /* Convert to milliseconds */
          dt = (ulong_t)((ht2 - ht1) / 1e6);

          /*
           * Compute failure history and take
           * action if needed
           */
          (void) scds_fm_action(scds_handle,
              probe_result, (long)dt);
       }   /* Each net resource */
    }    /* Keep probing forever */
```

示例 C–5 xfnts_probe.c （续）

```
}
```

xfnts_start 方法代码列表

当包含数据服务资源的资源组在群集节点上被联机或者该资源已启用时，RGM 将在该群集节点上运行 Start 方法。xfnts_start 方法将激活该节点上的 xfs 守护进程。

示例 C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_start.c - Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * The start method for HA-XFS. Does some sanity checks on
 * the resource settings then starts the HA-XFS under PMF with
 * an action script.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int rc;

   /*
    * Process all the arguments that have been passed to us from RGM
    * and do some initialization for syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Validate the configuration and if there is an error return back */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
```

示例 C–6 xfnts_start.c （续）

```
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }

   /* Start the data service, if it fails return with an error */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

   /* Wait for the service to start up fully */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Calling svc_wait to verify that service has started.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "Successfully started the service.");
   } else {
      scds_syslog(LOG_ERR, "Failed to start the service.");
   }


finished:
   /* Free up the Environment resources that were allocated */
   scds_close(&scds_handle);

   return (rc);
}
```

xfnts_stop 方法代码列表

包含 HA-XFS 资源的资源组在群集节点上脱机或禁用资源后，RGM 将在该节点上运行 Stop 方法。此方法将停止该节点上的 xfs 守护进程。

示例 C–7 xfnts_stop.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_stop.c - Stop method for HA-XFS
 */
```

示例 C–7 xfnts_stop.c （续）

```
#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Stops the HA-XFS process using PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = svc_stop(scds_handle);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of svc_stop method */
   return (rc);
}
```

xfnts_update 方法代码列表

RGM 调用 Update 方法通知运行资源其属性已被更改。设置资源或资源组的管理操作成功后，RGM 将运行 Update。

示例 C–8 xfnts_update.c

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_update.c - Update method for HA-XFS
```

示例 **C–8** xfnts_update.c （续）

```
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Some of the resource properties might have been updated. All such
 * updatable properties are related to the fault monitor. Hence, just
 * restarting the monitor should be enough.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /*
    * check if the Fault monitor is already running and if so stop and
    * restart it. The second parameter to scds_pmf_restart_fm() uniquely
    * identifies the instance of the fault monitor that needs to be
    * restarted.
    */

   scds_syslog(LOG_INFO, "Restarting the fault monitor.");
   result = scds_pmf_restart_fm(scds_handle, 0);
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to restart fault monitor.");
      /* Free up all the memory allocated by scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Completed successfully.");

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   return (0);
}
```

xfnts_validate 方法代码列表

此方法将验证 Confdir_list 属性所指向的目录是否存在。当创建数据服务以及群集管理员更新数据服务属性时，RGM 都将调用此方法。只要故障监视器将数据服务故障转移到新节点上，Monitor_check 方法就将调用此方法。

示例 C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_validate.c - validate method for HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Check to make sure that the properties have been set properly.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;

   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = svc_validate(scds_handle);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method */
   return (rc);

}
```

附录 D

合法的 RGM 名称和值

本附录列出了 资源组管理器 (RGM) 名称和值的合法字符的要求。

本附录包括以下主题：

RGM 合法名称

RGM 名称分为以下几类：

- 资源组名称
- 资源类型名称
- 资源名称
- 属性名称
- 枚举文字名称

命名规则（不包括资源类型名称的命名规则）

除了资源类型名称以外，所有名称必须遵从以下规则：

- 名称必须为 ASCII 形式。
- 名称必须以字母开头。
- 名称可以包含大写和小写字母、数字、破折号 (-) 和下划线 (_)。
- 一个名称中可以使用的最大字符数为 255。

资源类型名称的格式

完整的资源类型名称的格式取决于资源类型，如下所示：

- 如果资源类型的资源类型注册 (RTR) 文件包含 `#$upgrade` 指令，则其格式如下：

  vendor-id.*base-rt-name*:*rt-version*

- 如果资源类型的 RTR 文件**不**包含 `#$upgrade` 指令，则其格式如下：

  vendor-id.*base-rt-name*

句号将 *vendor-id* 和 *base-rt-name* 隔开。冒号将 *base-rt-name* 和 *rt-version* 隔开。

此格式中的变量元素的作用如下所示：

vendor-id 指定供应商 ID 的前缀，其值为 RTR 文件中 `Vendor_id` 资源类型属性的值。如果您要开发资源类型，请选择可以唯一标识供应商的供应商 ID 前缀，例如公司的股票买卖代号。例如，由 Sun Microsystems, Inc. 开发的资源类型的供应商 ID 前缀为 `SUNW`。

base-rt-name 指定基本资源类型名称，其值为 RTR 文件中 `Resource_type` 资源类型属性的值。

rt-version 指定版本后缀，其值为 RTR 文件中 `RT_version` 资源类型属性的值。如果 RTR 文件包含 `#$upgrade` 指令，则版本后缀**仅**是完整资源类型名称的一部分。Sun Cluster 产品的 3.1 版中引入了 `#$upgrade` 指令。

注 – 如果仅注册了基本资源类型名称的一个版本，则不必使用 `scrgadm` 命令中的完整名称。您可以省略供应商 ID 前缀、版本号后缀或两者均省略。

有关更多信息，请参见第 213 页中的 “资源类型属性” 。

示例 D–1 包含 `#$upgrade` 指令的资源类型的完整名称

此示例显示了在 RTR 文件中设置了其属性的资源类型的完整名称。

- `Vendor_id=SUNW`
- `Resource_type=sample`
- `RT_version=2.0`

由此 RTR 文件所定义的资源类型的完整名称如下：

```
SUNW.sample:2.0
```

示例 D–2 不包含 `#$upgrade` 指令的资源类型的完整名称

此示例显示了在 RTR 文件中设置了其属性的资源类型的完整名称。

- `Vendor_id=SUNW`
- `Resource_type=nfs`

示例 D–2 不包含 `#$upgrade` 指令的资源类型的完整名称 （续）

由此 RTR 文件所定义的资源类型的完整名称如下：

```
SUNW.nfs
```

RGM 值

RGM 值分为以下两类：属性值和描述值。两个类别均具有相同的规则：

- 值必须为 ASCII 码。
- 值的最大长度为 4 MB 减 1，即 4,194,303 字节。
- 值不能包含以下字符：
  - 空
  - 换行符
  - 逗号 (,)
  - 分号 (;)

附录 E

对不支持群集的应用程序的要求

不支持群集的普通应用程序必须满足一些要求才能具有高可用性 (HA)。本节第 27 页中的 “分析应用程序的适用性”列出了这些要求。本附录提供了有关该列表中各项要求的详细信息。

将应用程序的资源配置到资源组中可以使该应用程序具有高可用性。将应用程序的数据置于高可用性群集文件系统上，可以在一个服务器出现故障时，通过其他未发生故障的服务器继续访问这些数据。有关群集文件系统的信息，请参见《Sun Cluster 概念指南（适用于 Solaris OS）》。

为了网络上的客户机可进行网络访问，在逻辑主机名资源（与数据服务资源包含在同一个资源组内）中配置了逻辑网络 IP 地址。数据服务资源和网络地址资源共同进行故障转移，这样该数据服务的网络客户机可以访问其新主机上的数据服务资源。

本附录包括以下主题：

多主机数据

具有高可用性的群集文件系统的设备被托管在多个主机上，这样即使某个物理主机崩溃，其它某个未崩溃的主机也可以继续访问该设备。要使应用程序具有高可用性，其数据必须具有高可用性。因此，应用程序的数据必须位于可以从多个群集节点访问的文件系统中。Sun Cluster 支持的高可用性文件系统的示例包括 HA 文件系统、故障转移文件系统 (Failover File System, FFS) 和使用 Oracle Real Application Clusters 的环境中的 QFS 共享文件系统。

群集文件系统安装在作为独立实体创建的设备组上。您可以选择将一些设备组用于安装群集文件系统，而将其他设备组用作提供数据服务（例如 HA Oracle 软件）的原始设备。

应用程序可能包含指向数据文件位置的命令行切换或配置文件。如果应用程序使用硬链接路径名，则您可以更改指向群集文件系统中某个文件的符号链接的路径名，而无需更改应用程序代码。有关使用符号链接的详细讨论，请参见第 298 页中的 “将符号链接用于多主机数据放置”。

在最坏的情况下，必须修改应用程序的源代码以提供指向实际数据位置的机制。您可以通过创建其他命令行参数来实现此机制。

Sun Cluster 软件支持 UNIX UFS 文件系统的使用和卷管理器中配置的 HA 原始设备。安装和配置 Sun Cluster 软件时，群集管理员必须指定要用于 UFS 文件系统的磁盘资源和要用于原始设备的磁盘资源。通常，原始设备只供数据库服务器和多媒体服务器使用。

将符号链接用于多主机数据放置

有时候，应用程序数据文件的路径名为硬链接，且不具有覆盖硬链接路径名的机制。为避免修改应用程序代码，您可以适时地使用符号链接。

例如，假设该应用程序使用硬链接的路径名 `/etc/mydatafile` 命名其数据文件，那么您可以将文件的路径更改为符号链接（其值指向逻辑主机的某一个文件系统中的文件）。例如，您可以将路径更改为指向 `/global/phys-schost-2/mydatafile` 的符号链接。

该应用程序或其管理过程之一修改了该数据文件的名称及目录后，再这样使用该符号链接将会发生问题。例如，假定应用程序通过先创建一个新临时文件 `/etc/mydatafile.new` 来执行更新。然后，该应用程序通过使用 `rename()` 系统调用（或 `mv` 命令）来重命名该临时文件以使其具有真实文件名。通过创建临时文件并将其重命名为真实文件名，数据服务尝试确保其数据文件的内容始终完好。

不幸的是，`rename()` 操作将损坏符号链接。现在名为 `/etc/mydatafile` 的文件是常规文件，并与 `/etc` 目录位于相同的文件系统中，而不是位于群集的群集文件系统中。因为 `/etc` 文件系统是每个主机专用的，所以进行故障切换或转移之后，该数据不再可用。

潜在的问题是现有应用程序不支持符号链接，并且无法写入以处理符号链接。要使用符号链接将数据访问重定向到逻辑主机的文件系统，该应用程序实现必须以不会删除符号链接的方式运行。因此，对于在群集的文件系统中放置数据的问题，符号链接不是彻底的解决方法。

主机名

您必须确定数据服务是否始终需要知道运行该服务的服务器的主机名。如果是，则可能需要修改数据服务以使用逻辑主机名，而不使用物理主机名。在这种情况下，逻辑主机名是配置到与应用程序资源位于同一资源组的逻辑主机名资源中的主机名。

在数据服务的某些客户机服务器协议中，服务器将在发送给客户机的消息中把自己的主机名返回给客户机。对于这样的协议，客户机依赖于所返回的这个主机名，因为联系服务器时要使用该主机名。为了使所返回的主机名可以在故障转移或转移后可用，该主机名应该是该资源组的逻辑主机名，而不是物理主机的名称。在这种情况下，您必须修改该数据服务的代码，以将逻辑主机名返回到客户机。

多地址主机

术语**多地址主机**的意思是一个主机位于多个公共网络上。这样的主机具有多个主机名和 IP 地址。它针对每个网络具有一个主机名（IP 地址对）。Sun Cluster 允许某一个主机位于任意数量的网络上，包括仅位于一个网络上（非多地址情况）。就像物理主机名具有多个主机名（IP 地址对）一样，每个资源组也可以具有多个主机名（IP 地址对），一个公共网络一个。当 Sun Cluster 将资源组从一个物理主机移至另一物理主机时，该资源组的完整主机名（IP 地址对）组也将一同移动。

该资源组的主机名（IP 地址对）组将被配置为包含在资源组中的逻辑主机名资源。创建和配置资源组时，群集管理员将指定这些网络地址资源。Sun Cluster Data Service API 包含查询这些主机名（IP 地址对）的功能。

大多数针对 Solaris 操作系统编写的现有数据服务守护进程都已可以正确地处理多地址主机。许多数据服务通过绑定到 Solaris 通配符地址 `INADDR_ANY` 的方式来进行其所有网络通信。绑定操作自动引发数据服务处理所有网络接口的全部 IP 地址。`INADDR_ANY` 可以有效地绑定到当前在计算机上配置的所有 IP 地址。使用 `INADDR_ANY` 的数据服务守护进程通常不需要更改就可以处理 Sun Cluster 逻辑网络地址。

绑定到 `INADDR_ANY` 地址而非特定的 IP 地址

即使使用的是非多地址主机，Sun Cluster 逻辑网络地址概念也使计算机可以具有多个 IP 地址。计算机具有自身作为物理主机的一个 IP 地址，并针对其当前主控的每个网络地址（逻辑主机名）资源具有其他 IP 地址。当计算机控制某个网络地址资源时，它将动态获取其它 IP 地址。当它不再控制某个网络地址资源时，也将动态释放 IP 地址。

如果绑定到 INADDR_ANY，某些数据服务将不能在 Sun Cluster 环境中正常工作。随着资源组处于受控制或不受控制状态，这些数据服务必须动态更改资源组所绑定的这组 IP 地址。完成重新绑定操作的策略之一是使这些数据服务的启动和停止方法终止并重新启动该数据服务的守护进程。

Network_resources_used 资源属性允许最终用户配置特定的一组网络地址资源（应用程序将要绑定到该资源上）。对于需要此功能的资源类型，必须在 RTR 文件中为该资源类型声明 Network_resources_used 属性。

当 RGM 使资源组联机或脱机时，RGM 将遵循与 RGM 调用函数调用数据服务资源方法的时间相关的特定顺序探测网络地址、取消对网络地址的探测或将网络地址配置为打开或关闭。请参见第 41 页中的 “确定使用哪种 Start 或 Stop 方法”。

在数据服务的 Stop 方法返回之前，必须已通过使用资源组的网络地址停止该数据服务。同样，Start 方法返回时，该数据服务必定已开始使用该网络地址。

如果数据服务绑定到 INADDR_ANY 上而未绑定到单个 IP 地址上，则调用数据服务资源方法的顺序与调用网络地址方法的顺序不存在对应关系。

如果数据服务的停止和启动方法通过中止和重新启动数据服务的守护进程实现了它们的作用，则可以在恰当的时刻使用网络地址停止或启动数据服务。

客户机重试

对于网络客户机，故障切换或转移操作类似于逻辑主机崩溃，随后进行快速重新引导的过程。理想情况是将客户机服务器协议设计为执行若干次重试操作。如果应用程序和协议已可以处理单个服务器崩溃和重新引导的情况，则它们也可以处理接管和切换转移资源组的情况。一些应用程序可能选择无休止的重试。较复杂的应用程序将通知用户正在进行长时间重试，用户可以选择是否继续进行。

附录 F

CRNP 的文档类型定义

本附录包括群集重新配置通知协议 (CRNP) 的以下文档类型定义 (DTD)：

SC_CALLBACK_REG XML DTD

注 – 由 SC_CALLBACK_REG 和 SC_EVENT 使用的 NVPAIR 数据结构仅被定义一次。

```
<!— SC_CALLBACK_REG XML format specification
        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:

A client of the Cluster Reconfiguration Notification Protocol should use this xml format
to register initially with the service, to subsequently register for more events, to
subsequently remove registration of some events, or to remove itself from the service
entirely.

A client is uniquely identified by its callback IP and port.  The port is defined in the
SC_CALLBACK_REG element, and the IP is taken as the source IP of the registration
connection.  The final attribute of the root SC_CALLBACK_REG element is either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS, depending on which form of the
message the client is using.

The SC_CALLBACK_REG contains 0 or more SC_EVENT_REG sub-elements.
```

```
One SC_EVENT_REG is the specification for one event type.  A client may specify only the
CLASS (an attribute of the SC_EVENT_REG element), or may specify a SUBCLASS (an optional
attribute) for further granularity.  Also, the SC_EVENT_REG has as subelements 0 or more
NVPAIRs, which can be used to further specify the event.

Thus, the client can specify events to whatever granularity it wants.  Note that a client
cannot both register for and unregister for events in the same message.  However a client
can subscribe to the service and sign up for events in the same message.

Note on versioning: the VERSION attribute of each root element is marked "fixed", which
means that all message adhering to these DTDs must have the version value specified.  If a
new version of the protocol is created, the revised DTDs will have a new value for this
fixed" VERSION attribute, such that all message adhering to the new version must have the
new version number.
—>
<!— SC_CALLBACK_REG definition

The root element of the XML document is a registration message.  A registration message
consists of the callback port and the protocol version as attributes, and either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS attribute, specifying the
registration type.  The ADD_CLIENT, ADD_EVENTS, and REMOVE_EVENTS types should have one or
more SC_EVENT_REG subelements.  The REMOVE_CLIENT should not specify an SC_EVENT_REG
subelement.
        ATTRIBUTES:
               VERSION        The CRNP protocol version of the message.
               PORT           The callback port.
               REG_TYPE       The type of registration.  One of:
                       ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENTS:
SUBELEMENTS: SC_EVENT_REG (0 or more)
—>
<!ELEMENT SC_CALLBACK_REG (SC_EVENT_REG*)>
<!ATTLIST SC_CALLBACK_REG
       VERSION        NMTOKEN                                                  #FIXED
       PORT           NMTOKEN                                                  #REQUIRED
       REG_TYPE       (ADD_CLIENT|ADD_EVENTS|REMOVE_CLIENT|REMOVE_EVENTS)    #REQUIRED

>
<!— SC_EVENT_REG definition

The SC_EVENT_REG defines an event for which the client is either registering or
unregistering interest in receiving event notifications.  The registration can be for any
level of granularity, from only event class down to specific name/value pairs that must be
present.  Thus, the only required attribute is the CLASS.  The SUBCLASS attribute, and the
NVPAIRS sub-elements are optional, for higher granularity.

Registrations that specify name/value pairs are registering interest in notification of
messages from the class/subclass specified with ALL name/value pairs present.
Unregistrations that specify name/value pairs are unregistering interest in notifications
that have EXACTLY those name/value pairs in granularity previously specified.
Unregistrations that do not specify name/value pairs unregister interest in ALL event
notifications of the specified class/subclass.

        ATTRIBUTES:
```

```
                CLASS:        The event class for which this element is registering
                              or unregistering interest.
                SUBCLASS:     The subclass of the event (optional).

        CONTENTS:
                SUBELEMENTS: 0 or more NVPAIRs.
—>
<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA               #REQUIRED
       SUBCLASS       CDATA               #IMPLIED
>
```

NVPAIR XML DTD

```
<!— NVPAIR XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:
              An nvpair element is meant to be used in an SC_EVENT or SC_CALLBACK_REG
              element.
—>
<!— NVPAIR definition

       The NVPAIR is a name/value pair to represent arbitrary name/value combinations.
       It is intended to be a direct, generic, translation of the Solaris nvpair_t
       structure used by the sysevent framework.  However, there is no type information
       associated with the name or the value (they are both arbitrary text) in this xml
       element.

       The NVPAIR consists simply of one NAME element and one or more VALUE elements.
       One VALUE element represents a scalar value, while multiple represent an array
       VALUE.

       ATTRIBUTES:

       CONTENTS:
              SUBELEMENTS: NAME(1), VALUE(1 or more)
—>
<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— NAME definition

        The NAME is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
```

```
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT NAME (#PCDATA)>

<!— VALUE definition
        The VALUE is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT VALUE (#PCDATA)>
```

SC_REPLY XML DTD

```
<!— SC_REPLY XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.
—>
<!— SC_REPLY definition

        The root element of the XML document represents a reply to a message.  The reply
        contains a status code and a status message.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                STATUS_CODE:    The return code for the message.  One of the
                                following: OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
                                MALFORMED, INVALID_XML, VERSION_TOO_HIGH, or
                                VERSION_TOO_LOW.

                CONTENTS:
                        SUBELEMENTS: SC_STATUS_MSG(1)
—>
<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
        VERSION         NMTOKEN                                         #FIXED   "1.0"
        STATUS_CODE     OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
                        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>
<!— SC_STATUS_MSG definition
        The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
        code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.

        ATTRIBUTES:
```

```
        CONTENTS:
                Arbitrary string.
-->
<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

SC_EVENT XML DTD

注 – 由 SC_CALLBACK_REG 和 SC_EVENT 使用的 NVPAIR 数据结构仅被定义一次。

```
<!— SC_EVENT XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        The root element of the XML document is intended to be a direct, generic,
        translation of the Solaris sysseventd message format.  It has attributes to
        represent the class, subclass, vendor, and publisher, and contains any number of
        NVPAIR elements.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                CLASS:          The sysevent class of the event
                SUBCLASS:       The subclass of the event
                VENDOR:         The vendor associated with the event
                PUBLISHER:      The publisher of the event
        CONTENTS:
                SUBELEMENTS: NVPAIR (0 or more)
-->
<!ELEMENT SC_EVENT (NVPAIR*)>
<!ATTLIST SC_EVENT
        VERSION         NMTOKEN         #FIXED "1.0"
        CLASS           CDATA           #REQUIRED
        SUBCLASS        CDATA           #REQUIRED
        VENDOR          CDATA           #REQUIRED
        PUBLISHER       CDATA           #REQUIRED
>
```

附录 G

CrnpClient.java 应用程序

本附录介绍了完整的 CrnpClient.java 应用程序，第 12 章中对该应用程序进行了详细讨论。

CrnpClient.java 的内容

```
/*
 * CrnpClient.java
 * ================
 *
 * Note regarding XML parsing:
 *
 * This program uses the Sun Java Architecture for XML Processing (JAXP) API.
 * See http://java.sun.com/xml/jaxp/index.html for API documentation and
 * availability information.
 *
 * This program was written for Java 1.3.1 or higher.
 *
 * Program overview:
 *
 * The main thread of the program creates a CrnpClient object, waits for the
 * user to terminate the demo, then calls shutdown on the CrnpClient object
 * and exits the program.
 *
 * The CrnpClient constructor creates an EventReceptionThread object,
 * opens a connection to the CRNP server (using the host and port specified
 * on the command line), constructs a registration message (based on the
 * command-line specifications), sends the registartion message, and reads
 * and parses the reply.
 *
 * The EventReceptionThread creates a listening socket bound to
 * the hostname of the machine on which this program runs, and the port
 * specified on the command line. It waits for an incoming event callback,
```

```
 * at which point it constructs an XML Document from the incoming socket
 * stream, which is then passed back to the CrnpClient object to process.
 *
 * The shutdown method in the CrnpClient just sends an unregistration
 * (REMOVE_CLIENT) SC_CALLBACK_REG message to the crnp server.
 *
 * Note regarding error handling: for the sake of brevity, this program just
 * exits on most errors.  Obviously, a real application would attempt to handle
 * some errors in various ways, such as retrying when appropriate.
 */

// JAXP packages
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

// standard packages
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * class CrnpClient
 * ----------------
 * See file header comments above.
 */
class CrnpClient
{
    /*
     * main
     * ----
     * The entry point of the execution, main simply verifies the
     * number of command-line arguments, and constructs an instance
     * of a CrnpClient to do all the work.
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

        /* Verify the number of command-line arguments */
        if (args.length < 4) {
            System.out.println(
                "Usage: java CrnpClient crnpHost crnpPort "
                + "localPort (-ac | -ae | -re) "
                + "[(M | A | RG=name | R=name) [...]]");
            System.exit(1);
        }


        /*
```

```
     * We expect the command line to contain the ip/port of the
     * crnp server, the local port on which we should listen, and
     * arguments specifying the type of registration.
     */
    try {
        regIp = InetAddress.getByName(args[0]);
        regPort = (new Integer(args[1])).intValue();
        localPort = (new Integer(args[2])).intValue();
    } catch (UnknownHostException e) {
        System.out.println(e);
        System.exit(1);
    }

    // Create the CrnpClient
    CrnpClient client = new CrnpClient(regIp, regPort, localPort,
        args);

    // Now wait until the user wants to end the program
    System.out.println("Hit return to terminate demo...");

    // read will block until the user enters something
    try {
        System.in.read();
    } catch (IOException e) {
        System.out.println(e.toString());
    }

    // shutdown the client
    client.shutdown();
    System.exit(0);
}

/*
 * ======================
 * public methods
 * ======================
 */

/*
 * CrnpClient constructor
 * ----------------------
 * Parses the command line arguments so we know how to contact
 * the crnp server, creates the event reception thread, and starts it
 * running, creates the XML DocumentBuilderFactory obect, and, finally,
 * registers for callbacks with the crnp server.
 */
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
        localPort = localPortIn;
        regs = clArgs;
```

```
        /*
         * Setup the document builder factory for
         * xml processing.
         */
        setupXmlProcessing();

        /*
         * Create the EventReceptionThread, which creates a
         * ServerSocket and binds it to a local ip and port.
         */
        createEvtRecepThr();

        /*
         * Register with the crnp server.
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * ---------------
 * Callback into the CrnpClient, used by the EventReceptionThread
 * when it receives event callbacks.
 */
public void processEvent(Event event)
{
    /*
     * For demonstration purposes, simply print the event
     * to System.out.  A real application would obviously make
     * use of the event in some way.
     */
    event.print(System.out);
}

/*
 * shutdown
 * -------------
 * Unregister from the CRNP server.
 */
public void shutdown()
{
    try {
        /* send an unregistration message to the server */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}
```

```
/*
 * ======================
 * private helper methods
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Create the document builder factory for
 * parsing the xml replies and events.
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // We don't need to bother validating
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // We want to ignore comments and whitespace
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // Coalesce CDATA sections into TEXT nodes.
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -------------------
 * Creates a new EventReceptionThread object, saves the ip
 * and port to which its listening socket is bound, and
 * starts the thread running.
 */
private void createEvtRecepThr() throws Exception
{
    /* create the thread object */
    evtThr = new EventReceptionThread(this);

    /*
     * Now start the thread running to begin listening
     * for event delivery callbacks.
     */
    evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
 * Creates a socket connection to the crnp server and sends
 * an event registration message.
```

```
 */
private void registerCallbacks() throws Exception
{
    System.out.println("About to register");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the registration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}

/*
 * unregister
 * ----------
 * As in registerCallbacks, we create a socket connection to
 * the crnp server, send the unregistration message, wait for
 * the reply from the server, then close the socket.
 */
private void unregister() throws Exception
{
    System.out.println("About to unregister");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the unregistration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}
```

```
/*
 * createRegistrationString
 * ------------------
 * Constructs a CallbackReg object based on the command line arguments
 * to this program, then retrieves the XML string from the CallbackReg
 * object.
 */
private String createRegistrationString() throws Exception
{
    /*
     * create the actual CallbackReg class and set the port.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    // set the registration type
    if (regs[3].equals("-ac")) {
        cbReg.setRegType(CallbackReg.ADD_CLIENT);
    } else if (regs[3].equals("-ae")) {
        cbReg.setRegType(CallbackReg.ADD_EVENTS);
    } else if (regs[3].equals("-re")) {
        cbReg.setRegType(CallbackReg.REMOVE_EVENTS);
    } else {
        System.out.println("Invalid reg type: " + regs[3]);
        System.exit(1);
    }

    // add the events
    for (int i = 4; i < regs.length; i++) {
        if (regs[i].equals("M")) {
            cbReg.addRegEvent(createMembershipEvent());
        } else if (regs[i].equals("A")) {
            cbReg.addRegEvent(createAllEvent());
        } else if (regs[i].substring(0,2).equals("RG")) {
            cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
        } else if (regs[i].substring(0,1).equals("R")) {
            cbReg.addRegEvent(createREvent(regs[i].substring(2)));
        }
    }

    String xmlStr = cbReg.convertToXml();
    System.out.println(xmlStr);
    return (xmlStr);
}

/*
 * createAllEvent
 * ----------------
 * Creates an XML registartion event with class EC_Cluster, and no
 * subclass.
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
```

```
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
 * ---------------------
 * Creates an XML registration event with class EC_Cluster, subclass
 * ESC_cluster_memberhip.
 */
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
 * subclass ESC_cluster_rg_state, and one "rg_name" nvpair (based
 * on input parameter).
 */
private Event createRgEvent(String rgname)
{
    /*
     * Create a Resource Group state change event for the
     * rgname Resource Group.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    /*
     * Construct the event object and set the class and subclass.
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * Create the nvpair object and add it to the Event.
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
}

/*
 * createREvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
```

```
 * subclass ESC_cluster_r_state, and one "r_name" nvpair (based
 * on input parameter).
 */
private Event createREvent(String rname)
{
    /*
     * Create a Resource state change event for the
     * rgname Resource.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    Event rStateEvent = new Event();
    rStateEvent.setClass("EC_Cluster");
    rStateEvent.setSubclass("ESC_cluster_r_state");

    NVPair rNvpair = new NVPair();
    rNvpair.setName("r_name");
    rNvpair.setValue(rname);
    rStateEvent.addNvpair(rNvpair);

    return (rStateEvent);
}

/*
 * createUnregistrationString
 * ------------------
 * Constructs a REMOVE_CLIENT CallbackReg object, then retrieves
 * the XML string from the CallbackReg object.
 */
private String createUnregistrationString() throws Exception
{
    /*
     * Crate the CallbackReg object.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
     * we marshall the registration to the OutputStream
     */
    String xmlStr = cbReg.convertToXml();

    // Print the string for debugging purposes
    System.out.println(xmlStr);
    return (xmlStr);
}


/*
 * readRegistrationReply
 * ------------------------
 * Parse the xml into a Document, construct a RegReply object
 * from the document, and print the RegReply object.  Note that
 * a real application would take action based on the status_code
```

```
     * of the RegReply object.
     */
    private void readRegistrationReply(InputStream stream)
        throws Exception
    {
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
    }

    /* private member variables */
    private InetAddress regIp;
    private int regPort;
    private EventReceptionThread evtThr;
    private String regs[];

    /* public member variables */
    public int localPort;
    public DocumentBuilderFactory dbf;
}

/*
 * class EventReceptionThread
 * --------------------------
 * See file header comments above.
 */
class EventReceptionThread extends Thread
{
    /*
     * EventReceptionThread constructor
     * --------------------------------
     * Creates a new ServerSocket, bound to the local hostname and
     * a wildcard port.
     */
    public EventReceptionThread(CrnpClient clientIn) throws IOException
    {
        /*
         * keep a reference to the client so we can call it back
         * when we get an event.
         */
        client = clientIn;

        /*
         * Specify the IP to which we should bind.  It's
```

```
     * simply the local host ip.  If there is more
     * than one public interface configured on this
     * machine, we'll go with whichever one
     * InetAddress.getLocalHost comes up with.
     *
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
        System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Called by the Thread.Start method.
 *
 * Loops forever, waiting for incoming connections on the ServerSocket.
 *
 * As each incoming connection is accepted, an Event object
 * is created from the xml stream, which is then passed back to
 * the CrnpClient object for processing.
 */
public void run()
{
    /*
     * Loop forever.
     */
    try {
        //
        // Create the document builder using the document
        // builder factory in the CrnpClient.
        //
        DocumentBuilder db = client.dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        while(true) {
            /* wait for a callback from the server */
            Socket sock = listeningSock.accept();

            // parse the input file
            Document doc = db.parse(sock.getInputStream());

            Event event = new Event(doc);
            client.processEvent(event);

            /* close the socket */
            sock.close();
        }
        // UNREACHABLE
```

```
        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }

    /* private member variables */
    private ServerSocket listeningSock;
    private CrnpClient client;
}

/*
 * class NVPair
 * ------------
 * This class stores a name/value pair (both Strings).  It knows how to
 * construct an NVPAIR XML message from its members, and how to parse
 * an NVPAIR XML Element into its members.
 *
 * Note that the formal specification of an NVPAIR allows for multiple values.
 * We make the simplifying assumption of only one value.
 */
class NVPair
{
    /*
     * Two constructors: the first creates an empty NVPair, the second
     * creates an NVPair from an NVPAIR XML Element.
     */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
     * Public setters.
     */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }

    /*
     * Prints the name and value on a single line.
     */
    public void print(PrintStream out)
    {
```

```
    out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an NVPAIR XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.
 */
public Element createXmlElement(Document doc)
{
    // Create the element.
    Element nvpair = (Element)
        doc.createElement("NVPAIR");
    //
    // Add the name.  Note that the actual name is
    // a separate CDATA section.
    //
    Element eName = doc.createElement("NAME");
    Node nameData = doc.createCDATASection(name);
    eName.appendChild(nameData);
    nvpair.appendChild(eName);
    //
    // Add the value.  Note that the actual value is
    // a separate CDATA section.
    //
    Element eValue = doc.createElement("VALUE");
    Node valueData = doc.createCDATASection(value);
    eValue.appendChild(valueData);
    nvpair.appendChild(eValue);

    return (nvpair);
}

/*
 * retrieveValues
 * ----------------
 * Parse the XML Element to retrieve the name and value.
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // Find the NAME element
    //
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }
```

```
        //
        // Get the TEXT section
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            System.out.println("Error in parsing: can't find "
                + "TEXT section.");
            return;
        }

        // Retrieve the value
        name = n.getNodeValue();

        //
        // Now get the value element
        //
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "VALUE node.");
            return;
        }

        //
        // Get the TEXT section
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            System.out.println("Error in parsing: can't find "
                + "TEXT section.");
            return;
        }

        // Retrieve the value
        value = n.getNodeValue();
    }


    /*
     * Public accessors
     */
    public String getName()
    {
        return (name);
    }

    public String getValue()
    {
        return (value);
    }

    // Private member vars
    private String name, value;
}
```

```
/*
 * class Event
 * -----------
 * This class stores an event, which consists of a class, subclass, vendor,
 * publisher, and list of name/value pairs.  It knows how to
 * construct an SC_EVENT_REG XML Element from its members, and how to parse
 * an SC_EVENT XML Element into its members.  Note that there is an assymetry
 * here: we parse SC_EVENT elements, but construct SC_EVENT_REG elements.
 * That is because SC_EVENT_REG elements are used in registration messages
 * (which we must construct), while SC_EVENT elements are used in event
 * deliveries (which we must parse).  The only difference is that SC_EVENT_REG
 * elements don't have a vendor or publisher.
 */
class Event
{

    /*
     * Two constructors: the first creates an empty Event; the second
     * creates an Event from an SC_EVENT XML Document.
     */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // Convert the document to a string to print for debugging
        // purposes.
        //
        DOMSource domSource = new DOMSource(doc);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return;
        }
        System.out.println(strWrite.toString());

        // Do the actual parsing.
        retrieveValues(doc);
    }

    /*
     * Public setters.
```

```
 */
public void setClass(String classIn)
{
    regClass = classIn;
}

public void setSubclass(String subclassIn)
{
    regSubclass = subclassIn;
}

public void addNvpair(NVPair nvpair)
{
    nvpairs.add(nvpair);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an SC_EVENT_REG XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.  Relies on the NVPair createXmlElement ability.
 */
public Element createXmlElement(Document doc)
{
    Element event = (Element)
        doc.createElement("SC_EVENT_REG");
    event.setAttribute("CLASS", regClass);
    if (regSubclass != null) {
        event.setAttribute("SUBCLASS", regSubclass);
    }
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        event.appendChild(tempNv.createXmlElement(doc));
    }
    return (event);
}

/*
 * Prints the member vars on multiple lines.
 */
public void print(PrintStream out)
{
    out.println("\tCLASS=" + regClass);
    out.println("\tSUBCLASS=" + regSubclass);
    out.println("\tVENDOR=" + vendor);
    out.println("\tPUBLISHER=" + publisher);
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        out.print("\t\t");
        tempNv.print(out);
    }
}
```

```
 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the class, subclass, vendor,
 * publisher, and nvpairs.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_EVENT element.
    //
    nl = doc.getElementsByTagName("SC_EVENT");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "SC_EVENT node.");
        return;
    }

    n = nl.item(0);

    //
    // Retrieve the values of the CLASS, SUBCLASS,
    // VENDOR and PUBLISHER attributes.
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // Retrieve all the nv pairs
    //
    for (Node child = n.getFirstChild(); child != null;
         child = child.getNextSibling())
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 * Public accessor methods.
 */
public String getRegClass()
{
    return (regClass);
}

public String getSubclass()
{
    return (regSubclass);
}
```

```
    public String getVendor()
    {
        return (vendor);
    }

    public String getPublisher()
    {
        return (publisher);
    }

    public Vector getNvpairs()
    {
        return (nvpairs);
    }

    // Private member vars.
    private String regClass, regSubclass;
    private Vector nvpairs;
    private String vendor, publisher;
}


/*
 * class CallbackReg
 * -----------
 * This class stores a port and regType (both Strings), and a list of Events.
 * It knows how to construct an SC_CALLBACK_REG XML message from its members.
 *
 * Note that this class does not need to be able to parse SC_CALLBACK_REG
 * messages, because only the CRNP server must parse SC_CALLBACK_REG
 * messages.
 */
class CallbackReg
{
    // Useful defines for the setRegType method
    public static final int ADD_CLIENT = 0;
    public static final int ADD_EVENTS = 1;
    public static final int REMOVE_EVENTS = 2;
    public static final int REMOVE_CLIENT = 3;

    public CallbackReg()
    {
        port = null;
        regType = null;
        regEvents = new Vector();
    }

    /*
     * Public setters.
     */
    public void setPort(String portIn)
    {
        port = portIn;
    }
```

```
public void setRegType(int regTypeIn)
{
    switch (regTypeIn) {
    case ADD_CLIENT:
        regType = "ADD_CLIENT";
        break;
    case ADD_EVENTS:
        regType = "ADD_EVENTS";
        break;
    case REMOVE_CLIENT:
        regType = "REMOVE_CLIENT";
        break;
    case REMOVE_EVENTS:
        regType = "REMOVE_EVENTS";
        break;
    default:
        System.out.println("Error, invalid regType " +
            regTypeIn);
        regType = "ADD_CLIENT";
        break;
    }
}

public void addRegEvent(Event regEvent)
{
    regEvents.add(regEvent);
}


/*
 * convertToXml
 * ------------------
 * Constructs an SC_CALLBACK_REG XML Document from the member
 * variables.  Relies on the Event createXmlElement ability.
 */
public String convertToXml()
{
    Document document = null;
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.newDocument();
    } catch (ParserConfigurationException pce) {
        // Parser with specified options can't be built
        pce.printStackTrace();
        System.exit(1);
    }
    Element root = (Element) document.createElement("SC_CALLBACK_REG");
    root.setAttribute("VERSION", "1.0");
    root.setAttribute("PORT", port);
    root.setAttribute("REG_TYPE", regType);
    for (int i = 0; i < regEvents.size(); i++) {
        Event tempEvent = (Event)
```

```
                (regEvents.elementAt(i));
            root.appendChild(tempEvent.createXmlElement(document));
        }
        document.appendChild(root);

        //
        // Now convert the document to a string.
        //
        DOMSource domSource = new DOMSource(document);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return ("");
        }
        return (strWrite.toString());
    }

    // private member vars
    private String port;
    private String regType;
    private Vector regEvents;
}

/*
 * class RegReply
 * ------------
 * This class stores a status_code and status_msg (both Strings).
 * It knows how to parse an SC_REPLY XML Element into its members.
 */
class RegReply
{

    /*
     * The only constructor takes an XML Document and parses it.
     */
    public RegReply(Document doc)
    {
        //
        // Now convert the document to a string.
        //
        DOMSource domSource = new DOMSource(doc);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return;
```

```
    }
    System.out.println(strWrite.toString());

    retrieveValues(doc);
}

/*
 * Public accessors
 */
public String getStatusCode()
{
    return (statusCode);
}

public String getStatusMsg()
{
    return (statusMsg);
}

/*
 * Prints the info on a single line.
 */
public void print(PrintStream out)
{
    out.println(statusCode + ": " +
        (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the statusCode and statusMsg.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_REPLY element.
    //
    nl = doc.getElementsByTagName("SC_REPLY");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "SC_REPLY node.");
        return;
    }

    n = nl.item(0);

    // Retrieve the value of the STATUS_CODE attribute
    statusCode = ((Element)n).getAttribute("STATUS_CODE");

    //
    // Find the SC_STATUS_MSG element
```

```
        //
        nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "SC_STATUS_MSG node.");
            return;
        }

        //
        // Get the TEXT section, if there is one.
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Not an error if there isn't one, so we
            // just silently return.
            return;
        }

        // Retrieve the value
        statusMsg = n.getNodeValue();
    }

    // private member vars
    private String statusCode;
    private String statusMsg;
}
```

索引

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

安

编

变

菜

参

测

持

错

代

多

二

组