



# 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: 817-6391-10  
2004 年 4 月, 修订版 A

版权所有 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. 保留所有权利。

本产品或文档受版权保护，并按照限制其使用、复制、发行和反汇编的许可证进行分发。未经 Sun 及其许可方（如果有）的事先书面许可，不得以任何形式、任何手段复制本产品或文档的任何部分。第三方软件（包括字体技术）受 Sun 提供商的版权保护及许可协议限制。

本产品的某些部分来源于 Berkeley BSD 系统，并获得了加利福尼亚大学的许可。UNIX 是由 X/Open Company, Ltd. 在美国和其它国家/地区独家许可的注册商标。

Sun、Sun Microsystems、Sun 徽标和 Java 是 Sun Microsystems, Inc. 在美国和其它国家/地区的商标或注册商标。

Sun、Sun Microsystems、Sun 徽标、Java、docs.sun.com、AnswerBook、AnswerBook2、NetBeans、Sun StorEdge、Sun Cluster、SunPlex 和 Solaris 是 Sun Microsystems, Inc. 在美国和其它国家/地区的商标、注册商标或服务标记。所有 SPARC 商标的使用均已获得许可，它们是 SPARC International Inc. 在美国和其它国家/地区的商标或注册商标。标有 SPARC 商标的产品均基于由 Sun Microsystems, Inc. 开发的体系结构。Adobe 是 Adobe Systems, Incorporated 的注册商标。PostScript 徽标是 Adobe Systems, Incorporated 的商标或注册商标（在某些地区可能已注册）。ORACLE 是 Oracle Corporation 的注册商标。

OPEN LOOK 和 Sun™ 图形用户界面是 Sun Microsystems, Inc. 为其用户和被许可方开发的。Sun 感谢 Xerox 在研究和开发可视或图形用户界面的概念方面为计算机行业所做的超前贡献。Sun 已从 Xerox 获得了对 Xerox 图形用户界面的非独占性许可证，该许可证还适用于实现 OPEN LOOK GUI 和在其它方面遵守 Sun 书面许可协议的 Sun 许可证持有者。

本档按“原样”提供，对所有明示或默示的条件、陈述和担保，包括适销性、适用于某特定用途和非侵权的默认保证，均不承担任何责任，除非此免责声明的适用范围在法律上无效。



040521 @8606



# 目录

---

序	11
<b>1 资源管理概述</b>	<b>17</b>
Sun Cluster 应用程序环境	17
RGM 模型	18
资源类型	19
资源	19
资源组	19
资源组管理器	20
回叫方法	20
编程接口	21
RMAPI	21
数据服务开发库 (DSDL)	22
SunPlex Agent Builder	22
资源组管理器管理接口	22
SunPlex Manager	22
管理命令	23
<b>2 开发数据服务</b>	<b>25</b>
分析应用程序的适用性	25
确定要使用的接口	27
设置用来编写数据服务的开发环境	28
▼ 设置开发环境	28
将数据服务传送到群集	29
设置资源和资源类型特性	29

声明资源类型特性	30
声明资源特性	32
声明扩展特性	35
实现回叫方法	36
访问资源和资源组特性信息	36
方法的幂等性	37
普通数据服务	37
控制应用程序	38
启动和停止资源	38
Init、Fini 和 Boot 方法	40
监视资源	40
向资源添加消息日志	41
提供进程管理	41
提供资源的管理支持	42
实现故障切换资源	42
实现可伸缩资源	43
可伸缩服务的验证检查	45
编写和测试数据服务	45
使用持续连接机制	45
测试 HA 数据服务	46
协调资源间的依赖性	46
<b>3 升级资源类型</b>	<b>49</b>
概述	49
资源类型登记文件	50
资源类型名称	50
指令	51
更改 RTR 文件中的 RT_Version	51
Sun Cluster 早期版本中的资源类型名称	51
资源的 Type_version 特性	52
将资源迁移到其它版本	53
升级和降级资源类型	53
▼ 如何升级资源类型	53
▼ 如何将资源降级到其资源类型的早期版本	54
缺省特性值	55
资源类型开发者文档	56
资源类型名称和资源类型监视器实现	56

应用程序升级	57
资源类型升级实例	57
资源类型软件包的安装要求	59
您在更改 RTR 文件之前需了解的信息	60
更改监视器代码	60
更改方法代码	60
<b>4 资源管理 API 参考</b>	<b>63</b>
RMAPI 存取方法	64
RMAPI Shell 命令	64
C 函数	65
RMAPI 回叫方法	68
方法变量	69
出口代码	69
控制和初始化回叫方法	69
管理支持方法	70
与网络相关的回叫方法	71
监视器控制回叫方法	71
<b>5 数据服务样例</b>	<b>73</b>
数据服务样例概述	73
定义资源类型登记文件	74
RTR 文件概述	74
RTR 文件样例中的资源类型特性	74
RTR 文件样例中的资源特性	76
为所有方法提供通用功能	79
标识命令解释程序并输出路径	79
声明 PMF_TAG 和 SYSLOG_TAG 变量	79
分析函数变量	80
生成错误消息	82
获取特性信息	82
控制数据服务	83
Start 方法	83
Stop 方法	86
定义故障监视器	88
探测程序	88
Monitor_start 方法	93

Monitor_stop 方法	94
Monitor_check 方法	95
处理特性更新	96
Validate 方法	96
Update 方法	100
<b>6 数据服务开发库 (DSDL)</b>	<b>103</b>
DSDL 概述	103
管理配置特性	103
启动和停止数据服务	104
实现故障监视器	105
存取网络地址信息	105
调试资源类型实现	106
启动具有高可用性的本地文件系统	106
<b>7 设计资源类型</b>	<b>107</b>
RTR 文件	107
Validate 方法	108
Start 方法	109
Stop 方法	110
Monitor_start 方法	111
Monitor_stop 方法	111
Monitor_check 方法	112
Update 方法	112
Init、Fini 和 Boot 方法	113
设计故障监视器守护程序	113
<b>8 DSDL 资源类型实现样例</b>	<b>117</b>
X Font Server	117
X Font Server 配置文件	118
TCP 端口号	118
命名惯例	118
SUNW.xfnts RTR 文件	119
scds_initialize() 函数	119
xfnts_start 方法	120
在启动之前验证服务	120
启动服务	120

从 <code>svc_start()</code> 返回	122
<code>xfnts_stop</code> 方法	124
<code>xfnts_monitor_start</code> 方法	125
<code>xfnts_monitor_stop</code> 方法	126
<code>xfnts_monitor_check</code> 方法	127
SUNW.xfnts 故障监视器	127
<code>xfnts_probe</code> 主循环	128
<code>svc_probe()</code> 函数	129
确定故障监视器操作	132
<code>xfnts_validate</code> 方法	133
<code>xfnts_update</code> 方法	135

<b>9 SunPlex Agent Builder</b>	<b>137</b>
使用 Agent Builder	138
分析应用程序	138
安装和配置 Agent Builder	138
启动 Agent Builder	139
使用“创建”屏幕	140
使用“配置”屏幕	142
重复使用完成的工作	145
克隆现有的资源类型	145
编辑已生成的源代码	146
使用 Agent Builder 的命令行版本	147
目录结构	147
输出	148
源文件和二进制文件	148
公用程序脚本和手册页	149
支持文件	150
软件包目录	150
<code>rtconfig</code> 文件	151
浏览 Agent Builder	151
“浏览”按钮	152
菜单	153
Agent Builder 的 Cluster Agent 模块	154
▼ 安装和设置 Cluster Agent 模块	154
▼ 启动 Cluster Agent 模块	155
使用 Cluster Agent 模块	156

Cluster Agent 模块和 Agent Builder 之间的区别 157

## 10 普通数据服务 159

GDS 概述 159

预编译的资源类型 159

使用 GDS 的原因 160

创建使用 GDS 的服务的方法 160

所需的 GDS 特性 161

可选的 GDS 特性 161

使用 SunPlex Agent Builder 创建使用 GDS 的服务 164

在 SunPlex Agent Builder 中创建使用 GDS 的服务 164

SunPlex Agent Builder 的输出 168

使用标准的 Sun Cluster 管理命令来创建使用 GDS 的服务 168

▼ 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 168

▼ 标准的 Sun Cluster 管理命令（用于创建使用 GDS 的、可伸缩的服务） 169

SunPlex Agent Builder 的命令行界面 170

▼ 使用 Agent Builder 的命令行版本创建使用 GDS 的服务 170

## 11 数据服务开发库参考 173

DSDL 函数 173

通用函数 173

特性函数 174

网络资源存取函数 175

使用 TCP 连接进行故障监视 175

PMF 函数 176

故障监视器函数 176

公用程序函数 176

## 12 CRNP 177

CRNP 概述 177

CRNP 协议概述 178

CRNP 使用的消息类型 180

客户机如何向服务器进行登记 181

管理员设置服务器的前提 181

服务器如何标识客户机 181

如何在客户机和服务器之间传送 SC\_CALLBACK\_REG 消息 181

服务器如何对客户机进行应答 183

SC_REPLY 消息的内容	183
客户机如何处理错误状态	184
服务器如何向客户机传送事件	184
如何保障事件的传送	185
SC_EVENT 消息的内容	185
CRNP 如何鉴别客户机和服务器	187
创建使用 CRNP 的 Java 应用程序	188
▼ 设置环境	188
▼ 开始	189
▼ 分析命令行参数	190
▼ 定义事件接收线程	191
▼ 登记和撤销登记回叫	192
▼ 生成 XML	192
▼ 创建登记消息和撤销登记消息	196
▼ 设置 XML 分析器	198
▼ 分析登记应答	199
▼ 分析回叫事件	201
▼ 运行应用程序	204

<b>A 标准特性</b>	<b>205</b>
资源类型特性	205
资源特性	210
资源组特性	217
资源特性属性	219
<b>B 数据服务样例代码列表</b>	<b>221</b>
资源类型登记文件列表	221
Start 方法	224
Stop 方法	227
gettime 公用程序	229
PROBE 程序	230
Monitor_start 方法	235
Monitor_stop 方法	237
Monitor_check 方法	238
Validate 方法	240
Update 方法	244

<b>C</b>	<b>数据服务开发库资源类型代码列表样例</b>	<b>247</b>
	xfnts.c	247
	xfnts_monitor_check 方法	259
	xfnts_monitor_start 方法	260
	xfnts_monitor_stop 方法	261
	xfnts_probe 方法	262
	xfnts_start 方法	265
	xfnts_stop 方法	266
	xfnts_update 方法	267
	xfnts_validate 方法的代码列表	269
<b>D</b>	<b>标准的 RGM 名称和值</b>	<b>271</b>
	RGM 标准名称	271
	RGM 的值	272
<b>E</b>	<b>对不支持群集的应用程序的要求</b>	<b>273</b>
	多主机数据	273
	将符号链接用于多主机数据放置	274
	主机名	274
	多地址主机	275
	绑定到 INADDR_ANY 与绑定到特定的 IP 地址	275
	客户机重试	276
<b>F</b>	<b>CRNP 的文档类型定义</b>	<b>277</b>
	SC_CALLBACK_REG XML DTD	277
	NVPAIR XML DTD	279
	SC_REPLY XML DTD	280
	SC_EVENT XML DTD	281
<b>G</b>	<b>CrnpClient.java 应用程序</b>	<b>283</b>
	CrnpClient.java 的内容	283
	<b>索引</b>	<b>305</b>

# 序

---

《*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。

---

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

---

---

**注意** – Sun Cluster 软件在两种平台（SPARC 和 x86）上运行。除非在特殊章、节、说明、项目符号项、图、表或示例中另外指出，否则本文档中的信息均适用于这两种平台。

---

---

## 本手册的用途

本文档面向具有丰富的 Sun 软硬件知识的有经验的开发者。本书提供的信息是建立在假设您已对 Solaris™ 操作系统有所了解的基础上的。

---

## 本书的结构

《*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 概述开发数据服务所需的概念。
- 第 2 章 介绍有关开发数据服务的详细信息。

- 第 3 章 介绍升级资源类型和迁移资源时应了解的事项。
- 第 4 章 介绍如何访问组成资源管理 API (RMAPI) 的函数和回叫方法。
- 第 5 章 为 `in.named()` 应用程序为例提供了一个 Sun Cluster 数据服务样例。
- 第 6 章 概述组成数据服务开发库 (DSDL) 的应用程序编程接口。
- 第 7 章 介绍 DSDL 在设计和实现资源类型方面的典型用法。
- 第 8 章 介绍使用 DSDL 实现的资源类型样例。
- 第 9 章 介绍 SunPlex™ Agent Builder。
- 第 10 章 介绍如何创建普通数据服务。
- 第 11 章 介绍 DSDL API 函数。
- 第 12 章 介绍群集重配置通知协议 (CRNP)。CRNP 使故障切换和可伸缩应用程序能够“支持群集”。
- 附录 A 介绍资源类型、资源组和资源的标准特性。
- 附录 B 提供数据服务样例中的每个方法的完整代码。
- 附录 C 列出 `SUNW.xfnts()` 资源类型中的每个方法的完整代码。
- 附录 D 列出资源组管理器 (RGM) 名称和值的合法字符的要求。
- 附录 E 列出要具有高可用性，普通的、不支持群集的应用程序需要满足的要求。
- 附录 F 列出 CRNP 的文档类型定义。
- 附录 G 给出第 12 章中讨论的 `CrnpClient.java` 应用程序的完整内容。

---

## 相关文档

有关相关的 Sun Cluster 主题的信息，可从下表列出的文档中获得。  
<http://docs.sun.com> 中提供了所有的 Sun Solaris 文档。

主题	文档
概念	《Sun Cluster 概念指南 (适用于 Solaris OS)》
概述	《Sun Cluster 概述 (适用于 Solaris OS)》
硬件管理	<i>Sun Cluster 3.x Hardware Administration Manual for Solaris OS</i> 各个硬件管理指南
软件安装	《Sun Cluster 软件安装指南 (适用于 Solaris OS)》

主题	文档
数据服务管理	《Sun Cluster 数据服务规划和管理指南 (适用于 Solaris OS)》 各个数据服务指南
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

<http://docs.sun.com> 上 Sun Cluster 的发行版的发行说明中提供了 Sun Cluster 文档的完整列表。

## 获得帮助

如果您在安装或使用 Sun Cluster 时有任何问题，请与您的服务供应商联系并提供以下信息：

- 您的姓名和电子邮件地址（如果有）
- 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）

使用以下命令收集您的系统信息，将这些信息提供给服务供应商。

命令	功能
<code>prtconf -v</code>	显示系统内存的大小并报告有关外围设备的信息
<code>psrinfo -v</code>	显示有关处理器的信息
<code>showrev -p</code>	报告已安装了哪些修补程序
<code>SPARC: prtdiag -v</code>	显示系统诊断信息
<code>/usr/cluster/bin/scinstall -pv</code>	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。

---

## 联机访问 Sun 文档

可以通过 docs.sun.com<sup>SM</sup> Web 站点联机访问 Sun 技术文档。您可以浏览 docs.sun.com 档案或查找某个具体的书名或主题。URL 是 <http://docs.sun.com>。

---

## 订购 Sun 文档

Sun Microsystems 提供一些印刷的产品文档。有关文档列表以及如何订购它们，请参阅 <http://docs.sun.com> 上的“购买印刷的文档”。

---

## 印刷惯例

下表介绍了本书中使用的印刷惯例。

表 P-1 印刷惯例

字体或符号	含义	示例
AaBbCc123	命令、文件和目录的名称；计算机屏幕输出	编辑 .login 文件。 使用 <code>ls -a</code> 列出所有文件。  machine_name% you have mail.
<b>AaBbCc123</b>	您键入的内容，与计算机屏幕输出的内容相对照	machine_name% <b>su</b> Password:
<i>AaBbCc123</i>	命令行通配符：用实际名称或实际值替换	要删除文件，请键入 <b>rm filename</b> 。
<i>AaBbCc123</i>	书名、新词、术语或要强调的词。	请参见《 <b>用户指南</b> 》第 6 章。 这些称为 <b>类选项</b> 。 <b>必须是超级用户才能执行此操作。</b>

---

## 命令示例中的 shell 提示符

以下表格显示了 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

---

本书对如何为软件应用程序（例如，Oracle®、Sun Java™ System Web Server [以前称为 Sun™ ONE Web Server]、DNS 等）创建资源类型提供了准则，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理解的概念，并包含以下信息：

- 第 17 页 “Sun Cluster 应用程序环境”
- 第 18 页 “RGM 模型”
- 第 20 页 “资源组管理器”
- 第 20 页 “回叫方法”
- 第 21 页 “编程接口”
- 第 22 页 “资源组管理器管理接口”

---

注意 – 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**（尽管在本书中很少用到）也与**资源类型**和**数据服务**表示的含义相同。

---

---

## Sun Cluster 应用程序环境

Sun Cluster 系统允许应用程序作为具有高可用性和可伸缩性的资源运行和管理。作为资源组管理器（即 RGM）的群集工具提供了一种具有高可用性和可伸缩性的机制。构成此工具编程接口的元素包括以下内容。

- 一组您编写的回叫方法，用于启用 RGM 以控制群集上的应用程序
- 资源管理 API (RMAPI)，一组可用于编写回叫方法的低级 API 命令和函数。这些 API 在 `libscha.so` 库中实现。
- 用于监视和重启群集上进程的进程管理工具
- 数据服务开发库 (DSDL)，一组库函数，用于封装低级 API 和高级进程管理功能以及添加某些其它功能以便于编写回叫方法。这些功能是在 `libdsdev.so` 库中实现。

下图展示了这些元素之间的相互关系。

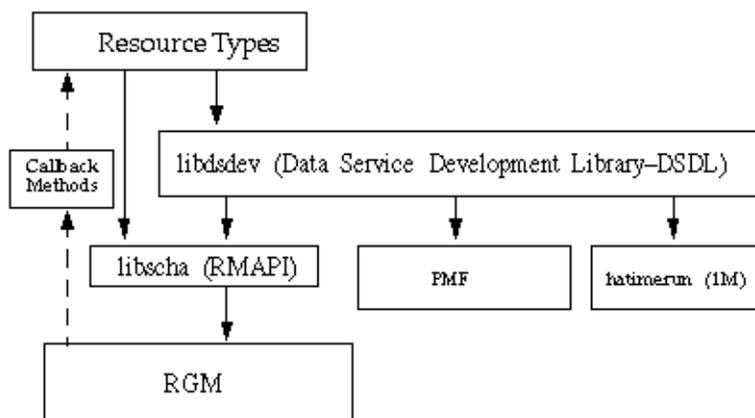


图 1-1 编程体系结构

Sun Cluster 软件包中包含的 SunPlex™ Agent Builder 是一种自动处理数据服务创建过程的工具（请参阅第 9 章）。Agent Builder 在 C shell（使用 DSDL 函数编写回叫方法）或 Korn shell (ksh)（使用低级 API 命令编写回叫方法）中生成数据服务代码。

RGM 在每个群集节点上作为守护程序运行，并根据预先配置的策略在选定节点上自动启动和停止资源。RGM 可以通过在受影响的节点上停止资源并在另一个节点上启动该资源，使该资源在节点失效或重新引导时具有高可用性。RGM 还可以自动启动和停止特定于资源的监视器，这些监视器可以检测到资源失效并将失效的资源重新定位到另一个节点，还可以监视资源性能的其他方面。

RGM 同时支持故障切换资源（一次只能在一个节点上联机）和可伸缩资源（可同时在多个节点上联机）。

## RGM 模型

本节介绍了一些基本术语并详细解释了 RGM 及其关联接口。

RGM 用于处理三种主要类型的相互关联对象：资源类型、资源和资源组。引入这些对象的一种方式是通过示例，如下所示。

开发者实现了资源类型 ha-oracle，使现有的 Oracle DBMS 应用程序具有高可用性。最终用户为营销、工程和财务定义了独立的数据库，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在独立的资源组中，以便它们可以在不同的节点上运

行以及分别进行故障切换。开发者创建了第二种资源类型 `ha-calendar`，以实现需要使用 Oracle 数据库的高可用日历服务器。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切换。

## 资源类型

资源类型由以下内容组成：在群集上运行的软件应用程序、由 RGM 用作回叫方法以管理作为群集资源的应用程序的控制程序以及一组构成部分静态群集配置的特性。RGM 使用资源类型特性来管理特定类型的资源。

---

**注意** – 除了软件应用程序之外，资源类型还可以表示其它系统资源，例如网络地址。

---

资源类型开发者指定资源类型的特性并在资源类型登记 (RTR) 文件中设置它们的值。RTR 文件遵循第 29 页“设置资源和资源类型特性”和 `rt_reg(4)` 手册页中介绍的已明确定义的格式。有关资源类型登记文件样例的说明，另请参阅第 74 页“定义资源类型登记文件”。

表 A-1 提供了资源类型特性的列表。

群集管理员在群集上安装和登记资源类型实现和基础应用程序。登记过程会将资源类型登记文件中的信息输入群集配置。《*Sun Cluster 数据服务规划和管理指南 (适用于 Solaris OS)*》中介绍了登记数据服务的过程。

## 资源

资源将继承其资源类型的特性和值。此外，开发者可以在资源类型登记文件中声明资源特性。要获得资源特性的列表，请参阅表 A-2。

根据某些特性在资源类型登记 (RTR) 文件中的指定方式，群集管理员可以更改这些特性的值。例如，特性定义可以指定允许值的范围以及特性何时可调（例如，创建时、随时或从不）。在这些规范中，群集管理员可以使用管理命令对特性进行更改。

群集管理员可以创建很多相同类型的资源，每个资源都具有自己的名称和特性值集，以便基础应用程序的多个实例都可以在群集上运行。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。RGM 使资源组联机或脱机时，它将调用组中各个资源上的回叫方法。

资源组当前联机的节点称为其**主要节点**或**主节点**。资源组由它的每个主要节点控制。每个资源组都具有一个关联的 `Nodelist` 特性，该特性由群集管理员进行设置，用来标识资源组所有的**潜在主节点**或**主控节点**。

资源组还具有一组特性。这些特性包括配置特性（可以由群集管理员进行设置）和动态特性（由 RGM 进行设置，可以反映资源组的活动状态）。

RGM 定义了两种类型的资源组：故障切换和可伸缩。故障切换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特性。有关这些特性的详细信息，请参阅第 29 页“将数据服务传送到群集”和第 36 页“实现回叫方法”。

要获得资源组特性的列表，请参阅表 A-3。

---

## 资源组管理器

资源组管理器 (RGM) 作为守护程序 `rgmd` 实现，该守护程序在群集的每个成员节点上运行。所有的 `rgmd` 进程之间相互通信，并一起作为一个适用于整个群集的工具。

RGM 支持以下功能：

- 节点引导或崩溃时，RGM 尝试通过自动使所有管理的资源组在相应的主控节点上联机来维护它们的可用性。
- 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
  - 启用或禁用资源组中的特定资源
  - 创建、删除或修改资源、资源组或资源类型

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。这种活动称为重新配置。要使状态更改在单个资源上生效，RGM 将调用该资源上特定于资源类型的回叫方法。

---

## 回叫方法

Sun Cluster 框架使用回叫机制提供数据服务与 RGM 之间的通信。该框架定义了一组回叫方法（包括变量和返回值）以及 RGM 调用每种方法的环境。

通过对一组单独的回叫方法进行编码并实现每种方法作为 RGM 可调用的控制程序，可以创建数据服务。也就是说，数据服务包含的不是单个可执行文件，而是大量可执行脚本 (`ksh`) 或二进制文件 (C)，每个脚本或二进制文件都可以直接由 RGM 调用。

回叫方法通过 RGM 在资源类型登记 (RTR) 文件中进行登记。在 RTR 文件中，可以标识已为数据服务实现的每种方法的程序。系统管理员在群集上登记数据服务时，RGM 将读取 RTR 文件，该文件将提供回叫程序的标识及其它信息。

资源类型唯一需要的回叫方法是 start 方法 (Start 或 Prenet\_start) 和 stop 方法 (Stop 或 Postnet\_stop)。

回叫方法可以分为以下几类：

- 控制和初始化方法
  - Start 和 Stop 分别用于启动和停止联机或脱机的组中的资源。
  - Init、Fini 和 Boot 分别用于执行资源上的初始化和终止代码。
- 管理支持方法
  - Validate 用于检验由管理操作设置的特性。
  - Update 用于更新联机资源的特性设置。
- 与网络相关的方法
  - Prenet\_start 和 Postnet\_stop 分别用于在将同一资源组中的网络地址配置为启用之前执行特殊的启动操作，或在将同一资源组中的网络地址配置为关闭之后执行特殊的关闭操作。
- 监视器控制方法
  - Monitor\_start 和 Monitor\_stop 启动或停止对某个资源的监视。
  - Monitor\_check 用于在将资源组移至节点之前评估该节点的可靠性。

关于回叫方法的详细信息，请参阅第 4 章和 `rt_callbacks(1HA)` 手册页。要获得数据服务样例中的回叫方法，另请参阅第 5 章和第 8 章。

---

## 编程接口

为了编写数据服务代码，资源管理体系结构提供了低级的基本 API、在基本 API 基础上生成的高级库以及 SunPlex Agent Builder 工具（该工具自动根据您提供的基本输入生成数据服务）。

## RMAPI

RMAPI（资源管理 API）提供了一组低级例行程序，使数据服务可以访问系统中有关资源、资源类型和资源组的信息，请求本地重新启动或故障切换以及设置资源状态。您可以通过 `libscha.so` 库访问这些功能。RMAPI 以 shell 命令和 C 函数的形式提供这些回叫方法。有关 RMAPI 例行程序的详细信息，请参阅 `scha_calls(3HA)` 和第 4 章。要获得如何在数据服务回叫方法样例中使用这些例行程序的示例，另请参阅第 5 章。

## 数据服务开发库 (DSDL)

DSDL 位于 RMAPI 顶端，它在保留 RGM 的基础方法回叫模型的同时提供了一种高级集成框架。DSDL 同时提供了多种数据服务开发工具，其中包括：

- `libscha.so`—低级资源管理 API
- PMF — 进程管理工具，提供了监视进程及其子进程以及无响应时重新启动这些进程的方法（请参阅 `pmfadm(1M)` 和 `rpc.pmf(1M)`）。
- `hatimerun`—在超时的情况下运行程序的工具（请参阅 `hatimerun(1M)`）。

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。实际上，很多 DSDL 函数都调用 `libscha.so` 函数。同样，您可以在使用 DSDL 对数据服务进行大量编码时直接调用 `libscha.so` 函数。`libdsdev.so` 库中包含了 DSDL 函数。

有关 DSDL 的详细信息，请参阅第 6 章和 `scha_calls(3HA)` 手册页。

## SunPlex Agent Builder

Agent Builder 是一种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 将生成数据服务，由资源和可执行码（C 或 Korn shell）、自定义 RTR 文件和 Solaris™ 软件包组成。

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，对附加特性添加验证检查）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，也可以使用 Agent Builder 生成大量代码并对其余内容手动进行编码。至少可以使用 Agent Builder 为您生成 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和一组用于管理群集的命令。

## SunPlex Manager

SunPlex Manager 是基于 Web 的工具，可以用来执行以下任务：

- 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 和如何使用 SunPlex Manager 安装群集软件的说明，请参阅《*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。

## 管理命令

用于管理 RGM 对象的 Sun Cluster 命令有 `scrgadm(1M)`、`scswitch(1M)` 和 `scstat(1M)-g`。

通过 `scrgadm` 命令可以查看、创建、配置和删除 RGM 使用的资源类型、资源组和资源对象。该命令是群集管理接口的一部分，但不会和本章其余部分所介绍的应用程序接口在同一编程上下文中使用。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。有关该命令可以执行的管理任务的详细信息，请参阅 `scrgadm(1M)` 手册页。

`scswitch` 命令用于切换资源组在指定节点上的联机和脱机，以及启用或禁用资源或其监视器。有关该命令可以执行的管理任务的详细信息，请参阅 `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。



## 第 2 章

---

# 开发数据服务

---

本章介绍了有关开发数据服务的详细信息。

本章包括以下主题：

- 第 25 页 “分析应用程序的适用性”
- 第 27 页 “确定要使用的接口”
- 第 28 页 “设置用来编写数据服务的开发环境”
- 第 29 页 “设置资源和资源类型特性”
- 第 36 页 “实现回叫方法”
- 第 37 页 “普通数据服务”
- 第 38 页 “控制应用程序”
- 第 40 页 “监视资源”
- 第 41 页 “向资源添加消息日志”
- 第 41 页 “提供进程管理”
- 第 42 页 “提供资源的管理支持”
- 第 42 页 “实现故障切换资源”
- 第 43 页 “实现可伸缩资源”
- 第 45 页 “编写和测试数据服务”

---

## 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。如果该应用程序不满足所有要求，您可以修改应用程序的源代码，使其满足要求。

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需要修改应用程序源代码，请参阅附录 B。

---

**注意** – 要具有高可用性，可伸缩服务必须满足以下所有条件，同时还需满足一些附加条件。

---

- 在 Sun Cluster 环境中，支持网络（客户机服务器模型）的应用程序和不支持网络（无客户机）的应用程序都可以设置为具有高可用性或可伸缩性。然而在分时环境（多个应用程序在通过 telnet 或 rlogin 访问的服务器上运行）中，Sun Cluster 无法提供增强的可用性。
- 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下启动应用程序，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。不要求数据服务可以恢复连接。
- 应用程序不能依赖于它在其上运行的节点的物理主机名。有关其它信息，请参阅第 274 页“主机名”。
- 应用程序必须在配置为启用了多个 IP 地址的环境下正确操作。该环境可以是具有多头主机环境，在该环境中节点位于多个公共网络中；或者是包含多个节点的环境，在节点上，一个硬件接口上配置启用多个逻辑接口。
- 要具有高可用性，应用程序数据必须驻留在群集文件系统中。请参阅第 273 页“多主机数据”。

如果应用程序为该数据的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个位置的符号链接，而无需更改应用程序源代码。有关其它信息，请参阅第 274 页“将符号链接用于多主机数据放置”。
- 应用程序二进制数和库都可以驻留在本地的每个节点上或群集文件系统中。将其驻留在群集文件系统中的优点在于进行单个安装就已足够。缺点在于轮询升级将成为问题，因为当应用程序在 RGM 的控制下运行时，二进制数正在使用。
- 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，则接下来将处理有关资源组进行故障切换或切换的问题。有关其它信息，请参阅第 276 页“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，可伸缩服务还必须满足以下要求。

- 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，允许任意实例对客户机请求作出反应的负载平衡策略 LB\_WEIGHTED 不适用于使用服务器内存中的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详细信息，请参阅第 42 页“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)，一组低层次的例行程序（位于 `libscha.so` 库中）。
- 数据服务开发库 (DSDL)，一组较高层次的函数（位于 `libdsdev.so` 库中），用来封装 RMAPI 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。

建议在开发数据服务时采用的方法有：

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，指定所请求的输入并生成数据服务，该服务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代码的方法，您还可以用一个您使用 DSDL 或 RMAPI 函数从头编写的程序，完全替换所生成的方法之一或生成的监视程序。但是，无论采用哪种方法，绝大多数情况下都要使用 Agent Builder 开始进行操作，采用这种方法比较好，原因如下：

- Agent Builder 生成的代码（本质上是通用的）已在多种数据服务中进行测试。
- Agent Builder 将生成 RTR 文件、make 程序的描述文件、资源的软件包以及数据服务的其它支持文件。即使您没有使用任何数据服务代码，使用这些其它文件也可以减少您相当一部分工作量。
- 您可以修改生成的代码。

---

**注意** – 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

在开始进行数据服务开发之前，必须先安装 Sun Cluster 开发软件包 (SUNWscdev)，以便可以访问 Sun Cluster 头文件和库文件。虽然此软件包已安装在所有群集节点上，但是通常却在独立的非群集开发机器上进行开发，而不是在群集节点上进行。在通常情况下，您必须使用 `pkgadd` 在开发机器上安装 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。在非群集节点上完成开发后，您可以将完整的数据服务传送到群集，以便运行和测试。

---

**注意** – 请确保使用的是 Solaris 5.8 或更高版本的开发版本。

---

请使用本小节中的各个步骤执行以下操作：

- 安装 Sun Cluster 开发软件包 (SUNWscdev) 并设置相应的编译器和链接程序选项
- 将数据服务传送到群集

### ▼ 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。

1. 成为超级用户或采用等效角色并将目录更改为所需的 CD-ROM 目录。

```
# cd CD-ROM_directory
```

2. 在当前目录下安装 SUNWscdev 软件包。

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

3. 在 `Makefile` 中，指定用于标识数据服务代码的包含文件和库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；指定 `-L` 选项，用于指定开发系统上的编译时库搜索路径；指定 `-R` 选项，用于指定指向群集上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

在开发机器上完成数据服务的开发后，您必须将其传送到群集以进行测试。要降低发生错误的几率，最好是先将数据服务代码和 RTR 文件打包，然后再将软件包安装在群集的所有节点上来完成此传送过程。

---

**注意** – 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。Agent Builder 将自动把 RTR 文件和数据服务代码打包。

---

## 设置资源和资源类型特性

Sun Cluster 提供了一组用来定义数据服务静态配置的资源类型特性和资源特性。资源类型特性用来指定资源的类型及其版本、API 的版本等，还可以用来指定指向每个回叫方法的路径。表 A-1 列出了所有资源类型特性。

资源特性（例如，`Failover_mode`、`Thorough_probe_interval`）和方法超时也定义该资源的静态配置。动态资源特性（例如 `Resource_state` 和 `Status`）反映被管理资源的活动状态。表 A-2 介绍了各种资源特性。

资源类型和资源特性在资源类型登记 (RTR) 文件中声明，该文件是数据服务的基本组件。RTR 文件用来在群集管理员使用 Sun Cluster 登记数据服务时定义数据服务的初始配置。

建议您使用 Agent Builder 生成数据服务的 RTR 文件，因为 Agent Builder 声明了一组对于任何数据服务来说既有用又必需的特性。例如，有些特性（如 `Resource_type`）必须在 RTR 文件中声明，否则数据服务的登记将失败。其它特性尽管没有此方面的要求，但是除非您在 RTR 文件中进行声明，否则系统管理员将无法使用它们；而一些特性无论是否进行声明都可用，因为 RGM 对它们进行了定义并为其提供了缺省值。要避免发生这种复杂的情况，您只需使用 Agent Builder，以确保生成适当的 RTR 文件。如果需要，今后您可以编辑该 RTR 文件，以更改具体值。

本小节的其余内容介绍了 Agent Builder 创建的 RTR 文件样例。

## 声明资源类型特性

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

---

**注意** – 资源类型特性 `Installed_nodes` 可由系统管理员配置。实际上，此特性只能由系统管理员配置，而您无法在 RTR 文件中对其进行声明。

---

资源类型声明的语法为：

```
property_name = value;
```

---

**注意** – RGM 认为特性名是不区分大小写的。由 Sun 提供的 RTR 文件中的特性名称惯例是将名称的首字母大写，其余字母小写（方法名除外）。方法名以及特性的属性中包含的都是大写字母。

---

以下是 RTR 文件中为 `smpl` 数据服务声明的资源类型：

```
# 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_PRIMARYES;

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` 特性声明为 RTR 文件中的第一项。否则，资源类型的登记将失败。

---

第一组资源类型声明可提供有关资源类型的基本信息，如下所示：

`Resource_type` 和 `Vendor_id` 提供资源类型的名称。您可以仅使用 `Resource_type` 特性指定资源类型名称 (`smpl`)，或在该特性前用 `Vendor_id` 作为前缀，前缀与资源类型之间用“.”分隔 (`SUNW.smpl`)，如样例中所示。如果使用 `Vendor_id`，请将其设置为用来定义资源类型的公司的股票代码。资源类型名称在群集中必须是唯一的。

---

**注意** – 按照惯例，资源类型名称 (`Resource_typeVendor_id`) 用作软件包名称。尽管 RGM 没有强制要求进行限制，但因为软件包名称的字符数不能超过九个，因此将这两个特性的字符总数限制在九个或更少字符数以内不失为一种好方法。另一方面，**Agent Builder** 将明确地根据该资源类型名称生成软件包名称，因此它要强制执行九个字符数限制。

---

`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` 表明数据服务在 Sun Cluster 3.0 版本中运行。

`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，即指定了一个故障切换数据服务。有关详细信息，请参阅第 29 页“将数据服务传送到群集”。

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

其余的资源类型声明用于提供配置信息，如下所示：

`Init_nodes = RG_PRIMARYES` 指定 RGM 仅对可以控制数据服务的节点调用 `Init`、`Boot`、`Fini` 和 `Validate` 方法。`RG_PRIMARYES` 指定的节点是安装了数据服务的所有节点的子集。将值设置为 `RT_INSTALLED_NODES` 可指定 RGM 对安装有数据服务的所有节点调用这些方法。

RT_basedir	指向 /opt/SUNWsample/bin, 作为指向完整相对路径 (例如回叫方法路径) 的目录路径。
Start、Stop、Validate 等	提供指向由 RGM 调用的各个回叫方法程序的路径。这些路径是基于 RT_basedir 所指定的目录的相对路径。

## 声明资源特性

与资源类型特性一样, 资源特性也在 RTR 文件中声明。按照惯例, 在 RTR 文件中资源特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;
}
```

对于由 Sun Cluster 提供的资源特性, 即所谓**系统定义**的特性, 您可以在 RTR 文件中更改具体属性。例如, Sun Cluster 为每个回叫方法提供方法超时特性, 并指定缺省值。在 RTR 文件中, 您可以指定各种缺省值。

您也可以使用一组由 Sun Cluster 提供的特性属性, 在 RTR 文件中定义新的资源特性, 即所谓的**扩展**特性。表 A-4 中列出了用于更改和定义资源特性的属性。在 RTR 文件中, 扩展特性声明位于系统定义的特性声明之后。

第一组系统定义的资源特性用于指定回叫方法的超时值:

```
...

# 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;
    }
}

```

该特性的名称 (PROPERTY = *value*) 必须是每个资源特性声明的第一个属性。您可以在根据 RTR 文件中的特性属性定义的限制范围内配置资源特性。例如，样例中每个方法超时的缺省值都是 300 秒。管理员可以更改此值；但是 MIN 属性指定的最小允许值为 60 秒。要获得资源特性属性的完整列表，请参阅表 A-4。

下一组资源特性用于定义在数据服务中有特定用法的特性。

```

{
    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 = AT_CREATION;
    DEFAULT = ;
}

```

这些资源特性声明将添加 TUNABLE 属性，该属性用于限制系统管理员在什么情况下可以更改特性值。AT\_CREATION 表明管理员只能在创建资源时指定该值，并且今后无法进行更改。

对于大多数特性，您可以接受由 Agent Builder 生成的缺省值，除非您有理由更改它们。下面是有关这些特性的信息（有关其它信息，请参阅第 210 页“资源特性”或 r\_properties(5) 手册页）：

#### Failover\_mode

表示在 Start 或 Stop 方法失败的情况下，RGM 是否应重定位资源组或中断节点。

#### Thorough\_probe\_interval、Retry\_count、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，系统管理员可以对这些特性进行调整。

#### Network\_resources\_used

数据服务使用的逻辑主机名或共享地址资源列表。Agent Builder 将声明此特性，因此配置数据服务时，系统管理员可以指定资源列表（如果有）。

### Scalable

设置为 `FALSE` 可表明此资源不使用群集联网（共享地址）工具。此设置与设为 `TRUE` 的资源类型 `Failover` 特性相符，用来表明故障切换服务。有关如何使用此特性的其它信息，请参阅第 29 页“将数据服务传送到群集”和第 36 页“实现回叫方法”。

### Load\_balancing\_policy、Load\_balancing\_weights

将自动声明这些特性，但是在故障切换资源类型中不能发挥作用。

### Port\_list

标识服务器侦听的端口的列表。`Agent Builder` 将声明此特性，因此系统管理员可以在配置数据服务时指定端口列表。

## 声明扩展特性

位于 `RTR` 文件样例结尾处的是扩展特性，如下所示：

```
# 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, 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 创建了一些对大多数的数据服务都有用的扩展特性，如下所示。

#### 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。配置数据服务时，系统管理员可以提供此目录的位置。

#### Monitor\_retry\_count、Monitor\_retry\_interval、Probe\_timeout

用于控制故障监视器本身的重启操作，而不是服务器守护程序的重启操作。

#### Child\_mon\_level

用于设置 PMF 要完成的监视操作的级别。有关详细信息，请参阅 pmfadm(1M)。

您可以在由用户添加的代码注释分隔的区域内创建其它扩展特性。

---

## 实现回叫方法

本小节介绍了有关实现回叫方法的信息。

## 访问资源和资源组特性信息

一般情况下，回叫方法需要对资源的特性进行访问。RMAPI 同时提供了可在回叫方法中使用的 shell 命令和 C 函数，以访问系统定义的资源特性和扩展资源特性。请参阅 scha\_resource\_get(1HA) 和 scha\_resource\_get(3HA) 手册页。

DSDL 提供了一组用来访问系统定义的特性的 C 函数（一种特性对应一种函数），还提供了一种用来访问扩展特性的函数。请参阅 `scds_property_functions(3HA)` 和 `scds_get_ext_property(3HA)` 手册页。

您无法使用特性机制来存储数据服务的动态状态信息，因为没有可用于设置资源特性（除了 `Status` 和 `Status_msg`）的 API 函数。您应该将动态状态信息存储在全局文件中。

---

**注意** – 使用 `scrgadm` 命令，或者通过可用的图形管理命令或图形管理界面，群集管理员可以设置某些资源特性。但是，请勿从任何回叫方法中调用 `scrgadm`，因为 `scrgadm` 将在重新配置群集过程中（即当 RGM 调用方法时）失败。

---

## 方法的幂等性

通常情况下，RGM 不会使用相同的变量对同一资源连续多次调用某个方法。但是，如果 `Start` 方法失败了，即使从未启动过该资源，RGM 也会对其调用 `Stop` 方法。同样地，如果资源守护程序主动停止运行，RGM 可能仍会对该程序调用 `Stop` 方法。相同的情况也适用于 `Monitor_start` 和 `Monitor_stop` 方法。

因为这些原因，您必须在 `Stop` 和 `Monitor_stop` 方法中内置幂等性。使用相同参数对同一资源重复调用 `Stop` 或 `Monitor_stop` 的结果与进行单一调用的结果相同。

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。

---

**注意** – `Init`、`Finis`、`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) 是一种使简单的应用程序具有高可用性或可伸缩性的机制，方法是将应用程序插入到 Sun Cluster 资源组管理器框架中。此机制不需要为代理编写代码，而编写代码是使应用程序具有高可用性或可伸缩性的通常做法。

GDS 模型依赖于预先编译的资源类型 `SUNW.gds`，以与 RGM 框架进行交互操作。

有关其它信息，请参阅第 10 章。

---

## 控制应用程序

每当节点加入或退出群集时，回叫方法将启用 RGM 来控制基础资源（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 将以适当的次数对适当的节点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 将把该节点控制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在其 `Start` 方法中设置了足够的超时值（在资源类型登记文件中设置 `Start_timeout` 特性的缺省值和最小值）。

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。当使资源组脱机时，RGM 对组中的资源调用 `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` 方法的最安全实现操作将终止与该资源相关的本地节点上的所有进程。应该在需要较长时间进行关闭的资源类型的 `Stop` 方法中为其设置足够的超时值。在资源类型登记文件中设置 `Stop_timeout` 特性。

`Stop` 方法的失败和超时将导致资源组进入需要操作员介入的错误状态。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。

### 决定使用哪种 `Start` 或 `Stop` 方法

本节将对照 `Preinet_start` 和 `Postnet_stop` 方法的使用，介绍一些有关何时使用 `Start` 和 `Stop` 方法的提示。要正确决定要使用的方法，您必须对客户机和数据服务的客户机服务器网络协议有深入的了解。

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关的特定顺序来执行启动或停止步骤。可选的回叫方法 `Preinet_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，而可选的回叫方法 `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 `Preinet_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 使资源组联机的顺序如下。

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。

当决定是否使用 `Start`、`Stop`、`Prenet_start` 或 `Postnet_stop` 方法时，首先要考虑服务器端。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，要启动或停止数据服务，您可能需要调用该数据服务的管理公用程序或库。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公用程序或库。即管理公用程序将调用服务器守护程序，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。在这种情况下请使用 `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` 变量的 `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作出不同反应。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。这样可以确保数据服务在完成将网络地址配置为启用后，立刻对客户机的请求作出反应，并且客户机不太可能停止重试。在这种情况下，请不要使用 `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。只有在网络地址配置为关闭后，才会调用 `Postnet_stop` 方法。结果是，数据服务的 TCP 或 UDP 服务端口或其 RPC 程序号似乎始终可以供网路上的客户机使用（除了该网络地址也未作出反应的情况）。

与使用 `Prenet_start` 和 `Postnet_stop` 方法相比，决定使用 `Start` 和 `Stop` 方法时，或决定同时使用这两套方法时，必须将服务器和客户机的要求和行为方式都考虑在内。

## 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。当资源处于被管理状态时（一种情况是该资源所在的资源组从非管理状态切换为被管理状态时，另一种情况是该资源在已处于被管理状态下的资源组中创建时），RGM 将调用 Init 方法对该资源执行一次性初始化。

当资源处于非管理状态时（一种情况是该资源所在的资源组切换为非管理状态时，另一种情况是该资源从被管理资源组中删除时），RGM 将调用 Fini 方法清除该资源。清除操作必须具有幂等性，即如果已执行了清除操作，Fini 将在返回 0（成功）的情况下退出。

RGM 将对新加入群集的节点（即已对该节点进行了引导或重新引导操作）调用 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，即如果该资源已在本地节点上进行了初始化，Boot 和 Init 将在返回 0（成功）的情况下退出。

---

## 监视资源

通常需要使用监视器，以对资源运行周期性故障探测程序来检测所测的资源是否工作正常。如果故障探测程序失败，监视器可尝试在本地进行重启，或通过调用 `scha_control()` RMAPI 函数或 `scds_fm_action()` DSDL 函数请求对受影响的资源组进行故障切换。

您也可以监视资源的性能并进行调节或报告性能。是否编写特定于资源类型的故障监视器完全是可选的操作。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，但它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()`）允许资源监视器向其它节点发出资源组的故障切换请求。作为一种合理性检查，`scha_control()` 将调用 `Monitor_check`（如果进行了定义）来确定被请求节点在控制包含该资源的资源组方面是否可靠。如果 `Monitor_check` 报告该节点不可靠或方法超时，RGM 将寻找其它节点来接受故障切换请求。如果 `Monitor_check` 在所有节点上都失败，将取消故障切换。

资源监视器可以设置 `Status` 和 `Status_msg` 特性以反映监视器查看到的资源状态。请使用 RMAPI `scha_resource_setstatus()` 函数或 `scha_resource_setstatus` 命令或 DSDL `scds_fm_action()` 函数来设置这些特性。

---

**注意** – 尽管对于资源监视器来说，`Status` 和 `Status_msg` 具有特定的作用，但是也可以使用任意程序来设置这些特性。

---

要想获得使用 RMAPI 实现的故障监视器实例，请参阅第 88 页“定义故障监视器”。要想获得使用 DSDL 实现的故障监视器实例，请参阅第 127 页“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，请参阅《*Sun Cluster 数据服务规划和管理指南（适用于 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它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叫。RMAPI 定义了以下工具（有关其中每一个命令和程序的详细信息，请参阅手册页）：

进程监视工具：

<code>pmfadm</code> 和 <code>rpc.pmf</code>	进程监视工具 (PMF)，提供了一种用来监视进程及其子进程以及在失败后重启进程的方法。该工具由 <code>pmfadm</code> 命令（用于启动和控制被监视的进程）和 <code>rpc.pmf</code> 守护程序组成。
<code>halockrun</code>	一种用于在保留文件锁定时运行子程序的程序。此命令在 <code>shell</code> 脚本中使用很方便。
<code>hatimerun</code>	一种用于在超时控制下运行子程序的程序。此命令在 <code>shell</code> 脚本中使用很方便。

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

DSDL 提供了一组用来实现 PMF 功能的函数 (scds\_pmf\_\*)。要获得 DSDL PMF 功能的概述以及各种函数的列表，请参阅第 176 页“PMF 函数”。

---

## 提供资源的管理支持

对资源进行的管理操作包括设置和更改资源特性。API 定义了 `Validate` 和 `Update` 回叫方法，因此您可以钩住这些管理操作。

当创建资源时以及通过管理操作更新资源特性或其包含的资源组时，RGM 将调用可选的 `Validate` 方法。RGM 将该资源及其资源组的特性值传送到 `Validate` 方法。RGM 将对由该资源类型的 `Init_nodes` 特性指定的那组群集节点调用 `Validate`。（有关 `Init_nodes` 的信息，请参阅第 205 页“资源类型特性”或 `rt_properties (5)` 手册页。）在进行创建或更新之前，RGM 将调用 `Validate`，而来自位于任意节点上的方法的故障出口代码将导致创建或更新操作失败。

仅当通过管理操作更改资源或组特性时（而不是在 RGM 设置特性时或监视器设置资源特性 `Status` 和 `Status_msg` 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特性进行了更改。在设置资源及其组的特性过程中成功地执行了管理操作后，RGM 将调用 `Update`。RGM 将对资源处于联机状态的节点调用此方法。此方法可用于通过 API 访问函数读取可能会影响活动资源的特性值，并相应地调整运行资源。

---

## 实现故障切换资源

故障切换资源组中包含网络地址（例如内置资源类型逻辑主机名和共享地址）和故障切换资源（例如故障切换数据服务的数据服务应用程序资源）。当数据服务发生故障切换或进行切换时，网络地址资源及其相关的数据服务资源将在群集节点间移动。RGM 提供了多个支持故障切换资源实现的特性。

将布尔资源类型特性 `Failover` 设置为 `TRUE` 可以限制资源，避免它在可同时在多个节点上处于联机状态的资源组中进行配置。此特性缺省为 `FALSE`，因此您必须在 RTR 文件中将其声明为 `TRUE`，以用于故障切换资源。

`Scalable` 资源特性用来确定该资源是否使用群集共享地址工具。对于故障切换资源，请将 `Scalable` 设置为 `FALSE`，因为故障切换资源不使用共享地址。

`RG_mode` 资源组特性允许群集管理员将资源组标识为故障切换或可伸缩。如果 `RG_mode` 的值为 `FAILOVER`，RGM 将把该组的 `Maximum primaries` 特性设置为 1，并将该资源组限制为由单个节点控制。RGM 不允许在 `RG_mode` 的值为 `SCALABLE` 的资源组中创建 `Failover` 特性是 `TRUE` 的资源。

`Implicit_network_dependencies` 资源组特性用来指定 RGM 应该对组内所有网络地址资源（逻辑主机名和共享地址）强制执行非网络地址资源的隐含强依赖性。这意味着将组中的网络地址配置为启用之前，组中的非网络地址（数据服务）资源不会调用其 `Start` 方法。`Implicit_network_dependencies` 特性值缺省为 `TRUE`。

---

## 实现可伸缩资源

可伸缩资源可以同时多个节点上处于联机状态。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特性。

相应地将布尔资源类型特性 `Failover` 设置为 `FALSE`，允许在可同时在多个节点上处于联机状态的资源组中配置资源。

`Scalable` 资源特性用来确定该资源是否使用群集共享地址工具。请将此特性设置为 `TRUE`，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作为单个服务向客户机显示。

`RG_mode` 特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` 的值是 `SCALABLE`，RGM 允许 `Maximum primaries` 的值大于 1，这意味着该组可同时由多个节点控制。RGM 允许 `Failover` 特性值是 `FALSE` 的资源在 `RG_mode` 的值是 `SCALABLE` 的资源组中实例化。

群集管理员将创建一个用来包含可伸缩服务资源的可伸缩资源组，还会创建一个用来包含可伸缩资源所依赖的共享地址资源的单独故障切换资源组。

群集管理员使用 `RG_dependencies` 资源组特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，管理员必须设置包含可伸缩服务的资源组的 `RG_dependencies` 特性，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` 特性时，RGM 将自动为该资源创建以下一组可伸缩特性：

`Network_resources_used` 用来标识此资源使用的共享地址资源。此特性缺省为空字符串，因此群集管理员在创建资源时，必须提供可伸缩服务所使用的共享地址的实际列表。

	<p>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。</p>
Load_balancing_policy	<p>用来指定资源的负载平衡策略。您可以在 RTR 文件中明确地设置该策略（或使用缺省值 LB_WEIGHTED）。在每一种情况下，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）。有效值为：</p> <p>LB_WEIGHTED 根据在 Load_balancing_weights 特性中设置的权重在不同的节点间分配负载。</p> <p>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。</p> <p>LB_STICKY_WILD 连接到通配粘滞服务的 IP 地址的某个给定客户机（由客户机的 IP 地址标识）总是发送到同一群集节点，而不管它来自哪个端口号。</p> <p>对于使用 Load_balancing_policy LB_STICKY 或 LB_STICKY_WILD 的可伸缩服务，在服务联机时更改 Load_balancing_weights 可能造成复位现有客户机关系。在这种情况下，一个不同的节点可能会处理后面的客户机请求（即使原来是由该群集中另一个节点处理客户机请求）。</p> <p>同样地，在群集上启动该服务的新实例可能也会重置现有的客户机关系。</p>
Load_balancing_weights	<p>用来指定要分配给每个节点的负载。格式为 <i>weight@node,weight@node</i>，其中 <i>weight</i> 是一个反映分配到指定 <i>node</i> 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重除以活动实例的所有权重的和。例如，1@1,3@2 表明节点 1 接收到 1/4 的负载，而节点 2 接收到 3/4 的负载。</p>
Port_list	<p>用来标识服务器所侦听的端口。此特性缺省为空字符串。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。</p>

您可以创建可由管理员配置为可伸缩或故障切换的数据服务。要进行此操作，请在数据服务的 RTR 文件中将 Failover 资源类型特性和 Scalable 资源特性都声明为 FALSE。创建时请将 Scalable 特性指定为可调。

Failover 特性值 (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，管理员可以启用共享地址，这样便创建了可伸缩服务。

从另一方面来说，即使将 `Failover` 的值设置为 `FALSE`，管理员也可以将资源配置到故障切换资源组中，以实现故障切换服务。管理员无需更改值为 `FALSE` 的 `Scalable`。要支持此操作，您必须对 `Scalable` 特性的 `Validate` 方法进行检查。如果 `Scalable` 的值是 `FALSE`，请检验是否已将该资源配置到故障切换资源组中。

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

## 可伸缩服务的验证检查

每当在将可伸缩特性值设置为 `TRUE` 的情况下创建资源或更新资源时，RGM 都将验证各种资源特性。如果特性值未正确配置，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` 特性必须非空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的 `Nodelist` 中的每个节点都必须显示在每个已命名共享地址资源的 `NetIfList` 特性或 `AuxNodeList` 特性中。
- 包含可伸缩资源的资源组的 `RG_dependencies` 特性必须包含所有共享地址资源（列在该可伸缩资源的 `Network_resources_used` 特性中）所属的资源组。
- `Port_list` 特性必须非空，并且包含端口协议对列表（协议可以是 TCP 或 UDP）。例如：

```
Port_list=80/tcp,40/udp
```

---

## 编写和测试数据服务

本节介绍一些有关编写和测试数据服务的信息。

### 使用持续连接机制

对于服务器端，使用 TCP 持续连接机制可以保护服务器，使其避免为关闭的（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不清除这些资源，当客户机崩溃或重新引导时，会导致浪费的资源量无限增长。

如果客户机和服务器使用 TCP 流进行通信，则它们都应该启用 TCP 持续连接机制。这也适用于非 HA 单服务器情况。

其它面向连接的协议可能也需要具有持续连接机制。

对于客户端，使用 TCP 持续连接机制，可以使该客户端在网络地址资源发生故障切换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，除非该客户端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，该客户端不一定会收到关于此情况的通知。

例如，假设客户端正在等待服务器对长时间运行请求的反应，且该客户端的请求消息已发送到服务器并已在 TCP 层得到了确认。在这种情况下，客户端的 TCP 模块无需不断重新发送请求，并且客户端应用程序处于阻塞状态，等待对该请求的反应。

在适当的情况下，客户端应用程序除了使用 TCP 持续连接机制以外，还必须在其级别上执行自己的周期性持续连接机制，因为 TCP 持续连接机制并不是在处理所有可能的边界问题上都是完美的。使用应用程序级别的持续连接机制通常要求客户端服务器协议支持一个空操作，或者至少支持一个有效的只读操作（例如状态操作）。

## 测试 HA 数据服务

本节给出有关如何在 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的 Sun Cluster 配置，以免测试工作影响生产机器。

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运行正常。这些情况包括系统崩溃以及使用 `scswitch` 命令的情况。测试客户端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

有时一个客户端服务器数据服务在满足某个客户端请求的同时，会向其它客户端服务器数据服务发出请求。用非正式的语言描述就是，如果数据服务 A 依赖于数据服务 B，则当 A 提供其服务时，B 也必须提供其服务。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，请参阅 `scrgadm(1M)` 手册页。

如果您的资源类型的资源依赖于其它类型的资源，则您必须指导用户适当地配置资源和资源组，或提供脚本或工具对其进行正确地配置。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。

决定是使用明确的资源依赖性，还是忽略它们，并对用您的 HA 数据服务自身代码编写的其它数据服务的可用性进行论询。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。在这种情况下，需要进行轮询，因为无法跨组配置资源依赖性。

有些数据服务自己不直接存储数据，而是依赖其它后端数据服务存储其全部数据。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，请考虑假设存在一个客户机服务器日程日历服务，该服务将所有数据保存在 SQL 数据库（例如，Oracle）中。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。这样您就可以编写用于启动或停止日程日历守护程序的简单方法。您的最终用户将使用 Sun Cluster 登记日程日历资源类型。

如果该日程日历应用程序必须和 Oracle 数据库在同一个节点上运行，则最终用户可以在配置 HA-ORACLE 资源的同一个资源组中配置日程日历资源，并使日程日历资源依赖于 HA-ORACLE 资源。这种依赖性是用 `scrgadm` 中的 `Resource_dependencies` 特性标记来指定的。

如果 HA-ORACLE 资源可以和日程日历资源运行在不同的节点上，则最终用户可以在两个不同的资源组中对它们进行配置。最终用户可以配置日历资源组对 Oracle 资源组的资源组依赖性。但是仅当在同一节点上同时启动或停止两个资源组时，资源组依赖性才有效。因此，日历数据服务守护程序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。在这种情况下，日历资源类型的 `Start` 方法通常仅返回成功消息，因为如果 `Start` 方法无限期地处于阻塞状态，它将使其资源组处于忙碌状态，这会阻止对该组进行进一步的状态更改（例如进行编辑、故障切换或切换）。但是，如果日历资源的 `Start` 方法超时或在非零的情况下退出，则可能会导致该资源组在两个或多个节点间交替切换，而这时 Oracle 数据库仍然不可用。



## 第 3 章

---

# 升级资源类型

---

本章介绍升级资源类型和迁移资源时应了解的事项。

- 第 49 页 “概述”
- 第 50 页 “资源类型登记文件”
- 第 52 页 “资源的 `Type_version` 特性”
- 第 53 页 “将资源迁移到其它版本”
- 第 53 页 “升级和降级资源类型”
- 第 55 页 “缺省特性值”
- 第 56 页 “资源类型开发者文档”
- 第 56 页 “资源类型名称和资源类型监视器实现”
- 第 57 页 “应用程序升级”
- 第 57 页 “资源类型升级实例”
- 第 59 页 “资源类型软件包的安装要求”

---

## 概述

系统管理员需要具备相应的能力，以便可以安装和登记现有资源的新版本，允许登记给定资源类型的多个版本，以及将现有资源迁移到资源类型的新版本，而无需删除和重新创建资源。资源开发者需要了解进行资源类型升级和资源迁移的要求。

资源类型在开发时考虑到升级问题称为**升级支持**。

资源类型的新版本与以前版本的不同之处可能有以下几个方面：

- 资源类型特性的属性可能发生更改
- 已声明的资源特性（包括标准特性和扩展特性）的设置可能发生更改
- 资源特性的属性（例如 `default`、`min`、`max`、`arraymin`、`arraymax` 或可调性）可能发生更改
- 已声明方法的设置可能产生不同

- 方法或监视器的实现可能发生变更。

资源类型开发者可以从以下可调性选项中进行选择，确定何时可以将现有资源迁移到新版本中。这些选项按限制性的高低（从低到高）列出：

- 随时 (Anytime)
- 当资源不受监视时 (When\_unmonitored)
- 当资源处于脱机状态时 (When\_offline)
- 当资源处于禁止状态时 (When\_disabled)
- 当资源组处于不受管理状态时 (When\_unmanaged)
- 在创建时 (At\_creation)

有关每个选项的说明，请参阅第 52 页“资源的 `Type_version` 特性”。

---

**注意** – 在本章中，介绍如何进行升级时使用的是 `scrgadm` 命令。管理员并不仅限于使用 `scrgadm` 命令，也可以使用 GUI 或 `scsetup` 命令进行升级。

---

## 资源类型登记文件

### 资源类型名称

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` 的特性。`scrgadm` 命令用来插入句点和冒号分界符以创建资源类型的名称：

```
vendor_id.resource_type:rt_version
```

`Vendor_id` 前缀用来区分不同供应商提供的两个名称相同的登记文件。`RT_version` 用来区分同一基本资源类型的多个登记版本（升级）。为了确保 `Vendor_id` 的唯一性，建议使用创建该资源类型的公司的股票标志。

如果 `RT_version` 字符串包含空格、制表符、正斜杠 (/)、反斜杠 (\)、星号 (\*)、问号 (?)、逗号 (,)、分号 (;)、左方括号 ( [ ) 或右方括号 ( ] ) 字符，资源类型的登记将失败。

从 Sun Cluster 3.1 开始，`RT_version` 特性（在 Sun Cluster 3.0 中为可选特性）已成为必需的特性。

以下命令将返回全限定名称：

```
scha_resource_get -O Type -R resource_name -G resource_group_name
```

在 Sun Cluster 3.1 以前的版本中登记的资源类型名称继续采用以下语法：

```
vendor_id.resource_type
```

## 指令

支持升级的资源类型的 RTR 文件必须包含 `#$upgrade` 指令，后面跟有零或多个采用以下格式的指令：

```
#$upgrade_from version tunability
```

`upgrade_from` 指令由字符串 `#$upgrade_from` 组成，其后跟有 `RT_Version`，再后面跟有对该资源的可调性约束。如果从中执行升级操作的资源类型不具有任何版本，则将 `RT_Version` 指定为空字符串，如下面最后一个示例所示：

```
#$upgrade_from "1.1" when_offline
#$upgrade_from "1.2" when_offline
#$upgrade_from "1.3" when_offline
#$upgrade_from "2.0" when_unmonitored
#$upgrade_from "2.1" anytime
#$upgrade_from "" when_unmanaged
```

当系统管理员尝试更改资源的 `Type version` 时，RGM 将对资源执行这些约束。如果该资源类型的当前版本未出现在列表中，RGM 将强制执行可调性 `When_unmanaged`。

这些指令必须显示在 RTR 文件中的资源类型特性声明部分和 RTR 文件中的资源声明部分之间。请参阅 `rt_reg(4)`。

## 更改 RTR 文件中的 RT\_Version

每当更改 RTR 文件的内容之后，都需要更改 RTR 文件中的 `RT_Version` 字符串。此特性的值必须明确表明哪一个是该资源类型的新版本，哪一个是旧版本。如果未更改 RTR 文件，则无需更改 `RT_Version` 字符串。

## 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：

```
vendor_id.resource_name
```

原来在 Sun Cluster 3.0 中登记的资源类型将继续使用采用此格式的名称（即使是在您将群集软件升级为 Sun Cluster 3.1 或更高版本之后）。同样地，如果 RTR 文件是在运行 Sun Cluster 3.1 或更高版本软件的群集上进行登记的，则其 RTR 文件中缺少 `#$upgrade` 指令的资源类型将被指定为 Sun Cluster 3.0 格式的名称，不包含版本后缀。

在 Sun Cluster 3.0 中，您可以登记包含 `#$upgrade` 或 `#$upgrade_from` 的 RTR 文件，但是不支持将现有资源迁移到新资源类型。

---

## 资源的 `Type_version` 特性

标准资源特性 `Type_version` 用来存储资源类型的 `RT_Version` 特性。此特性未出现在 RTR 文件中。系统管理员可以使用以下命令编辑此特性值：

```
scrgadm -c -j resource -y Type_version=new_version
```

此特性的可调性继承自：

- 资源类型的当前版本
- RTR 文件中的 `#$upgrade_from` 指令

在 `#$upgrade_from` 指令中使用以下可调性值：

### `Anytime`

适用于对于何时升级资源没有限制的情况。该资源可以始终处于联机状态。

### `When_unmonitored`

适用于新资源类型版本的 `Update`、`Stop`、`Monitor_check` 和 `Postnet_stop` 方法与旧资源类型版本的启动方法（`Prenet_stop` 和 `Start`）兼容且新资源类型版本的 `Fini` 方法与旧资源类型版本的 `Init` 方法兼容的情况。此方案仅要求在升级之前停止资源监视器程序

### `When_offline`

如果新资源类型版本的 `Update`、`Stop`、`Monitor_check` 或 `Postnet_stop` 方法与旧资源类型版本的启动方法（`Prenet_stop` 和 `Start`）不兼容，但是与旧版本的 `Init` 方法兼容，则对资源进行类型升级时，该资源必须脱机。

### `When_disabled`

与 `When_offline` 类似。但是，该可调性值会强加更严格的禁用资源的条件。

### `When_unmanaged`

适用于新资源类型版本的 `Fini` 方法与旧版本的 `Init` 方法不兼容的情况。此可调性值要求您必须将现有资源组切换到不受管理的状态才能升级该资源。

### `At_creation`

适用于无法将资源升级到新资源类型版本的情况。仅能创建新版本的新资源。

`At_creation` 可调性表明资源类型开发者可以禁止将现有资源迁移到新类型。在这种情况下，系统管理员必须删除并重新创建该资源。这等同于声明该资源版本仅可以在创建时进行设置。

---

## 将资源迁移到其它版本

在系统管理员编辑资源的 `Type_version` 特性后，现有资源将采用新版本。遵循用来编辑其它资源特性的同一惯例，不同的是要从新资源类型版本中（而不是当前版本）继承或获取一些信息：

- 所有特性的资源特性属性，例如 `min`、`max`、`arraymin`、`arraymax`、缺省值和可调性是从新资源类型版本中获取的。
- 适用于 `Type_version` 特性的可调性是从现有资源的资源类型的 RTR 文件中的 `#$upgrade_from` 指令和 `RT_version` 特性获取的。此可调性与 `property_attributes(5)` 中介绍的可调性不同。
- 将应用新资源类型版本的 `validate` 方法。这可以确保特性属性对新资源类型有效。如果现有资源特性属性不能满足新资源类型版本的验证条件，系统管理员必须在 `scrgadm` 命令行中为这样的特性提供有效值。如果新资源类型版本开始使用未在早期版本中声明且不具有缺省值的特性，可能会发生这样的情况。如果现有资源的某个特性被分配了对新资源类型版本来说无效的值，则也可能发生这种情况。
- 已在旧版本中声明的资源特性在新版本中可能未进行声明。当资源迁移到新版本后，该特性将从该资源中删除。

---

注意 - `validate` 方法可以查询资源的当前 `Type_version`（使用 `scha_resource_get`）和新的 `Type_version`（传递到了 `validate` 命令行）。因此，`validate` 可以取消从不支持的版本进行升级。

---

---

## 升级和降级资源类型

有关升级或迁移资源类型的其它信息，请参阅《*Sun Cluster 数据服务规划和管理指南（适用于 Solaris OS）*》中的“升级资源类型”一节。

### ▼ 如何升级资源类型

1. 请阅读新资源类型的升级文档，找出资源类型更改和资源可调性约束。
2. 在所有群集节点上安装资源类型升级软件包。  
建议按照轮询升级方式安装新的资源类型软件包：在非群集模式下引导节点时，将出现 `pkgadd`。  
以下是在处于群集模式下的节点上安装新资源类型软件包时可能出现的情况：

- 如果安装资源类型软件包时不更改方法代码并且仅更新监视器，则必须在安装过程中停止监视该类型的所有资源。
- 如果安装资源类型软件包时既不更改方法也不更改监视器代码，则不需要在安装过程中停止监视该资源，因为安装操作仅在磁盘上放置一个新的 RTR 文件。

3. 使用 `scrgadm` (或等效) 命令登记新资源类型版本，并引用升级版本的 RTR 文件。

RGM 将创建新资源类型，其名称格式为

```
vendor_id.resource_type:version
```

4. 如果资源类型升级版本仅安装在节点的子集上，则您必须将新资源类型的 `Installed_nodes` 特性设置为实际进行安装的节点。

当资源采用新类型时（无论是通过重新创建还是更新），RGM 要求资源组的 `odelist` 为该资源类型的 `Installed_nodes` 列表的子集。

```
scrgadm -c -t resource_type -h installed_node_list
```

5. 对于预备升级类型（即计划迁移到升级类型的类型）的每一个资源，请调用 `scswitch` 将该资源及其资源组的状态更改为升级文档中指示的相应状态。
6. 对于预备升级类型（即计划迁移到升级类型的类型）的每一个资源，请编辑该资源并将其 `Type_version` 特性更改为新的版本。

```
scrgadm -c -j resource -y Type_version=new_version
```

如果需要，请使用同一命令将同一资源的其它特性编辑为适当的值。

7. 通过使用与步骤 5 中所调用命令相反的命令来恢复资源或资源组原来的状态。

## ▼ 如何将资源降级到其资源类型的早期版本

您可以将资源降级为其资源类型的早期版本。将资源降级为资源类型的早期版本比升级到资源类型的更新版本所需的条件更受限制。首先必须使资源组不受管理。此外，只能将资源降级到资源类型的可升级版本。您可以使用 `scrgadm -p` 命令标识可升级版本。在输出中，可升级版本包含后缀 `:version`。

1. 将包含要降级资源的资源组切换为脱机状态。

```
scswitch -F -g resource_group
```

2. 禁用要降级的资源以及资源组中的所有资源。

```
scswitch -n -j resource_to_downgrade
scswitch -n -j resource1
scswitch -n -j resource2
scswitch -n -j resource3
...
```

---

注意 – 按相关性的顺序禁用资源，从相关性最强的资源（应用程序资源）开始，到相关性最弱的资源（网络地址资源）结束。

---

3. 使资源组不受管理。

```
scswitch -u -g resource_group
```

4. 您要降级到的旧版本的资源类型是否仍注册在群集中？

- 如果是，请转到下一步。
- 如果不是，请重新注册所需的旧版本。

```
scrgadm -a -t resource_type_name
```

5. 通过为 `Type_version` 指定所需的旧版本来降级资源。

```
scrgadm -c -j resource_to_downgrade -y Type_version=old_version
```

如果需要，请使用同一命令将同一资源的其它特性编辑为适当的值。

6. 使包含已降级资源的资源组转为受管理状态，启用所有资源并将组切换为联机状态。

```
scswitch -Z -g resource_group
```

---

## 缺省特性值

RGM 将存储所有资源，这样系统管理员未明确设置的特性（和具有缺省值的特性）将不存储在 CCR（群集配置系统信息库）中的资源项中。如果是从 CCR 读入资源，RGM 将从资源类型中获得缺少的资源特性的缺省值（如果未在该处定义，则使用系统定义的缺省特性值）。正是这个存储特性的方法允许升级后的资源类型定义新特性或为现有特性定义新的缺省值。

编辑资源特性时，RGM 将通过编辑命令指定的特性存储在 CCR 中。

如果资源类型的升级版本声明了缺省特性的新缺省值，则新缺省值被现有资源继承，即使该特性被声明为可变的 `At_creation` 或 `When_disabled`。如果使用新的缺省值导致方法（例如 `Stop`、`Postnet_stop` 或 `Fini`）失败，资源类型实现器必须相应地在升级该资源时限制其状态。限制 `Type_version` 特性的可调性即可实现此目的。

新资源类型版本的 `Validate` 方法可用于进行检查，以确保现有特性属性适用。如果不适用，系统管理员可以通过编辑 `Type_version` 特性使用的同一命令来编辑现有资源的特性，使其具有适当的值，以将该资源升级为新资源类型版本。

---

**注意** – 在将 Sun Cluster 3.0 中创建的资源迁移到更高版本时，它们不会从该资源类型中继承新的缺省特性值，因为它们的缺省特性存储在 CCR 中。

---

---

## 资源类型开发者文档

资源类型开发者必须随新资源一起提供包含以下信息的文档：

- 介绍所有特性的添加、更改或删除操作
- 介绍如何使特性符合新的要求
- 说明对资源的可调性约束
- 说明所有新的缺省特性属性
- 通知系统管理员可以使用编辑 `Type_version` 特性时使用的同一命令来编辑现有资源类型，使其具有适当的值，以将该资源升级到新资源类型版本

---

## 资源类型名称和资源类型监视器实现

虽然您可以在 Sun Cluster 3.0 中登记支持升级的资源类型，但是在 CCR 中记录的资源类型名称不包含版本后缀。要在 Sun Cluster 3.0 和 Sun Cluster 3.1（及更高版本）中正确运行，该资源类型的监视器必须可以处理以下两种命名惯例：

```
vendor_id.resource_name:version
vendor_id.resource_name
```

通过运行与以下代码等效的代码，监视器可以确定要使用的适当名称：

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

然后比较包含 `vers` 的输出值。对于 `vers` 的某一特定值，只有一个命令会成功，因为不可能使用两个不同名称将资源类型的同一版本登记两次。

---

## 应用程序升级

虽然有些方面相似，但是应用程序代码的升级与代理程序代码的升级还是存在不同之处。应用程序升级可能伴有资源类型升级，也可能不伴有资源类型升级。

---

## 资源类型升级实例

这些实例说明了几种不同资源类型的安装和升级方案。已经根据对资源类型实现进行的更改类型选择了可调性和封装信息。可调性适用于将资源迁移到新资源类型的迁移。

所有实例均假设：

- 资源类型来自 Solaris 软件包。请参阅 `pkgadd(1M)` 和 `pkgrm(1M)`。
- 新的 RTR 文件中只有一个 `#$upgrade_from` 指令（由于只有一个以前版本的资源类型）
- 如果从磁盘删除方法时 RGM 要调用这些方法，则安装过程不会删除或覆写这些方法
- 新方法与旧方法兼容（除非另外说明）
- 在安装或迁移之前使用正确的 `scswitch(1M)` 命令或等效命令将资源和资源组转换到所需状态。以下实例说明了如何将资源组转换到不受管理状态：

```
scswitch -M -n -j resource
scswitch -n -j resource
scswitch -F -g resource_group
scswitch -u -g resource_group
```

- 使用此命令登记资源类型：

```
scrgadm -a -t resource_type -f path_to_RTR_file
```

- 使用此命令迁移资源：

```
scrgadm -c -j resource -y Type_version=version \
-y property=value \
-x property=value ...
```

- 迁移之后，将使用相应的 `scswitch(1M)` 命令或等效命令将资源和资源组恢复成以前的状态：

```
scswitch -M -e -j resource
scswitch -e -j resource
scswitch -o -g resource_group
scswitch -Z -g resource_group
```

资源类型开发者可能需要指定比这些实例中所用可调性值的限定性更强的值。可调性值取决于对资源类型实现进行的确切更改。此外，资源类型开发者可以选择使用不同的封装机制来代替这些实例中所用的 Solaris 封装。

表 3-1 升级资源类型的实例

更改的类型	可调性	封装	过程
仅在 RTR 文件中进行特性更改。	Anytime	仅提供新的 RTR 文件。	在所有节点上执行新 RTR 文件的 pkgadd 方法。 登记新资源类型。 迁移资源。
方法已更新。	Anytime	将更新后的方法放置在与旧方法不同的路径下。	对所有节点执行已更新的方法中的 pkgadd。 登记新资源类型。 迁移资源。
更新监视器程序。	When_unmonitored	仅覆写监视器的以前版本。	禁止监视功能。 对所有节点执行新的监视器程序的 pkgadd。 登记新资源类型。 迁移资源。 启用监视功能。
方法将被更新。新 Update/ Stop 方法与旧 Start 方法兼容。	When_offline	将更新后的方法放置在与旧方法不同的路径下。	在所有的节点上执行更新后的方法的 pkgadd。 登记新资源类型。 使资源脱机。 迁移资源。 使资源联机。
方法将被更新且新特性将添加到 RTR 文件中。新方法需要新特性。（为了使包含资源的资源组可以保持联机状态，但又要避免该资源处于联机状态，则该资源组在节点上应该从脱机状态转换为联机状态。）	When_disabled	覆写方法的以前版本。	禁用资源。 对于每个节点： <ul style="list-style-type: none"> <li>■ 使节点脱离群集</li> <li>■ 执行所更新方法中的 pkgrm/pkgadd</li> <li>■ 使节点重新处于群集中</li> </ul> 登记新资源类型。 迁移资源。 启用资源。

表 3-1 升级资源类型的实例 (续)

更改的类型	可调性	封装	过程
方法将被更新且新特性将添加到 RTR 文件中。新方法不需要新特性。	Anytime	覆写方法的以前版本。	<p>对于每个节点：</p> <ul style="list-style-type: none"> <li>■ 使节点脱离群集</li> <li>■ 执行所更新方法中的 pkgrm/pkgadd</li> <li>■ 使节点重新处于群集中</li> </ul> <p>在此过程中，RGM 将调用新的方法，即使还未执行迁移操作（将因此配置新特性）。重要的一点是即使没有新特性，新方法也能够正常工作。</p> <p>登记新资源类型。</p> <p>迁移资源。</p>
方法将被更新。新的 Fini 方法与旧的 Init 方法不兼容。	When_unmanaged	将更新后的方法放置在与旧方法不同的路径下。	<p>使包含资源的资源组处于不受管理状态。</p> <p>对所有节点执行已更新的方法中的 pkgadd。</p> <p>注册资源类型。</p> <p>迁移资源。</p> <p>使包含资源的资源组处于被管理状态。</p>
方法将被更新。不修改 RTR 文件。	不适用。不修改 RTR 文件。	覆写方法的以前版本。	<p>对于每个节点：</p> <ul style="list-style-type: none"> <li>■ 使节点脱离群集</li> <li>■ 执行已更新方法中的 pkgadd</li> <li>■ 使节点重新处于群集中。</li> </ul> <p>因为未修改 RTR 文件，所以无需登记或迁移该资源。</p>

## 资源类型软件包的安装要求

以下是与新资源类型软件包安装相关的两个要求：

- 登记新资源类型时，其 RTR 文件必须可以在磁盘上存取
- 创建新类型的资源时，新资源类型的所有已声明方法路径名和监视器程序必须位于磁盘上并且可执行。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。

为了确定最合适的封装机制，资源类型实现器必须考虑以下几个方面：

- 是否更改 RTR 文件？
- 是否更改特性的缺省值或可调性？
- 是否更改特性的 min 或 max 值？
- 升级过程中是否添加或删除特性？
- 是否更改方法代码？
- 是否更改监视器代码？
- 新的方法或监视器代码与以前版本是否兼容？

## 您在更改 RTR 文件之前需了解的信息

一些资源类型升级并不涉及新的方法或监视器代码。例如，资源类型升级可能仅更改资源特性的缺省值或可调性。既然不会更改方法代码，则对安装升级软件包仅有一项要求，即要具有指向可读 RTR 文件的有效路径名称。

如果无需登记旧的资源类型，新的 RTR 文件可以覆写以前版本。否则，可以将新的 RTR 文件放置在新的路径名下。

如果升级过程中更改了特性的缺省值或可调性，则新版本的 `validate` 方法可以在迁移时检验现有特性属性对新资源类型是否有效。如果升级过程中更改了特性的 `min`、`max` 或 `type` 属性，`scrgadm` 命令将在迁移时自动验证这些约束。

升级文档必须说明所有新的缺省特性属性。该文档必须告知系统管理员使用与编辑 `Type_version` 特性时使用的同一命令编辑各个值，使其具有适当的值，以将资源升级到新的资源类型版本。

如果升级过程中添加或删除了特性，则可能需要更改某些回叫方法或监视器代码。

## 更改监视器代码

如果已更新的资源类型中仅更改了监视器代码，软件包安装可以覆写监视器二进制文件。文档必须通知系统管理员在安装新软件包之前暂停监视。

## 更改方法代码

如果已更新的资源类型中仅更改了方法代码，那么确定新的方法代码是否与以前版本兼容就变得十分重要。这样可以确定是否必须将新方法代码存储在新路径名下或旧的方法是否可以被覆写。

如果新的 `Stop`、`Postnet_stop` 和 `Fini` 方法（如果已声明）可以应用到通过 `Start`、`Preinet_stop` 或 `Init` 方法的旧版本初始化或启动的资源，那么就能够用新方法覆写旧方法。

如果新方法代码与以前版本不兼容，则必须使用该方法的旧版本停止或取消配置资源才能迁移到已升级的资源类型。如果新方法要覆写旧方法，那么在升级资源类型之前，它可能会要求关闭（也许还要取消管理）该类型的所有资源。如果新方法 with 旧方法存储在不同的位置（可以同时存取），则即使没有向下兼容性，也可以安装新的资源类型版本并逐个升级这些资源。

如果新方法可以向下兼容，则可能要求每次将一个资源升级为使用新方法，而其它资源继续使用旧的方法。仍然需要将新方法存储在单独的目录中，而不覆写旧的方法。

将方法的每一个资源类型版本都存储在单独的目录中的优点是当新版本出现问题时可以很容易地转换回旧的资源类型版本。

一种封装方法是包含所有在软件包中仍然支持的早期版本。这样做允许新的软件包版本替换旧的版本，而不覆写或删除旧的方法路径。由资源类型开发者决定可以支持多少个以前版本。

---

**注意** – 建议不要在当前处于群集中的节点上覆写方法或对方法执行 `pkgrm/pkgadd`。如果方法在磁盘上不可存取时，则 `RGM` 调用该方法时可能会导致意外的结果。删除或替换正在运行的方法的二进制文件也可能导致意外的结果。

---



## 第 4 章

# 资源管理 API 参考

---

本章提供组成资源管理 API (RMAPI) 的存取函数和回叫方法的参考信息，其中列出并简单介绍了每一个函数和方法。但是，RMAPI 手册页才是有关这些函数和方法的权威参考。

本章内容包括：

- 第 64 页 “RMAPI 存取方法” – 以 shell 脚本命令和 C 函数的形式
  - `scha_resource_get(1HA)`、`scha_resource_close(3HA)`、`scha_resource_get(3HA)`、`scha_resource_open(3HA)`
  - `scha_resource_setstatus(1HA)`、`scha_resource_setstatus(3HA)`
  - `scha_resourcetype_get(1HA)`、`scha_resourcetype_close(3HA)`、`scha_resourcetype_get(3HA)`、`scha_resourcetype_open(3HA)`
  - `scha_resourcegroup_get(1HA)`、`scha_resourcegroup_get(3HA)`、`scha_resourcegroup_close(3HA)`、`scha_resourcegroup_open(3HA)`
  - `scha_control(1HA)`、`scha_control(3HA)`
  - `scha_cluster_get(1HA)`、`scha_cluster_close(3HA)`、`scha_cluster_get(3HA)`、`scha_cluster_open(3HA)`
  - `scha_cluster_getlogfacility(3HA)`
  - `scha_cluster_getnodename(3HA)`
  - `scha_strerror(3HA)`
- 第 68 页 “RMAPI 回叫方法” – 在 `rt_callbacks(1HA)` 手册页中进行了介绍。
  - `Start`
  - `Stop`
  - `Init`
  - `Fini`
  - `Boot`
  - `Prenet_start`
  - `Postnet_stop`
  - `Monitor_start`

- Monitor\_stop
- Monitor\_check
- Update
- ValidateS

---

## RMAPI 存取方法

API 提供用来存取资源、资源类型和资源组特性以及其它群集信息的函数。这些函数以 shell 命令和 C 函数的形式提供，从而使资源类型提供者可以实现 shell 脚本或 C 程序形式的控制程序。

### RMAPI Shell 命令

shell 命令用于资源类型的回叫方法的 shell 脚本实现，这些资源类型表示群集的 RGM 所控制的服务。您可以使用这些命令执行以下操作：

- 存取关于资源、资源类型、资源组和群集的信息
- 与监视器配套使用以设置资源的 Status 和 Status\_msg 特性
- 请求重新启动或重定位资源组

---

**注意** – 虽然本节中提供了对 shell 命令的简单介绍，但是 1HA 一节中各个手册页提供的信息才是有关 shell 命令的权威参考。每个命令都对应一个具有相同名称的手册页，除非另外注明。

---

### RMAPI 资源命令

您可以使用下面这些命令存取资源的信息或设置资源的 Status 和 Status\_msg 特性。

`scha_resource_get`

存取关于 RGM 所控制的资源和资源类型的信息。它提供的信息与 `scha_resource_get()` 函数提供的相同。

`scha_resource_setstatus`

设置 RGM 所控制的资源的 Status 和 Status\_msg 特性。资源的监视器用它来指示监视器探测到的资源状态。它提供的功能与 `scha_resource_setstatus()` C 函数相同。

---

**注意** – 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

下面的命令用来存取使用 RGM 登记的资源类型的信息。

`scha_resourcetype_get`

此命令提供与 `scha_resourcetype_get()` C 函数相同的功能。

## 资源组命令

您可以使用下面这些命令存取资源组的信息或重新启动资源组。

`scha_resourcegroup_get`

存取关于 RGM 所控制的资源组的信息。此命令提供与 `scha_resourcetype_get()` C 函数相同的功能。

`scha_control`

请求重新启动 RGM 所控制的资源组或将其重定位到其它节点。此命令提供与 `scha_control()` C 函数相同的功能。

## 群集命令

下面的命令用来存取有关群集的信息，例如节点名称、节点 ID、节点状态、群集名称、资源组等。

`scha_cluster_get`

此命令提供与 `scha_cluster_get()` C 函数相同的信息。

## C 函数

C 函数用于资源类型的回叫方法的 C 程序实现，该资源类型表示群集的 RGM 所控制的服务。您可以使用这些函数执行以下操作：

- 存取关于资源、资源类型、资源组和群集的信息
- 与监视器配套使用以设置资源的 `Status` 和 `Status_msg` 特性
- 请求重新启动或重定位资源组
- 将错误代码转换成相应的错误消息

---

**注意** – 虽然本节中提供了对 C 函数的简单说明，但是各个 (3HA) 手册页中提供的信息才是有关 C 函数的权威参考。每个函数都对应一个具有相同名称的手册页，除非另外注明。有关 C 函数的输出变量和返回代码的信息，请参阅 `scha_calls(3HA)` 手册页。

---

## 资源函数

下面这些函数用来存取关于 RGM 所管理的资源的信息或用来指示监视器探测到的资源状态。

`scha_resource_open()`、`scha_resource_get()` 和 `scha_resource_close()`

这些函数共同用于存取有关 RGM 所管理的资源的信息。`scha_resource_open()` 函数用来初始化对资源的存取并返回 `scha_resource_get()` 的句柄，用于存取资源信息。`scha_resource_close()` 函数可使该句柄无效并释放为 `scha_resource_get()` 的返回值分配的内存。

在 `scha_resource_open()` 返回资源的句柄后，通过群集重新配置或管理操作，可以更改资源；在这种情况下，`scha_resource_get()` 通过该句柄获得的信息可能不准确。对资源进行群集重新配置或管理操作时，RGM 将向 `scha_resource_get()` 返回 `scha_err_seqid` 错误代码，以表明该资源的信息可能已更改。这是一个非致命性错误消息；函数将成功返回。您可以选择忽略该消息并接受返回的信息；您也可以关闭当前句柄并打开一个新的句柄，以存取该资源的信息。

在单个手册页中描述了这三个函数。您可以通过以下任意一个函数存取此手册页：`scha_resource_open(3HA)`、`scha_resource_get(3HA)` 或 `scha_resource_close(3HA)`。

`scha_resource_setstatus()`

设置 RGM 所控制的资源的 `Status` 和 `Status_msg` 特性。资源的监视器使用此函数指示该资源的状态。

---

**注意** – 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

下面这些函数共同用于存取通过 RGM 登记的资源类型的信息。

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

`scha_resourcetype_open()` 函数用来初始化对资源类型的存取并返回 `scha_resourcetype_get()` 的句柄，用于存取资源类型的信息。`scha_resourcetype_close()` 函数可使该句柄无效并释放为 `scha_resourcetype_get()` 的返回值分配的内存。

在 `scha_resourcetype_open()` 返回资源类型的句柄后，通过群集重新配置或管理操作，可以更改资源类型；在这种情况下，`scha_resourcetype_get()` 通过该句柄获得的信息可能不准确。对资源类型进行群集重新配置或管理操作时，RGM 将向 `scha_resourcetype_get()` 返回 `scha_err_seqid` 错误代码，以表明该资源类型的信息可能已更改。这是一个非致命性错误消息；函数将成功返回。您可以选择忽略该消息并接受返回的信息；您也可以关闭当前句柄并打开一个新的句柄，以存取关于该资源类型的信息。

在单个手册页中描述了这三个函数。您可以通过以下任意一个函数存取此手册页：`scha_resourcetype_open(3HA)`、`scha_resourcetype_get(3HA)` 或 `scha_resourcetype_close(3HA)`。

## 资源组函数

您可以通过下面这些函数存取资源组信息或重新启动该资源组。

`scha_resourcegroup_open(3HA)`、`scha_resourcegroup_get(3HA)` 和 `scha_resourcegroup_close(3HA)`

这些函数共同用于存取有关 RGM 所管理的资源组的信息。

`scha_resourcegroup_open()` 函数用来初始化对资源组的存取并返回

`scha_resourcegroup_get()` 的句柄，用于存取资源组信息。

`scha_resourcegroup_close()` 函数可使该句柄无效并释放为

`scha_resourcegroup_get()` 的返回值分配的内存。

可以在 `scha_resourcegroup_open()` 返回该资源的句柄后通过群集重新配置或管理操作更改资源类型，这时 `scha_resourcegroup_get()` 通过该句柄获得信息可能会不准确。对资源组进行群集重新配置或管理操作时，RGM 将向

`scha_resourcegroup_get()` 返回 `scha_err_seqid` 错误代码，以表明该资源组的信息可能已更改。这是一个非致命性错误消息；函数将成功返回。您可以选择忽略该消息并接受返回的信息；您也可以关闭当前句柄并打开一个新的句柄，以存取关于该资源组的信息。

`scha_control(3HA)`

请求重新启动 RGM 所控制的资源组或将其重定位到其它节点。

## 群集函数

下面这些新函数用来存取或返回群集的信息。

`scha_cluster_open(3HA)`、`scha_cluster_get(3HA)`、

`scha_cluster_close(3HA)`

这些函数共同用来存取群集信息，例如节点名称、节点 ID 和节点状态、群集名称、资源组等。

可以在 `scha_cluster_open()` 返回该资源的句柄后通过群集重新配置或管理操作更改资源，这时 `scha_cluster_get()` 通过该句柄获得信息可能会不准确。对群集进行重新配置或管理操作时，RGM 将向 `scha_cluster_get()` 返回

`scha_err_seqid` 错误代码，以表明该群集的信息可能已更改。这是一个非致命性错误；函数将成功返回。您可以选择忽略该消息并接受返回的信息；您也可以关闭当前句柄并打开一个新的句柄，以存取关于该群集的信息。

`scha_cluster_getlogfacility(3HA)`  
返回用作群集记录的系统记录工具的数量。使用返回值和 Solaris `syslog()` 函数将事件和状态消息记录到群集记录中。

`scha_cluster_getnodename(3HA)`  
返回在其上调用该函数的群集节点的名称。

## 公用程序函数

此函数用来将错误代码转换成错误消息。

`scha_strerror(3HA)`  
将某个 `scha_` 函数返回的错误代码转换成相应的错误消息。使用此函数和 `logger` 可以将消息记录到系统日志 (`syslog`) 中。

---

## RMAPI 回叫方法

回叫方法是 API 提供的用来实现资源类型的主要元素。当群集成员有所更改时（例如节点引导或崩溃），回叫方法将启用 RGM 来控制该群集中的资源。

---

**注意** – RGM 以引导权限执行该回叫方法，因为客户机程序将控制群集系统上的 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。有时需要为它们指定特权属主，例如 `bin` 或 `root`，请不要使它们可写。

---

本节介绍回叫方法变量和出口代码，还按以下分类列出并介绍了各个回叫方法：

- 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

---

**注意** – 虽然本节提供了回叫方法的简要说明（包括何时调用方法以及对资源的预期影响），但是 `rt_callbacks(1HA)` 手册页才是有关回叫方法的权威参考。

---

## 方法变量

RGM 按照以下方式调用回叫方法：

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

其中 `method` 是登记为 `Start`、`Stop` 或其它回叫方法的程序的路径名。资源类型的回叫方法在其登记文件中进行声明。

所有回叫方法变量都作为标记值传送，`-R` 用来表示资源实例的名称，`-T` 用来表示资源的类型，`-G` 用来表示配置该资源的组。使用变量和存取函数来检索关于资源的信息。

`Validate` 方法通过附加变量（所调用资源和资源组的特性值）调用。

有关详细信息，请参阅 `scha_calls(3HA)`。

## 出口代码

所有回叫方法都定义了相同的出口代码，以指定方法调用对资源状态的影响。`scha_calls(3HA)` 手册页中介绍了所有这些出口代码。出口代码为：

- 0 — 方法成功
- 任意非零值 — 方法失败

RGM 还可处理回叫方法执行过程中的不正常失败，例如超时和信息转储。

方法实现必须使用每个节点上的 `syslog` 输出失败信息。并不能保证写入 `stdout` 或 `stderr` 的输出信息一定会传送到用户（尽管它当前显示在本地节点的控制台中）。

## 控制和初始化回叫方法

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

### Start

当包含该资源的资源组在某个群集节点联机时，会对此节点调用此必需方法。此方法激活该节点上的资源。

在所激活的资源启动并且在本地节点上可用之前，`Start` 方法不应退出。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。此外，您应该为此方法设置足够长的超时值。例如，某些资源（例如数据库守护程序）需要较长的时间进行启动，因此，就要求该方法具有较长的超时值。

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

资源类型登记文件中的 `START_TIMEOUT` 特性用来设置资源的 `Start` 方法的超时值。

### Stop

当包含该资源的资源组在群集节点上脱机时，将对群集节点调用此必需方法。如果该资源处于激活状态，则此方法可取消激活该资源。

在所控制资源在本地节点上完全停止所有活动并关闭所有文件描述符之前，Stop 方法不应退出，否则，因为 RGM 将认为该资源已经停止，而实际上它仍处于活动状态，就会导致数据被破坏。避免数据被破坏的最可靠方法是终止与该资源有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。此外，您应该为此方法设置足够长的超时值。例如，某些资源（如数据库守护程序）需要较长的时间进行停止操作，因此，就要求该方法具有较长的超时值。

RGM 对 Stop 方法失败的响应方式取决于 Failover\_mode 特性的设置（请参阅表 A-2）。

资源类型登记文件中的 STOP\_TIMEOUT 特性用来设置资源的 Stop 方法的超时值。

### Init

当资源变为被管理的资源时（资源所在的资源组从未被管理状态切换到已管理状态或资源创建在已被管理的资源组中），调用此可选方法以对资源执行一次性初始化。对哪些节点调用该方法由 Init\_nodes 资源特性决定。

### Fini

当资源变为未被管理的资源时（资源所在的资源组切换到未被管理状态或从被管理的资源组中删除了该资源），调用此可选方法以清除资源。对哪些节点调用该方法由 Init\_nodes 资源特性决定。

### Fini

此可选方法与 Init 相似，在包含资源的资源组处于 RGM 的管理之下后调用此方法可初始化与群集相连接的节点上的资源。对哪些节点调用该方法由 Init\_nodes 资源特性决定。引导或重新引导操作导致该节点连接或重新连接群集时，将调用 Boot 方法。

---

**注意** - Init、Fini 或 Boot 方法的失败将导致 syslog() 函数生成错误消息，但是不会影响该资源的 RGM 管理。

---

## 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特性。Validate 和 Update 回叫方法将启用资源类型实现来介入这些管理操作。

### Validate

当创建了资源且管理操作更新资源或其资源组的特性时，调用此可选方法。将对由资源类型的 Init\_nodes 特性指定的一组群集节点调用此方法。在进行创建或更新之前，将调用 Validate，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。

仅当通过管理操作更改资源或资源组特性时（而不是在 RGM 设置特性时或监视器设置资源组特性 `Status` 和 `Status_msg` 时），才调用 `Validate`。

#### Update

调用此可选方法将特性已更改的消息通知正在运行的资源。在管理操作成功地设置了资源及其资源组后，将调用 `Update`。对资源处于联机状态的节点调用此方法。该方法将通过 API 存取函数来读取可能会影响活动资源的特性值并相应地调整正在运行的资源。

`Update` 方法的失败将导致 `syslog()` 函数生成错误消息，但是不会影响该资源的 RGM 管理。

## 与网络相关的回叫方法

使用网络地址资源的服务可能要求按某种顺序（相对于网络地址配置）执行启动或停止步骤。以下可选回叫方法 `Prenet_start` 和 `Postnet_stop` 使资源类型实现在配置/取消配置相关网络地址之前/之后执行特殊的启动和关闭操作。

#### `Prenet_start`

在配置同一资源组中的网络地址之前，将调用此可选方法执行特殊的启动操作。

#### `Postnet_stop`

在同一资源组中的网络地址配置为停止使用之后，调用此可选的方法执行特殊停机操作。

## 监视器控制回叫方法

资源类型实现中可包含一个用来监视资源性能、报告其状态或针对资源失败情况进行操作的程序（可选）。在资源类型实现中，`Monitor_start`、`Monitor_stop` 和 `Monitor_check` 方法支持资源监视器的实现。

#### `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`

在资源组重新定位到节点之前，将调用此可选方法来评估该节点的可靠性。必须实现 `Monitor_check` 方法，以免与当前运行的其它方法发生冲突。



## 第 5 章

---

# 数据服务样例

---

本章介绍了 `in.named` 应用程序的 Sun Cluster 数据服务 HA-DNS 的样例。`in.named` 守护程序是域名服务 (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。

本章内容包括：

- 第 73 页 “数据服务样例概述”
- 第 74 页 “定义资源类型登记文件”
- 第 79 页 “为所有方法提供通用功能”
- 第 83 页 “控制数据服务”
- 第 88 页 “定义故障监视器”
- 第 96 页 “处理特性更新”

---

## 数据服务样例概述

数据服务样例在群集的各个节点间启动、停止、重启和切换 DNS 应用程序，以响应群集事件（例如管理操作、应用程序故障或节点故障）。

应用程序的重启操作由进程监视工具 (PMF) 管理。如果应用程序故障次数超出了故障时间窗口中的故障计数，故障监视器将把含有该应用程序资源的资源组故障切换到其它节点。

数据服务样例通过 PROBE 方法提供故障监视，该方法使用 `nslookup` 命令确保应用程序正常运行。如果探测程序检测到挂起的 DNS 服务，它将尝试通过在本地重启 DNS 应用程序的方法来更正这种情况。如果没有改善情况并且探测程序多次检测到该服务的问题，则探测程序试图对该服务进行故障切换，切换到群集中的另一个节点。

具体地说，数据服务样例包括：

- 用于定义数据服务的静态特性的资源类型登记文件。
- 使含有 HA-DNS 数据服务的资源组联机时，由 RGM 调用的用来启动 in.named 守护程序的 Start 回叫方法。
- 使含有 HA-DNS 的资源组脱机时，由 RGM 调用的用来停止 in.named 守护程序的 Stop 回叫方法。
- 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。该故障监视器由用户定义的 PROBE 方法实现，并且分别通过 Monitor\_start 和 Monitor\_stop 回叫方法进行启动和停止。
- 由 RGM 调用的用来检验服务的配置目录是否可以存取的 validate 回叫方法。
- 系统管理员更改资源特性的值时，由 RGM 调用的用来重启故障监视器的 Update 回叫方法。

---

## 定义资源类型登记文件

本实例中的资源类型登记 (RTR) 文件中定义了 DNS 资源类型的静态配置。此类型的资源继承了在 RTR 文件中定义的特性。

群集管理员登记 HA-DNS 数据服务时，由 RGM 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特性、系统定义的资源特性和扩展特性。有关详细信息，请参阅 `rt_reg(4)` 手册页和第 29 页“设置资源和资源类型特性”。

本小节介绍了 RTR 文件样例中的具体特性。其中列出了该文件的不同部分。要想获得 RTR 文件样例的完整内容列表，请参阅第 221 页“资源类型登记文件列表”。

### RTR 文件样例中的资源类型特性

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

```
#
# Copyright (c) 1998-2004 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` 特性声明为 RTR 文件中的第一项。否则，资源类型的登记将失败。

---



---

**注意** – RGM 认为特性名是不区分大小写的。由 Sun 提供的 RTR 文件中的特性名称惯例是将名称的首字母大写，其余字母小写（方法名除外）。方法名以及特性的属性中包含的都是大写字母。

---

以下是有关这些特性的一些信息。

- 您可以仅使用 `Resource_type` 特性指定资源类型名称 (`sample`)；或在该特性前用 `Vendor_id` 作为前缀，前缀与资源类型之间用“.”分隔 (`SUNW.sample`)。如果使用 `Vendor_id`，请将其设置为用来定义资源类型的公司的股票代码。在群集中资源类型的名称必须唯一。
- `Rt_version` 特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` 特性用于标识 Sun Cluster 的版本。例如，`API_version = 2` 表明数据服务在 Sun Cluster 3.0 版本中运行。
- `Failover = TRUE` 表明数据服务无法在可同时在多个节点联机的资源组中运行。
- `RT_basedir` 指向 `/opt/SUNWsample/bin`，作为指向完整相对路径（例如回叫方法路径）的目录路径。
- `Start`、`Stop`、`Validate` 等提供了 RGM 调用的相应回叫方法程序的路径。这些路径是基于 `RT_basedir` 所指定的目录的相对路径。

- Pkglist 用于将 SUNWsample 标识为包含数据服务样例安装的软件包。

未在此 RTR 文件中指定的资源类型特性（例如 `Single_instance`、`Init_nodes` 和 `Installed_nodes`）将使用缺省值。要获得这些资源类型特性及其缺省值的完整列表，请参阅表 A-1。

群集管理员无法更改 RTR 文件中所指定的资源类型特性的值。

## RTR 文件样例中的资源特性

按照惯例，在 RTR 文件中声明资源类型特性之后声明资源特性。资源特性包括由 Sun Cluster 提供的系统定义的特性和由您定义的扩展特性。对于每一种类型，您都可以指定多个由 Sun Cluster 提供的特性属性，例如最小值、最大值和缺省值。

## RTR 文件中的系统定义的特性

下面列出了 RTR 文件样例中的系统定义的特性。

```
# 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 = "";
}

```

虽然 Sun Cluster 提供了系统定义的特性，但是您也可以使用资源特性属性设置不同的缺省值。要获得可应用于资源特性的属性的完整列表，请参阅第 219 页“资源特性属性”。

请注意以下有关 RTR 文件样例中的系统定义的资源特性的信息：

- 对于所有超时设置，Sun Cluster 都提供了最小值（1 秒）和缺省值（3600 秒）。在 RTR 文件样例中将最小值更改为 60 秒，并将缺省值更改为 300 秒。群集管理员既可以接受此缺省值，又可以将超时值更改为其它值（60 或更大值）。Sun Cluster 不具有最大允许值。
- 特性 Thorough\_Probe\_Interval、Retry\_count 和 Retry\_interval 的 TUNABLE 属性的值已设置成 ANYTIME。此设置意味着即使在数据服务运行的情况下，群集管理员也可以更改这些特性的值。这些特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特性后停止和重启故障监视器。请参阅第 100 页“Update 方法”。
- 资源特性分为以下几类：
  - **必需的** — 创建资源时，群集管理员必须指定一个值；
  - **可选的** — 如果管理员未指定值，系统将应用缺省值。
  - **有条件的** — 仅当在 RTR 文件中声明之后，RGM 才会创建该特性。

数据服务样例的缺省监视器使用 Thorough\_probe\_interval、Retry\_count、Retry\_interval 和 Network\_resources\_used 条件特性，因此开发者需要在 RTR 文件中声明它们。有关特性如何分类的信息，请参阅 r\_properties(5) 手册页或第 210 页“资源特性”。

## RTR 文件中的扩展特性

位于 RTR 文件样例结尾处的是扩展特性，如下所示：

```
# 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)";
}
```

RTR 文件样例中定义了两种扩展特性 `Confdir` 和 `Probe_timeout`。`Confdir` 用于指定指向 DNS 配置目录的路径。此目录中包含 DNS 要成功地进行操作所需的 `in.named` 文件。在启动 DNS 之前，数据服务样例的 `Start` 和 `Validate` 方法将使用此特性来检验配置目录和 `in.named` 文件是否可以存取。

配置数据服务之后，`Validate` 方法将检验新目录是否可以存取。

数据服务样例的 `PROBE` 方法不是 Sun Cluster 回叫方法，而是用户定义的方法，因此，Sun Cluster 未为该方法提供 `Probe_timeout` 特性。开发者已在 RTR 文件中定义了一个扩展特性，以便群集管理员配置 `Probe_timeout` 值。

---

## 为所有方法提供通用功能

本小节介绍以下适用于数据服务样例中所有回叫方法的功能：

- 第 79 页“标识命令解释程序并输出路径”。
- 第 79 页“声明 `PMF_TAG` 和 `SYSLOG_TAG` 变量”。
- 第 80 页“分析函数变量”。
- 第 82 页“生成错误消息”。
- 第 82 页“获取特性信息”。

### 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。数据服务样例中每一个方法脚本都用来标识命令解释程序，如下所示：

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

样例应用程序中的所有方法脚本输出指向 Sun Cluster 二进制文件和库的路径，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

所有的方法脚本（`validate` 除外）使用 `pmfadm` 启动（或停止）数据服务或监视器，并传送资源的名称。每个脚本中都定义了变量 `PMF_TAG`，该变量可以传送到 `pmfadm` 来标识数据服务或监视器。

同样，每个方法脚本都使用 `logger` 命令通过系统日志记录消息。每个脚本中都定义了变量 `SYSLOG_TAG`，该变量可以通过 `-t` 选项传送到 `logger`，以标识要记录其消息的资源类型、资源组和资源名称。

所有的方法都用相同的方式定义了 `SYSLOG_TAG`，如下样例所示。`dns_probe`、`dns_svc_start`、`dns_svc_stop` 和 `dns_monitor_check` 方法定义的 `PMF_TAG` 如下所示：（使用的是 `dns_svc_stop` 方法中的 `pmfadm` 和 `logger`）：

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

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCE_GROUP_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 ${SYSLOG_FACILITY}.info \
        -t [SYSLOG_TAG] \
        "${ARGVO} Failed to stop HA-DNS with SIGTERM; Retry with \
        SIGKILL"
```

`dns_monitor_start`、`dns_monitor_stop` 和 `dns_update` 方法定义的 `PMF_TAG` 如下所示（使用的是 `dns_monitor_stop` 方法中的 `pmfadm`）：

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

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCE_GROUP_NAME,$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
```

## 分析函数变量

RGM 将调用所有回叫方法（`validate` 除外），如下所示。

```
method_name -R resource_name -T resource_type_name -G resource_group_name
```

方法名是实现回叫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。这些路径名与由 `Rt_basedir` 特性（也在 RTR 文件中）指定的目录相关。例如，在数据服务样例的 RTR 文件中，基目录和方法名按照以下方式指定。

```
RT_BASEDIR=/opt/SUNWsample/bin;
START = dns_svc_start;
```

```
STOP = dns_svc_stop;
...
```

所有回叫方法变量都作为标记值传送，-R 用来表示资源实例的名称，-T 用来表示资源的类型，-G 用来表示配置该资源的组。有关回叫方法的详细信息，请参阅 `rt_callbacks (1HA)` 手册页。

---

**注意** -Validate 方法是使用其它变量（调用该方法的资源和资源组的特性值）调用的。有关详细信息，请参阅第 96 页“处理特性更新”。

---

每个回叫方法都需要一个函数来分析其传送的变量。因为所有回叫方法都传送相同的变量，所以数据服务提供了一个用于应用程序中所有回叫方法的分析函数。

下面显示了用于应用程序样例中的回叫方法的 `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 ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCE_NAME,$RESOURCEGROUP_NAME,$RESOURCETYPE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
    done
}
```

---

**注意** – 虽然应用程序样例中的 `PROBE` 方法是用户定义的（不是 Sun Cluster 回叫方法），但是也使用与调用回叫方法相同的变量来调用该方法。因此，此方法包含的分析函数与其它回叫方法所使用的分析函数相同。

---

分析函数在 `MAIN` 中调用，如下所示：

```
parse_args "$@"
```

## 生成错误消息

建议回叫方法使用 `syslog` 工具向最终用户输出错误消息。数据服务样例中的所有回叫方法都使用 `scha_cluster_get()` 函数检索用于群集日志的 `syslog` 工具的编号，如下所示：

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

该值存储在 `shell` 变量 `SYSLOG_FACILITY` 中，该工具可用作 `logger` 命令的工具，以便在群集日志中记录消息。例如，数据服务样例中的 `Start` 方法将检索 `syslog` 工具并记录一条说明该数据服务已启动的消息，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
```

有关详细信息，请参阅 `scha_cluster_get(1HA)` 手册页。

## 获取特性信息

多数回叫方法都需要获取有关数据服务的资源特性和资源类型特性的信息。为实现此操作，API 提供了 `scha_resource_get()` 函数。

可以获取系统定义的特性和扩展特性这两种资源特性。已经预先定义了系统定义的特性，然而您需要在 `RTR` 文件中定义扩展特性。

当您使用 `scha_resource_get()` 获取系统定义的特性的值时，需要使用 `-O` 参数指定该特性的名称。命令仅返回该特性的值。例如，在数据服务样例中，`Monitor_start` 方法需要定位探测程序，以便可以启动该程序。该探测程序驻留在数据服务的基目录中，通过 `RT_BASEDIR` 特性指向该程序，因此 `Monitor_start` 方法将检索 `RT_BASEDIR` 的值，并将其放置在 `RT_BASEDIR` 变量中，如下所示。

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

对于扩展特性，您必须用 `-O` 参数指定特性为扩展特性，并提供该特性的名称作为最后一个参数，命令返回特性的**类型**和**值**。例如在样例数据服务中，探测程序检索 `probe_timeout` 扩展特性的类型和值，然后使用 `awk` 将该值仅放置在 `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` 方法来激活群集上的应用程序守护程序；还必须提供 `Stop` 或 `Postnet_stop` 方法来停止群集上的应用程序守护程序。数据服务样例中实现了 `Start` 和 `Stop` 方法。有关何时应使用 `Prenet_start` 和 `Postnet_stop` 方法的信息，请参阅第 38 页“决定使用哪种 `Start` 或 `Stop` 方法”。

### Start 方法

当使包含数据服务资源的资源组在群集节点上联机时，或当该资源组已联机且该资源已启动时，RGM 将对群集节点调用 `Start` 方法。在样例应用程序中，`Start` 方法在该节点激活 `in.named` (DNS) 守护程序。

本小节介绍了应用程序样例中的 `Start` 方法的重要方面，但未介绍所有回叫方法都通用的功能，例如 `parse_args()` 函数和获取 `syslog` 工具，这些在第 79 页“为所有方法提供通用功能”中介绍。

有关 `Start` 方法的完整列表，请参阅第 224 页“`Start` 方法”。

### Start 概述

在尝试启动 DNS 之前，数据服务样例中的 `Start` 方法将检验配置目录和配置文件 (`named.conf`) 是否可以存取并可用。`named.conf` 中的信息对于能否成功地进行 DNS 操作来说至关重要。

此回叫方法使用进程监视器工具 (`pmfadm`) 启动 DNS 守护程序 (`in.named`)。如果 DNS 崩溃或无法启动，PMF 将按指定的次数在指定的时间间隔内尝试启动该 DNS。重试的次数和间隔由数据服务的 RTR 文件中的特性指定。

### 检验配置

要进行操作，DNS 需要配置目录中 `named.conf` 文件的信息。因此，在尝试启动 DNS 之前，`Start` 方法将执行一些合理性检查，以检验该目录和文件是否可以存取。

Confdir 扩展特性提供了指向配置目录的路径。特性本身是在 RTR 文件中定义的，但是其实际位置却是群集管理员在配置数据服务时指定的。

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

---

**注意** – 因为 Confdir 是扩展特性，所以 `scha_resource_get()` 将同时返回类型和值。awk 命令仅对值进行检索并将其放置在 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}'`
```

然后，Start 方法将使用 `CONFIG_DIR` 的值检验目录是否可以存取。如果目录不可以存取，Start 将记录一条错误消息并在错误状态下退出。请参阅第 85 页“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程序之前，此方法将执行 **final** 检查，以检验 `named.conf` 文件是否存在。如果不存在，Start 将记录一条错误消息并在错误状态下退出。

```
# 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

此方法使用进程管理器工具 (pmfadm) 来启动应用程序。pmfadm 允许您设置在指定时间内重启应用程序的次数。RTR 文件中包含两个特性 `Retry_count` 和 `Retry_interval`，分别用来指定尝试重启应用程序的次数和两次重启操作之间的时间间隔。

`Start` 方法将使用 `scha_resource_get()` 函数检索 `Retry_count` 和 `Retry_interval` 的值，并将这些值存储在 shell 变量中。然后使用 `-n` 和 `-t` 选项将这些值传送到 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

在基础应用程序确实正在运行并可用之前（尤其是其它数据服务依赖于该程序的情况下），`Start` 方法不应在成功状态下退出。一种用来检验是否成功的方法是：在退出 `Start` 方法之前探测该应用程序以检验该程序是否正在运行。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` 特性值，使应用程序有足够长的时间进行初始化和执行崩溃恢复操作。

---

**注意** – 因为数据服务样例中的应用程序资源 `DNS` 将迅速启动，所以在成功状态下退出之前，数据服务样例不会进行轮询来检验该应用程序资源是否正在运行。

---



```
-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 守护程序是否正在运行。如果正在运行，Stop 首先将使用 `pmfadm -s` 发送 TERM 信号，以终止 DNS 进程。如果此信号无法在超时值的 80% 时间内终止该进程，Stop 将发送 SIGKILL 信号。如果此信号也无法在超时值的 15% 时间内终止该进程，该方法将记录一条错误消息并在错误状态下退出。

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。以下代码样例说明了 Stop 是如何使用 `pmfadm` 停止 DNS 进程的。

```
# 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 ${SYSLOG_FACILITY}.err \
            -t [$RESOURCE_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 ${SYSLOG_FACILITY}.err \  
-t [${RESOURCE_TYPE_NAME},${RESOURCE_GROUP_NAME},${RESOURCE_NAME}] \  
\  
"HA-DNS successfully stopped" \  
exit 0
```

## Stop 退出状态

在基础应用程序确实停止之前（尤其是在其它数据服务依赖于该程序的情况下），Stop 方法不应在成功状态下退出。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 特性值，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 特性。数据服务样例未明确设置 Failover\_mode 特性，因此此特性使用缺省值 NONE（除非群集管理员已经覆盖了该缺省值并指定了其它值）。在这种情况下，RGM 仅执行将数据服务状态设置为 Stop\_failed 的操作。需要用户干预以强制停止应用程序并清除 Stop\_failed 状态。

---

## 定义故障监视器

应用程序样例实现一个基本的故障监视器，以监视 DNS 资源 (in.named) 的可靠性。故障监视器包括：

- dns\_probe，这是一个使用 nslookup 检验数据服务样例所控制的 DNS 资源是否正在运行的用户定义的程序。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，这是一个用来启动 dns\_probe 的回叫方法。如果启用了监视操作，则在使数据服务样例联机后，RGM 将自动调用 dns\_monitor\_start。
- dns\_monitor\_stop，这是一个用来停止 dns\_probe 的回叫方法。使数据服务样例脱机之前，RGM 将自动调用 dns\_monitor\_stop。
- dns\_monitor\_check，一种调用 validate 方法的回叫方法，检验配置目录在 PROBE 程序将数据服务故障切换到新节点时是否可用。

## 探测程序

dns\_probe 程序将实现一个持续运行的进程，用来检验数据服务样例所控制的 DNS 资源是否正在运行。dns\_probe 由 dns\_monitor\_start 方法启动，使数据服务样例联机后，RGM 将自动调用该方法。该数据服务由 dns\_monitor\_stop 方法停止，使数据服务样例脱机前，RGM 将调用该方法。

本小节介绍了应用程序样例中的 PROBE 方法的重要方面，但未介绍所有回叫方法都通用的功能，例如 `parse_args()` 函数和获取 `syslog` 工具，这些在第 79 页“为所有方法提供通用功能”中介绍。

有关 PROBE 方法的完整列表，请参阅第 230 页“PROBE 程序”。

## 探测程序概述

探测程序以死循环的形式运行。它使用 `nslookup` 检验适当的 DNS 资源是否正在运行。如果 DNS 正在运行，探测程序将按照指定的时间间隔（由系统定义的特性 `Thorough_probe_interval` 进行设置）进行休眠，然后再次进行检查。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。

## 获取特性值

此程序需要以下特性值：

- `Thorough_probe_interval` — 用来设置探测程序休眠的时间段
- `Probe_timeout` — 用来对进行探测操作的 `nslookup` 命令强制执行探测程序的超时值
- `Network_resources_used` — 用来获取 DNS 服务器的 IP 地址
- `Retry_count` 和 `Retry_interval` — 用来确定重启尝试的次数和进行计数的时间间隔
- `Rt_basedir` — 用来获取包含 PROBE 程序和 `gettime` 公用程序的目录

`scha_resource_get()` 函数可用来获取这些特性的值，并将其存储在 `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'
```

---

注意 - 对于系统定义的特性 (例如 `Thorough_probe_interval`) ,  
`scha_resource_get()` 仅返回值。对于扩展特性 (例如 `Probe_timeout`) ,  
`scha_resource_get()` 将返回类型和值。使用 `awk` 命令仅能获取值。

---

## 检查服务的可靠性

探测程序本身是 `nslookup` 命令的 `while` 死循环。在进行 `while` 循环之前, 将设置一个临时文件, 以保留 `nslookup` 应答。 `probefail` 和 `retries` 变量初始化为 0。

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

`while` 循环本身将:

- 设置探测程序的休眠间隔
- 使用 `hatimerun` 启动 `nslookup`, 传送 `Probe_timeout` 的值并标识目标主机
- 根据 `nslookup` 返回代码的成功或失败状态, 设置 `probefail` 变量
- 如果将 `probefail` 设置为 1 (失败), 则可检验对 `nslookup` 的应答是由数据服务样例发出的还是由其它 DNS 服务器发出的

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
fi

```

## 评估重启和故障切换

如果 *probfail* 变量是 0（成功）之外的值，这意味着 `nslookup` 命令超时或表明应答是某个服务器发出的，而不是服务样例的 DNS 发出的。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 *probfail* 变量是 0，则会生成一条表明探测程序成功的消息。

```

if [ $probfail -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`) 来确定是在本地重启 DNS，还是请求 RGM 将数据服务重定位到其它节点。它实现了以下有条件的代码（请参阅第 230 页“PROBE 程序”中的 `decide_restart_or_failover()` 的代码列表）。

- 如果这是第一次出现故障，重新启动数据服务。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。
- 如果时间仍处于窗口的范围内，但已超出重试计数器的计数范围，请故障切换到另一个节点。如果故障切换未成功，将记录一条错误消息并以状态 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。记录一条错误消息并取消 `retries` 变量中的计数器。

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` 公用程序用来跟踪两次重启操作之间的时间。这是驻留在 (`Rt_basedir`) 目录中的 C 程序。

- 系统定义的资源特性 `Retry_count` 和 `Retry_interval` 可确定重启尝试的次数以及进行计数的时间间隔。在 `RTR` 文件中这些特性缺省设置为在 5 分钟（300 秒）时间内尝试 2 次，但是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启该数据服务。有关此函数的信息，请参阅下一节第 92 页“重启数据服务”。
- 使用 `scha_control()` API 函数和 `GIVEOVER` 选项可以使含有数据服务样例的资源组脱机并在其它节点上重新联机。

## 重启数据服务

`decide_restart_or_failover()` 调用 `restart_service()` 函数以试图在同一个节点上重新启动数据服务。该函数执行以下操作。

- 它可确定数据服务是否仍然在 PMF 的控制下进行登记。如果仍然对该服务进行登记，则该函数将：
  - 获取数据服务的 `Stop` 方法名和 `Stop_timeout` 值。
  - 使用 `hatimerun` 启动数据服务的 `Stop` 方法，并传送 `Stop_timeout` 值。
  - （如果成功停止数据服务）获取数据服务的 `Start` 方法名称和 `Start_timeout` 值。
  - 使用 `hatimerun` 启动数据服务的 `Start` 方法，并传送 `Start_timeout` 值。
- 如果该数据服务不再在 PMF 的控制下进行登记，则表明该数据服务已超出了在 PMF 控制下进行重试的最大允许次数，因此可以使用 `GIVEOVER` 选项调用 `scha_control()` 函数将数据服务故障切换到其它节点。

```
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_NAME?
        STOP_METHOD=`scha_resource_get -O STOP \
            -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME?
        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
    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_NAME?
START_METHOD=`scha_resource_get -O START \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME?
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

如果尝试在本地进行重启的操作以及尝试故障切换到其它节点的操作都失败后，数据服务样例的 PROBE 程序将在失败状态下退出。它将记录错误消息“故障切换尝试失败”。

## Monitor\_start 方法

在使数据服务样例联机之后，RGM 将调用 Monitor\_start 方法来启动 dns\_probe 方法。

本小节介绍了应用程序样例中的 Monitor\_start 方法的重要方面，但未介绍所有回叫方法都通用的功能，例如 parse\_args() 函数和获取 syslog 工具，这些在第 79 页“为所有方法提供通用功能”中介绍。

要获得 Monitor\_start 方法的完整列表，请参阅第 235 页“Monitor\_start 方法”。

## Monitor\_start 概述

此方法使用进程监视工具 (pmfadm) 启动探测程序。

## 启动探测程序

Monitor\_start 方法获取 Rt\_basedir 特性的值以构造 PROBE 程序的完整路径名。此方法将使用 pmfadm (-n -1, -t -1) 的无限重试选项启动探测程序，这意味着如果该探测程序无法启动，PMF 将在无限长的时间间隔内尝试无限多次地启动该程序。

```
# 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

使数据服务样例脱机后，RGM 将调用 Monitor\_stop 方法停止 dns\_probe 的执行。

本小节介绍了应用程序样例中的 Monitor\_stop 方法的重要方面，但未介绍所有回叫方法都通用的功能，例如 parse\_args() 函数和获取 syslog 工具，这些在第 79 页“为所有方法提供通用功能”中介绍。

要获得 Monitor\_stop 方法的完整列表，请参阅第 237 页“Monitor\_stop 方法”。

## Monitor\_stop 概述

此方法将使用进程监视工具 (pmfadm) 查看探测程序是否正在运行，如果正在运行，就停止该程序。

## 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。

```
# 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
exit 0

```




---

**注意** – 请务必通过 `pmfadm` 使用 `KILL` 信号停止该探测程序，而不要使用可屏蔽信号，例如 `TERM`。否则，`Monitor_stop` 方法将无限期地挂起，最终将会超时。导致此问题的原因是必要时 `PROBE` 方法会调用 `scha_control()`，以重启或故障切换数据服务。作为使数据服务脱机进程的一部分，当 `scha_control()` 调用 `Monitor_stop` 时，如果 `Monitor_stop` 使用可屏蔽信号，它将挂起，等待 `scha_control()` 的完成，且 `scha_control()` 也将挂起，等待 `Monitor_stop` 的完成。

---

## 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 将使数据服务样例在主节点上处于 `MONITOR_FAILED` 状态，此操作可能会影响该主节点。

在探测程序停止之前，`Monitor_stop` 不应退出。

## Monitor\_check 方法

每当 `PROBE` 方法视图将包含数据服务的资源组故障切换到新节点上时，RGM 都将调用 `Monitor_check` 方法。

本小节介绍了应用程序样例中的 `Monitor_check` 方法的重要方面，但未介绍所有回叫方法都通用的功能，例如 `parse_args()` 函数和获取 `syslog` 工具，这些在第 79 页“为所有方法提供通用功能”中介绍。

要获得 `Monitor_check` 方法的完整列表，请参阅第 238 页“`Monitor_check` 方法”。

必须实现 `Monitor_check` 方法，以免与当前运行的其它方法发生冲突。

`Monitor_check` 方法调用 `Validate` 方法检验 DNS 配置目录在新节点上是否可用。`Confdir` 扩展特性指向 DNS 配置目录。因此，`Monitor_check` 将获取 `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_NAME?

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

# 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 $RESOURCECETYPE_NAME -x Confdir=$CONFIG_DIR

```

要了解应用程序样例如何检验一个节点是否适合作数据服务主机，请参阅第 96 页“Validate 方法”。

---

## 处理特性更新

数据服务样例实现 `validate` 和 `update` 方法，以处理群集管理员对特性进行的更新。

### Validate 方法

创建资源时以及通过管理操作更新资源或其资源组的特性时，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。

仅当通过管理操作更改资源或组特性时（而不是在 RGM 设置特性时或监视器设置资源特性 `Status` 和 `Status_msg` 时），RGM 才调用 `validate`。

---

**注意** – 当 `PROBE` 方法试图将数据服务故障切换到新节点时，`Monitor_check` 方法还显式调用 `validate` 方法。

---

## Validate 概述

RGM 通过不同于传送到其它方法的变量的附加变量（包括所更新的特性和值）调用 Validate。因此，在数据服务样例中此方法必须实现一个不同的 `parse_args()` 函数以处理附加变量。

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

---

**注意** – 因为在 DNS 运行时无法更改配置目录，RTR 文件中 `Confdir` 特性被声明为 `TUNABLE = AT_CREATION`。因此，进行更新后将永远不会调用 Validate 方法来检验 `Confdir` 特性，仅当创建数据服务资源时才会调用。

---

如果 `Confdir` 是 RGM 传送到 Validate 的特性之一，则 `parse_args()` 函数将进行检索并保存其值。然后 Validate 将检验通过 `Confdir` 的新值所指向的目录是否可以存取，并检验在该目录中 `named.conf` 文件是否存在并包含有数据。

如果 `parse_args()` 函数无法从 RGM 传送的命令行变量中检索到 `Confdir` 的值，Validate 仍将尝试验证 `Confdir` 特性。Validate 使用 `scha_resource_get()` 从静态配置中获取 `Confdir` 的值。然后它将执行相同的检查操作，以检验该配置目录是否可以存取并包含非空的 `named.conf` 文件。

如果 Validate 在失败状态下退出，所有特性（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

RGM 将向 Validate 方法传送一组与其它回叫方法不同的参数，因此 Validate 需要使用与其它方法的函数不同的函数来分析变量。有关传送到 Validate 以及其它回叫方法的参数的详细信息，请参阅 `rt_callbacks(1HA)` 手册页。以下内容说明了 Validate `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
}

```

与其它方法的 `parse_args()` 函数一样，此函数将提供一个用于捕获资源名称的标记 (R)、用于捕获资源组名称的标记 (G) 和用于捕获 RGM 所传送的资源类型的标记 (T)。

将忽略 `r` 标记（用于表示系统定义的特性）、`g` 标记（用于表示资源组特性）和 `c` 标记（用于表示在创建资源时已进行了检验），因为当更新资源时将调用此方法来验证扩展特性。

u 标记将 UPDATE\_PROPERTY shell 变量的值设置为 1 (TRUE)。x 标记捕获所更新的特性的名称和值。如果 Confdir 是所更新的特性之一，则其值将放置在 CONFDIR shell 变量中，且变量 CONFDIR\_FOUND 将设置为 1 (TRUE)。

## 验证 Confdir

在 MAIN 函数中，Validate 首先将 CONFDIR 变量设置成空字符串，并将 UPDATE\_PROPERTY 和 CONFDIR\_FOUND 设置成 0。

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

然后，Validate 将调用 parse\_args() 分析 RGM 传送的变量。

```
parse_args "$@"
```

Validate 随后将检查特性更新后是否调用了 Validate，并检查 Confdir 扩展特性是否位于命令行上。接下来，Validate 将检验 Confdir 特性是否具有值，如果没有，将在失败状态下退出并记录一条错误消息。

```
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
```

---

注意 – 具体地说，上述代码将检查更新后 (\$UPDATE\_PROPERTY == 1) 是否调用了 Validate，并检查该特性是否位于命令行上，如果不在命令行上 (CONFDIR\_FOUND == 0)，它将使用 scha\_resource\_get() 检索 Confdir 的现有值。如果发现 Confdir 位于命令行上 (CONFDIR\_FOUND == 1)，则 CONFDIR 的值由 parse\_args() 函数确定，而不是由 scha\_resource\_get() 确定。

---

然后，Validate 方法将使用 CONFDIR 的值检验该目录是否可以存取。如果不可以存取，Validate 将记录一条错误消息并在错误状态下退出。

```
# 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 特性的更新之前，Validate 将执行 final 检查来检验 named.conf 文件是否存在。如果不存在，该方法将记录一条错误消息并在错误状态下退出。

```
# 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
```

如果该 final 检查通过，Validate 将记录一条表明检查成功的消息并在成功状态下退出。

```
# 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 在成功 (0) 状态下退出，则使用新值创建 Confdir。如果 Validate 在失败 (1) 状态下退出，将不会创建 Confdir 和其它任何特性并记录一条消息，说明发送至群集管理员的原因。

## Update 方法

RGM 调用 Update 方法通知运行资源其特性已被更改。在设置资源及其组的特性过程中成功地执行了管理操作后，RGM 将调用 Update。在资源联机的节点上调用此方法。

## Update 概述

Update 方法并不更新特性，此操作由 RGM 完成，而该方法将通知运行进程已进行了更新。数据服务样例中唯一受特性更新操作影响的进程是故障监视器，因此 Update 方法将停止和重启此进程。

Update 方法必须检验故障监视器是否正在运行，如果正在运行，将使用 pmfadm 终止该监视器。该方法将获取实现故障监视器的探测程序的位置，然后再次使用 pmfadm 重启该程序。

## 使用 Update 停止监视器

Update 方法将使用 `pmfadm -q` 检验监视器是否正在运行，如果正在运行，它将使用 `pmfadm -s TERM` 终止该监视器。成功终止监视器后，将向管理用户发送表明相应情况的消息。如果无法停止监视器，Update 将在失败状态下退出并向管理用户发送一条错误消息。

```
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 ${SYSLOG_FACILITY}.err \
      -t [$RESOURCE_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。该探测程序驻留在数据服务的基目录上，由 `Rt_basedir` 特性指向该目录。Update 将检索 `Rt_basedir` 的值并将其存储在 `RT_BASEDIR` 变量中，如下所示。

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

然后，Update 将通过 `pmfadm` 使用 `RT_BASEDIR` 的值来重启 `dns_probe` 程序。如果成功，Update 将在成功状态下退出并向管理用户发送一条表明相应情况的消息。如果 `pmfadm` 无法启动该探测程序，Update 将在失败状态下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。此状态对资源的 RGM 管理并无影响，但是它表明通过 `syslog` 工具更新管理工具的操作失败。



## 第 6 章

---

# 数据服务开发库 (DSDL)

---

本章概要介绍了组成数据服务开发库或 DSDL 的应用程序编程接口。DSDL 在 `libdsdev.so` 库中实现，它包含在 Sun Cluster 软件包中。

本章包括以下主题：

- 第 103 页 “DSDL 概述”
- 第 103 页 “管理配置特性”
- 第 104 页 “启动和停止数据服务”
- 第 105 页 “实现故障监视器”
- 第 105 页 “存取网络地址信息”
- 第 106 页 “调试资源类型实现”

---

## DSDL 概述

DSDL API 位于 RMAPI 的顶层。因此，它并没有取代 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，因此您可以将大部分开发时间用在您的应用程序内部的高可用性和可伸缩问题上，避免在将应用程序的启动、关闭和监视过程与 Sun Cluster 相集成这一问题上花费大量时间。

---

## 管理配置特性

所有回叫方法都需要存取配置特性。通过执行以下操作，DSDL 支持对特性的存取：

- 初始化环境
- 提供一组公用函数以检索特性值

每个回叫方法开始执行时必须调用 `scds_initialize` 函数，此函数可执行以下操作：

- 检查并处理 RGM 传送到回叫方法的命令行变量（`argc` 和 `argv []`），而无需编写命令行分析函数。
- 设置其它 DSDL 函数使用的内部数据结构。例如，从 RGM 检索特性值的公用函数将在这些结构中存储检索出来的值，同样，在这些数据结构中也将存储命令行值，命令行值优先于从 RGM 检索出来的值。

---

**注意** – 对于 `Validate` 方法，`scds_initialize` 将分析传送到命令行的特性值，而无需编写 `Validate` 的分析函数。

---

`scds_initialize` 函数还将初始化记录环境并验证故障监视器的探测设置。

DSDL 提供了多组函数，以检索资源、资源类型和资源组特性以及常用的扩展特性。通过采用以下惯例，这些函数可对特性的存取操作进行标准化。

- 每个函数仅使用一个句柄变量（由 `scds_initialize` 返回）。
- 每个函数都对应一个特定的特性。函数的返回值类型与其检索的特性值的类型相匹配。
- 因为已预先通过 `scds_initialize` 计算了这些值，所以函数不会返回错误消息。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` 方法将用来执行在群集节点上启动数据服务时所需的操作。通常，这些操作包括检索资源特性，定位特定于应用程序的可执行文件和配置文件，以及使用相应的命令行变量启动应用程序。

`scds_initialize` 函数用来检索资源配置。`Start` 方法可以使用特性的公用函数来检索特定特性（例如 `Confdir_list`）的值，这些特性用于标识要启动的应用程序的配置目录和文件。

`Start` 方法可以在进程监视工具 (PMF) 的控制下调用 `scds_pmf_start` 来启动应用程序。PMF 可以用来指定要应用到进程中的监视级别，并可以在失败的情况下重启该进程。要获得使用 DSDL 实现的 `Start` 方法的实例，请参阅第 120 页“`xfnts_start` 方法”。

`Stop` 方法必须具有幂等性，以便即使是在应用程序不运行的情况下对节点调用了该方法，它也可以在成功状态下退出。如果 `Stop` 方法失败，则要停止的资源被设置成 `STOP_FAILED` 状态，这将导致群集的硬重新启动。

要避免使资源处于 `STOP_FAILED` 状态，`stop` 方法必须全力停止该资源。`scds_pmf_stop` 函数将进行阶段性尝试以停止该资源。它首先将使用 `SIGTERM` 信号尝试停止该资源，如果失败，它将使用 `SIGKILL` 信号。有关详细信息，请参阅 `scds_pmf_stop(3HA)`。

---

## 实现故障监视器

通过提供一个预设模型，DSDL 大大降低了实现故障监视器的复杂性。当在节点上启动资源时，`Monitor_start` 方法将在 PMF 的控制下启动故障监视器。只要该资源在节点上运行，故障监视器也将循环运行。DSDL 故障监视器的高级逻辑如下：

- `scds_fm_sleep` 函数使用 `Thorough_probe_interval` 特性确定探测操作之间的时间值。在此间隔中 PMF 确定的任何应用程序进程失败都将导致资源的重启。
- 探测程序本身将返回一个值，表明失败的严重级别，值的范围是从 0（无失败）到 100（完全失败）。
- 探测程序返回的值将发送到 `scds_action` 函数，该函数将在 `Retry_interval` 特性指定的间隔内维护累积的失败历史记录。
- `scds_action` 函数将确定失败时应该执行什么操作，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），将重启该数据服务。如果超出 `Retry_interval` 的值，将重置该历史记录。
  - 如果重启的次数超出了 `Retry_count` 特性的值，则将在 `Retry_interval` 指定的时间内对该数据服务进行故障切换。

---

## 存取网络地址信息

DSDL 提供了公用函数，以返回资源和资源组的网络地址信息。例如，`scds_get_netaddr_list` 用来检索资源所使用的网络地址资源，并启用故障监视器来探测该应用程序。

DSDL 还提供了一组用来进行基于 TCP 的监视操作的函数。这些函数通常与某个服务建立简单的套接字连接，向服务读取和写入数据，然后断开与该服务的连接。探测结果可以发送到 DSDL `scds_fm_action` 函数，以确定要执行的操作。

要获得基于 TCP 的故障监视的实例，请参阅第 133 页“`xfnts_validate` 方法”。

---

## 调试资源类型实现

DSDL 具有可帮助您调试数据服务的内置功能。

DSDL 公用程序 `scds_syslog_debug()` 提供了一个基本框架，可用于向资源类型实现添加调试语句。可以按每个群集节点上的每个资源类型实现动态设置调试级别（1 至 9 之间的数字）。名为 `/var/cluster/rgm/rt/rtname/loglevel` 的文件（仅包含 1 至 9 之间的整数）供所有资源类型回叫方法读取。DSDL 例行程序 `scds_initialize()` 将读取此文件并在内部将调试级别设置为指定级别。缺省调试级别是 0，指定数据服务不记录任何调试消息。

`scds_syslog_debug()` 函数将在 `LOG_DEBUG` 优先级别下使用 `scha_cluster_getlogfacility()` 函数返回的工具。您可以在 `/etc/syslog.conf` 中配置这些调试消息。

您可以使用 `scds_syslog` 公用程序将一些调试消息转换成资源类型常规操作的说明性消息（可能在 `LOG_INFO` 优先级别下进行）。如果您查看一下第 8 章中的 DSDL 应用程序样例，就可以发现它可以自由地使用 `scds_syslog_debug` 和 `scds_syslog` 函数。

---

## 启动具有高可用性的本地文件系统

您可以使用 `HASStoragePlus` 资源类型使本地文件系统在 Sun Cluster 环境中高度可用。本地文件系统分区必须位于全局磁盘组中。必须启用关系转换，还要将 Sun Cluster 环境配置为可以进行故障切换。通过此设置，用户可以将多主机磁盘上的任意文件系统设置为可与这些多主机磁盘直接相连接的任意主机上存取。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 数据服务规划和管理指南（适用于 Solaris OS）*》中的“启用具有高可用性的本地文件系统”中包含了有关配置 `HASStoragePlus` 资源类型的信息。

## 第 7 章

---

# 设计资源类型

---

本章介绍了 DSDL 在设计和实现资源类型方面的典型用法，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。在本章的最后还介绍了如何使用 DSDL 实现资源类型回叫方法。

有关其它信息，请参阅 `rt_callbacks(1HA)` 手册页。

要完成这些任务，您需要存取资源的特性设置。DSDL 公用程序 `scds_initialize()` 向您提供了一种用来存取资源特性的统一方法。本函数设计成在每个回叫方法的开头部分进行调用。此公用程序函数用来从群集框架检索资源的所有特性，并使其可供 `scds_getname()` 系列函数使用。

本章包括以下主题：

- 第 107 页 “RTR 文件”
- 第 108 页 “Validate 方法”
- 第 109 页 “Start 方法”
- 第 110 页 “Stop 方法”
- 第 111 页 “Monitor\_start 方法”
- 第 111 页 “Monitor\_stop 方法”
- 第 112 页 “Monitor\_check 方法”
- 第 112 页 “Update 方法”
- 第 113 页 “Init、Fini 和 Boot 方法”
- 第 113 页 “设计故障监视器守护程序”

---

## RTR 文件

资源类型登记 (RTR) 文件是资源类型的重要组成部分。此文件用来向 Sun Cluster 指定资源类型的详细信息。这些详细信息包括实现所需的特性、这些特性的数据类型以及缺省值、资源类型实现的回叫方法的文件系统路径以及系统定义的特性的各种设置。

DSDL 附带的 RTR 文件样例应该能够满足大多数资源类型实现的要求。您仅需编辑一些基本的元素，例如资源类型的名称和该资源类型回叫方法的路径名。如果实现该资源类型时需要一个新的特性，您可以在该资源类型实现的资源类型登记 (RTR) 文件中将该特性声明为扩展特性，然后使用 `DSDL scds_get_ext_property()` 公用程序存取新特性。

---

## Validate 方法

RGM 会在两种情况下调用资源类型实现的 `Validate` 方法：1) 创建资源类型的新资源时；2) 更新资源或资源组的特性时。这两种方案可以根据传送到资源的 `Validate` 方法的命令行选项 `-c` (创建) 或 `-u` (更新) 是否存在来区分。

对一组节点中的每一个节点都调用 `Validate` 方法，其中该组节点都是由资源类型特性 `INIT_NODES` 的值定义的。如果 `INIT_NODES` 设置为 `RG_PRIMARYES`，则将对每一个可以承载含有该资源的资源组的节点 (主节点) 调用 `Validate`。如果 `INIT_NODES` 设置为 `RT_INSTALLED_NODES`，则将对每一个安装有资源类型软件的节点 (通常是群集中的所有节点) 调用 `Validate`。`INIT_NODES` 的缺省值为 `RG_PRIMARYES` (请参阅 `rt_reg(4)`)。调用 `Validate` 方法时，RGM 尚未创建资源 (在创建回叫的情况下) 或尚未应用所更新特性的更新后的值 (在更新回叫的情况下)。资源类型实现的 `Validate` 回叫方法的用途是检查所建议的资源设置 (按照该资源的建议特性设置指定) 是否适用于该资源类型。

---

**注意** - 如果您使用 `HASStoragePlus` 管理的本地文件系统，则使用 `scds_hasp_check` 检查 `HASStoragePlus` 资源的状态。使用为该资源定义的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系统特性，从资源依赖的所有 `SUNW.HASStoragePlus(5)` 资源中获取此信息。有关 `scds_hasp_check` 调用返回的状态码的完整列表，请参阅 `scds_hasp_check(3HA)`。

---

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

- 针对创建资源的情况，它将分析传送到命令行的所建议的资源特性，从而资源类型开发者可以使用所建议的资源特性值，就象已经在系统中创建了该资源一样。
- 针对更新资源或资源组的情况，将从命令行读取管理员所更新的特性的建议值；对于其余特性 (值未进行更新)，将使用资源管理 API 从 `Sun Cluster` 读取。使用 DSDL 的资源类型开发者无需自己执行所有这些内务处理任务，资源的验证就可以完成，就象该资源的所有特性都可供开发者使用一样。

假设实现资源特性验证的函数为 `svc_validate()`，它使用 `scds_get_name()` 系列函数来查看它要验证的特性。假设可接受的资源设置由此函数的 0 返回代码表示，则该资源类型的 `Validate` 方法可以通过以下代码段表示：

```
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);
}

```

该验证函数还应记录资源验证失败的原因。如果不考虑相应细节内容（有关对验证函数的更为实际的处理方法，请参阅下一章），简单的 `svc_validate()` 函数实例可用以下方式实现：

```

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，资源类型的开发者仅需执行 `svc_validate()` 函数的实现任务。资源类型实现的典型实例可以确保名为 `app.conf` 的应用程序配置文件存在于 `Confdir_list` 特性中。通过对源自 `Confdir_list` 特性的相应路径名进行 `stat()` 系统调用，就可以很方便地实现此目的。

---

## Start 方法

RGM 将对所选群集节点调用资源类型实现的 `Start` 回叫方法，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` 方法应执行在群集节点上启动数据服务资源时所需的操作。这些操作通常涉及检索资源特性、定位应用程序特定的可执行文件和/或配置文件以及使用相应的命令行变量启动该应用程序。

借助 DSDL，资源配置已可以被 `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` 方法的简化代码为：

```

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，可能是因为资源配置和应用程序配置不匹配，也可能是因为当前与系统相关的这个群集节点上存在问题。例如，当前此群集节点上可能没有提供该资源所需的全局文件系统。在这种情况下，在此群集节点上尝试启动该资源是无用的，最好是让 RGM 在其它节点上启动该资源。注意，以上假设 `svc_validate()` 是十分保守的（因此它仅检查群集节点上应用程序绝对需要的资源），否则资源可能无法在所有群集节点上启动并处于 `START FAILED` 状态。有关对此状态的说明，请参阅 `scswitch(1M)` 和《*Sun Cluster 数据服务规划和管理指南（适用于 Solaris OS）*》。

如果资源在节点上成功启动，则 `svc_start()` 函数必须返回 0。如果启动函数遇到了问题，则它必须返回一个非零值。此函数失败后，RGM 将尝试在其它群集节点上启动该资源。

为了尽可能充分地利用 DSDL，`svc_start()` 函数可以使用 `scds_pmf_start()` 公用程序在进程管理工具 (PMF) 控制之下启动该应用程序。此公用程序还将利用 PMF 的失败回叫操作功能（请参阅 `pmfadm(1M)` 中的 `-a` 操作标志）来实现进程失败检测。

---

## Stop 方法

RGM 将对群集节点调用资源类型实现的 `Stop` 回叫方法，以停止该应用程序。`Stop` 方法的回叫语义要求：

- `Stop` 方法必须具有**幂等性**，因为即使 `Start` 方法在节点上未成功完成，RGM 也能调用 `Stop` 方法。因此，即使应用程序未在群集节点上运行并且没有要执行的操作，`Stop` 方法也必须成功（在返回零的情况下退出）。
- 如果资源类型的 `Stop` 方法在群集节点上失败（在返回非零值的情况下退出），则要停止的资源将在 `STOP_FAILED` 状态下停止。是否会导致 RGM 对群集节点进行硬重新引导，这取决于资源的 `Failover_mode` 设置。因此，设计 `Stop` 方法尤为重要，使其可以在应用程序无法终止时尽力尝试真正停止该应用程序，甚至使用硬终止或突然终止该应用程序的方法（例如，使用 `SIGKILL`）。还应确保它及时地执行此操作，因为框架将 `Stop_timeout` 期满视作停止失败，并使资源处于 `STOP_FAILED` 状态。

对于大多数应用程序来说，DSDL 公用程序 `scds_pmf_stop()` 应该就满足要求，因为它首先试图软（通过 `SIGTERM`）停止应用程序（假设它是通过 `scds_pmf_start()` 在 PMF 控制下启动的），然后向进程发送 `SIGKILL`。有关此公用程序的详细信息，请参阅第 176 页“PMF 函数”。

以下是迄今为止我们一直在使用的代码模型，其中假设用来停止应用程序的应用程序特定函数为 `svc_stop()`（`svc_stop()` 的实现是否使用 `scds_pmf_stop()` 与此处介绍的内容无关，而是取决于该应用程序是否是在 PMF 控制下通过 `Start` 方法启动的），`Stop` 方法可用以下方式实现：

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

`svc_validate()` 方法不用于 `Stop` 方法的实现，因为即使系统当前存在问题，`Stop` 方法应试图停止此节点上的应用程序。

---

## Monitor\_start 方法

RGM 将调用 `Monitor_start` 方法来启动资源的故障监视器。故障监视器用来监视资源管理的应用程序的运行情况。资源类型实现通常将故障监视器作为在后台运行的独立守护程序来实现。`Monitor_start` 回叫方法用来通过合适的变量启动此守护程序。

因为监视器守护程序本身易于失败（例如，它可能毁坏，从而使应用程序处于不受监视的状态），所以您应该使用 PMF 启动该监视器守护程序。DSDL 公用程序 `scds_pmf_start()` 具有启动故障监视器的内置支持。此公用程序使用监视器守护程序的相对路径名（相对于资源类型回叫方法实现的位置的 `RT_basedir`）。它使用由 DSDL 管理的 `Monitor_retry_interval` 和 `Monitor_retry_count` 扩展特性，以避免会无限多次地重启该守护程序。虽然 RGM 从不直接调用监视器守护程序，但它仍然将与为所有回叫方法定义的命令行语法相同的语法（即 `-R resource -G resource_group -T resource_type`）强加到该监视器守护程序中。它允许监视器守护程序实现本身利用 `scds_initialize()` 公用程序来设置其自己的环境。主要用于设计监视器守护程序本身方面。

---

## Monitor\_stop 方法

RGM 将调用 `Monitor_stop` 方法来停止原先通过 `Monitor_start` 方法启动的故障监视器守护程序。此回叫方法的故障处理方式与 `Stop` 方法完全相同，因此 `Monitor_stop` 方法必须像 `Stop` 方法一样等幂和强大。

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

---

## Monitor\_check 方法

可以对指定资源的某节点调用资源中的 `Monitor_check` 回叫方法，以确定该群集节点是否可以控制该资源（即该资源所管理的应用程序是否可以在该节点上成功运行）。通常此情况涉及到要确保该群集节点上确实提供了应用程序所需的所有系统资源。如第 108 页“`Validate` 方法”中所述，由开发者实现的函数 `svc_validate()` 可用于确定此方面的情况。

根据资源类型实现所管理的具体应用程序，也可以编写 `Monitor_check` 方法，以执行一些附加任务。必须实现 `Monitor_check` 方法，以免该方法与同时运行的其它方法发生冲突。对于使用 DSDL 的开发者，建议 `Monitor_check` 方法利用为实现资源特性的应用程序特定验证这一目的而编写的 `svc_validate()` 函数。

---

## Update 方法

RGM 调用资源类型实现的 `Update` 方法来应用系统管理员对活动资源的配置进行的更改。仅对资源当前处于联机状态的节点（如果有）调用 `Update` 方法。

刚才对资源配置所进行的更改一定会适用于资源类型实现，因为 RGM 在运行 `Update` 方法之前先运行了该资源类型的 `Validate` 方法。`Validate` 方法在更改资源或资源组特性之前调用，`Validate` 方法可以否决所建议的更改。`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。

作为资源类型开发者，您需要谨慎决定您希望能够动态更新的特性，并在 RTR 文件中用 `TUNABLE = ANYTIME` 设置来标记这些特性。通常，如果 `Update` 方法的实现可以重启故障监视器守护程序，您就可以指定您希望能够动态更新该故障监视器守护程序所使用的资源类型实现的任意特性。

可能要更改的特性包括：

- `Thorough_Probe_Interval`
- `Retry_Count`
- `Retry_Interval`
- `Monitor_retry_count`
- `Monitor_retry_interval`
- `Probe_timeout`

这些特性会影响故障监视器守护程序对该服务的运行情况进行检查的方式以及频率、用来跟踪错误的历史间隔和 PMF 对其设置的重启阈值。为实现这些特性的更新，在 DSDL 中提供了公用程序 `scds_pmf_restart()`。

如果您需要能够动态更新资源特性，而对该特性进行修改又可能会影响运行应用程序，则需要执行相应操作，以便对该特性进行的更新可以正确地应用到该应用程序的所有运行实例中。当前无法通过 DSDL 使该操作变得更容易。Update 不向命令行传递修改后的特性（虽然 Validate 进行了传递）。

---

## Init、Fini 和 Boot 方法

这些是按照资源管理 API 规范定义的一次性操作方法。DSDL 附带的实现样例中并未说明这些方法的使用。然而，如果资源类型开发者需要使用这些方法，也可以使用 DSDL 中的所有工具。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。Fini 方法通常用来执行撤消 Init 或 Boot 方法的操作。

---

## 设计故障监视器守护程序

使用 DSDL 的资源类型实现通常都具有一个故障监视器守护程序，该守护程序可执行以下任务。

- 定期监视所管理的应用程序的运行情况。监视器守护程序的这一特定方面主要由应用程序决定，各个资源类型之间可能具有较大差别。DSDL 具有一些内置公用程序函数，用来检查基于 TCP 的简单服务的运行情况。应用程序实现基于 ASCII 码的协议，例如 HTTP、NNTP、IMAP 和 POP3 都可以通过这些公用程序实现。
- 使用资源特性 `Retry_interval` 和 `Retry_count` 跟踪该应用程序遇到的问题。当该应用程序完全失败时，确定 PMF 操作脚本是否应重启该服务，或确定是否已迅速积累了较多应用程序失败从而需要考虑进行故障切换。DSDL 公用程序 `scds_fm_action()` 和 `scds_fm_sleep()` 可用来帮助您实现此机制。
- 采取相应的操作（通常是重启该应用程序或尝试故障切换包含的资源组）。DSDL 公用程序 `scds_fm_action()` 可实现这样的算法。为此它将计算过去 `Retry_interval` 时间（以秒为单位）内探测失败的当前积累值。
- 更新资源状态以使应用程序的运行状态可供 `scstat` 命令和群集管理 GUI 使用。

已对 DSDL 公用程序进行了设计，因此故障监视器守护程序的主循环可以用以下伪代码表示。

对于使用 DSDL 实现的故障监视器，

- `scds_fm_sleep()` 检测到应用程序进程遭到破坏的速度相当快，因为通过 PMF 发出的进程破坏通知是异步的。将该情况与频繁唤醒故障监视器以检查服务的运行情况并查找破坏的应用程序的情况相比较，故障检测时间大幅度减少了，从而提高了该服务的可用性。
- 如果 RGM 拒绝了通过 `scha_control(3HA)` API 对该服务进行故障切换的尝试操作，则 `scds_fm_action()` 将重置（遗忘）其当前失败历史记录。原因在于失败历史记录已超出 `Retry_count`，如果故障监视器守护程序在下一个重复期内被唤醒并无法成功完成守护程序的运行情况检查，则将再次试图调用 `scha_control()`，这可能仍然被拒绝，因为上一个重复期内导致这被拒绝的状态仍然有效。重置历史记录可以保证故障监视器至少会在下一个重复期内尝试在本地改善该情况（例如，通过应用程序重启的方式）。
- 在重启失败的情况下，`scds_fm_action()` 不会重置应用程序失败历史记录，因为如果情况本身没有进行改善，通常会立即尝试使用 `scha_control()`。
- 根据失败历史记录的不同，公用程序 `scds_fm_action()` 会相应地将资源状态更新为 `SCHA_RSSTATUS_OK`、`SCHA_RSSTATUS_DEGRADED` 或 `SCHA_RSSTATUS_FAULTED`。这样此状态就可用于群集系统管理。

在大多数情况下，应用程序特定运行情况检查操作可以在独立的公用程序（例如 `svc_probe()`）中实现并与此普通主循环相集成。

```
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

---

本章介绍使用 DSDL 实现的资源类型 `SUNW.xfnts` 的样例。该数据服务是用 C 语言编写的。基础应用程序是 X Font Server，一种基于 TCP/IP 的服务。

本章内容包括：

- 第 117 页 “X Font Server”
- 第 119 页 “SUNW.xfnts RTR 文件”
- 第 119 页 “`scds_initialize()` 函数”
- 第 120 页 “`xfnts_start` 方法”
- 第 124 页 “`xfnts_stop` 方法”
- 第 125 页 “`xfnts_monitor_start` 方法”
- 第 126 页 “`xfnts_monitor_stop` 方法”
- 第 127 页 “`xfnts_monitor_check` 方法”
- 第 127 页 “SUNW.xfnts 故障监视器”
- 第 133 页 “`xfnts_validate` 方法”

---

## X Font Server

X Font Server 是一种简单的基于 TCP/IP 的服务，它可以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 守护程序包含一个服务器二进制文件 `/usr/openwin/bin/xfns`。虽然通常从 `inetd` 启动该守护程序，但是，对于当前样例，假设已禁用 `/etc/inetd.conf` 文件中的相应项（例如，通过 `fsadmin -d` 命令），这样守护程序就在 Sun Cluster 的单一控制下。

## X Font Server 配置文件

缺省情况下，X Font Server 从文件 `/usr/openwin/lib/X11/fontserver.cfg` 中读取其配置信息。此文件中的目录项包含了守护程序可以提供的字体目录列表。群集管理员可以将字体目录定位到全局文件系统中（以通过在系统上维护单个字体数据库副本来优化 X Font Server 在 Sun Cluster 中的使用）。如果执行此操作，管理员必须编辑 `fontserver.cfg` 来反映该字体目录的新路径。

为了可以轻松地进行配置，管理员也可以将配置文件本身放置在全局文件系统中。`xfss` 守护程序提供了命令行变量，以覆盖此文件的缺省内置位置。`SUNW.xfnts` 资源类型使用以下命令启动受 Sun Cluster 控制的守护程序。

```
/usr/openwin/bin/xfss -config <location_of_cfg_file>/fontserver.cfg \  
-port <portnumber>
```

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

## TCP 端口号

`xfss` 服务器守护程序所侦听的 TCP 端口号通常是“fs”端口（其端口号在 `/etc/services` 文件中通常定义为 7100）。然而，使用 `xfss` 命令行上的 `-port` 选项，系统管理员可以覆盖该缺省设置。可以使用 `SUNW.xfnts` 资源类型中的 `Port_list` 特性设置缺省值，并支持在 `xfss` 命令行上使用 `-port` 选项。在 RTR 文件中您需要将此特性的缺省值定义为 `7100/tcp`。在 `SUNW.xfnts Start` 方法中，将 `Port_list` 传送到 `xfss` 命令行上的 `-port` 选项。因此，不要求此资源类型的用户指定端口号（缺省端口为 `7100/tcp`），但是，如果需要，当配置资源类型时，通过为 `Port_list` 特性指定不同的值，可以指定其它端口。

---

## 命名惯例

您可以采用以下命名惯例来标识多个代码样例段。

- RMAPI 函数以 `scha_` 开头。
- DSDL 函数以 `scds_` 开头。
- 回叫方法以 `xfnts_` 开头。
- 用户编写的函数以 `svc_` 开头。

---

## SUNW.xfnts RTR 文件

本节介绍 SUNW.xfnts RTR 文件中的几个主要特性。本节并未介绍该文件中每个特性的作用，有关此方面的介绍，请参阅第 29 页“设置资源和资源类型特性”。

Confdir\_list 扩展特性用来标识配置目录（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 特性未指定缺省值。创建资源时，群集管理员必须指定一个目录。因为可调性被限制为 AT\_CREATION，因此创建以后无法更改此值。

Port\_list 特性用来标识服务器守护程序所侦听的端口，如下所示。

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

因为该特性声明了一个缺省值，所以在创建资源时，群集管理员既可以选择指定一个新值，又可以接受该缺省值。因为可调性被限制为 AT\_CREATION，因此创建以后无法更改此值。

---

## scds\_initialize() 函数

DSDL 要求每个回叫方法在该方法的开头调用 scds\_initialize(3HA) 函数。此函数可执行以下操作：

- 检查和处理框架传送到数据服务方法的命令行变量（argc 和 argv）。该方法不必额外处理命令行变量。
- 设置内部数据结构，以供 DSDL 中的其它函数使用。
- 初始化记录环境。
- 验证故障监视器探测程序设置。

使用 scds\_close() 函数回收通过 scds\_initialize() 分配的资源。

---

## xfnts\_start 方法

使含有数据服务资源的资源组在群集节点上联机时或启用该资源时，RGM 将对该节点调用 start 方法。在 SUNW.xfnts 资源类型样例中，xfnts\_start 方法将激活该节点上的 xfs 守护程序。

xfnts\_start 方法将在 PMF 控制下调用 scds\_pmf\_start() 启动该守护程序。除了与故障监视器集成以外，PMF 还提供了自动的失败通知并重启的功能。

---

**注意** - 在 xfnts\_start 中首先调用的是 scds\_initialize()，该函数执行一些必要的内务处理功能（第 119 页“scds\_initialize() 函数”和 scds\_initialize(3HA) 手册页中包含详细信息）。

---

## 在启动之前验证服务

在尝试启动 X Font Server 之前，xfnts\_start 方法将调用 svc\_validate() 验证是否已进行了适当的配置，以支持 xfs 守护程序（有关详细信息，请参阅第 133 页“xfnts\_validate 方法”），如下所示。

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

## 启动服务

xfnts\_start 方法将调用 xfnts.c 中定义的 svc\_start() 方法来启动 xfs 守护程序。本节将介绍 svc\_start()。

用来启动 xfs 守护程序的命令如下所示。

```
xfs -config config_directory/fontserver.cfg -port port_number
```

Confdir\_list 扩展特性用来标识 *config\_directory*，Port\_list 系统特性用来标识 *port\_number*。群集管理员配置数据服务时，需要提供这些特性的具体值。

xfnts\_start 方法将使用 scds\_get\_ext\_confdir\_list() 和 scds\_get\_port\_list() 函数将这些特性声明为字符串数组并获取管理员设置的值（在 scds\_property\_functions(3HA) 中进行介绍），如下所示。

```
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` 变量指向该数组的第一个元素 (0)。

`xfnts_start` 方法使用 `sprintf` 组成 `xfs` 命令行，如下所示。

```

/* 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，输出将重定向到 `dev/null`，以抑制该守护程序生成的消息。

`xfnts_start` 方法将把 `xfs` 命令行传送到 `scds_pmf_start()`，以在 PMF 控制下启动该数据服务，如下所示。

```

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 ");
}

```

请注意以下几点关于对 `scds_pmf_start()` 进行调用的内容。

- `SCDS_PMF_TYPE_SVC` 参数用来标识要作为数据服务应用程序启动的程序 — 此方法也可以启动故障监视器或一些其它类型的应用程序。
- `SCDS_PMF_SINGLE_INSTANCE` 参数用来将此资源标识为单实例资源。
- `cmd` 参数是指先前生成的命令行。
- 最后一个参数 `-1` 用来指定子监视级别。 `-1` 指定 PMF 要监视所有子进程以及原始进程。

在返回之前，`scds_pmf_start()` 将释放为 `portlist` 结构分配的内存，如下所示。

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

## 从 `svc_start()` 返回

即使 `svc_start()` 成功地返回，基础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。该探测程序还必须将应用程序可能无法立即可用这一点考虑在内，因为该程序需要一些时间来启动。

`svc_start()` 方法将调用在 `xfnts.c` 中定义的 `svc_wait()` 来检验应用程序是否正在运行，如下所示。

```
/* 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(3HA)` 来获取探测应用程序时所需的网络地址资源，如下所示。

```
/* 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);
}
```

然后 `svc_wait()` 将获取 `start_timeout` 和 `stop_timeout` 的值，如下所示。

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

考虑到该服务器启动可能花费的时间，`svc_wait()` 将调用 `scds_svc_wait()` 并传递等于 `start_timeout` 值的百分之三的超时值。然后 `svc_wait()` 将调用 `svc_probe()` 来检验该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。如果无法连接该端口，`svc_probe()` 将返回值 100，该值表明操作完全失败。如果连接成功但是断开连接操作失败，`svc_probe()` 将返回值 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` 特性指定的时间内尝试次数超过资源的 `Retry_count` 特性的值，则 `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);
```

---

注意 - 退出前，`xfnts_start` 方法调用 `scds_close()` 来回收由 `scds_initialize()` 分配的资源。有关详细信息，请参阅第 119 页“`scds_initialize()` 函数”和 `scds_close(3HA)` 手册页。

---

---

## xfnts\_stop 方法

因为 `xfnts_start` 方法使用 `scds_pmf_start()` 在 PMF 控制下启动服务，所以 `xfnts_stop` 将使用 `scds_pmf_stop()` 停止该服务。

---

**注意** – 在 `xfnts_stop` 中首先调用的是 `scds_initialize()`，该函数可以执行一些必要的内务处理功能（第 119 页“`scds_initialize()` 函数”和 `scds_initialize(3HA)` 手册页中包含了详细信息）。

---

`xfnts_stop` 方法将调用在 `xfnts.c` 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 */
```

请注意以下关于在 `svc_stop()` 中调用 `scds_pmf_stop()` 函数的内容。

- `SCDS_PMF_TYPE_SVC` 参数用来标识要作为数据服务应用程序停止的程序 — 此方法也可以停止故障监视器或一些其它类型的应用程序。
- `SCDS_PMF_SINGLE_INSTANCE` 参数标识信号。
- `SIGTERM` 参数标识用来停止该资源实例的信号。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，请参阅 `scds_pmf_stop(3HA)` 手册页。
- 超时值即该资源的 `Stop_timeout` 特性的值。

---

**注意** – 在退出之前，`xfnts_stop` 方法将调用 `scds_close()` 回收通过 `scds_initialize()` 分配的资源。有关详细信息，请参阅第 119 页“`scds_initialize()` 函数”和 `scds_close(3HA)` 手册页。

---

---

## xfnts\_monitor\_start 方法

在节点上启动资源后，RGM 将对该节点调用 Monitor\_start 方法，以启动故障监视器。xfnts\_monitor\_start 方法将使用 scds\_pmf\_start() 在 PMF 的控制下启动监视器守护程序。

---

**注意** – 在 xfnts\_monitor\_start 中首先调用的是 scds\_initialize()，该函数可以执行一些必要的**内务处理**功能（第 119 页“scds\_initialize() 函数”和 scds\_initialize(3HA) 手册页中包含了详细信息）。

---

xfnts\_monitor\_start 方法将调用在 xfnts.c 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 */
}
```

请注意以下关于在 svc\_mon\_start() 中调用 scds\_pmf\_start() 函数的内容。

- SCDS\_PMF\_TYPE\_MON 参数用来标识要作为故障监视器启动的程序 — 此方法也可以启动数据服务或一些其它类型的应用程序。
- SCDS\_PMF\_SINGLE\_INSTANCE 参数用来将此资源标识为单实例资源。
- xfnts\_probe 参数用来标识要启动的监视器守护程序。假设监视器守护程序与其它回叫程序位于同一个目录。
- 最后一个参数 0 用来指定子监视级别 — 在这种情况下，仅监视监视器守护程序。

---

**注意** – 在退出之前，`xfnts_monitor_start` 方法将调用 `scds_close()` 来回收通过 `scds_initialize()` 分配的资源。有关详细信息，请参阅第 119 页“`scds_initialize()` 函数”和 `scds_close(3HA)` 手册页。

---

---

## xfnts\_monitor\_stop 方法

因为 `xfnts_monitor_start` 方法使用 `scds_pmf_start()` 在 PMF 的控制下启动监视器守护程序，所以 `xfnts_monitor_stop` 将使用 `scds_pmf_stop()` 停止该监视器守护程序。

---

**注意** – 在 `xfnts_monitor_stop` 中首先调用的是 `scds_initialize()`，该函数可以执行一些必要的内务处理功能（第 119 页“`scds_initialize()` 函数”和 `scds_initialize(3HA)` 手册页中包含了详细信息）。

---

`xfnts_monitor_stop()` 方法将调用在 `xfnts.c` 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 */
}
```

请注意以下关于在 `svc_mon_stop()` 中调用 `scds_pmf_stop()` 函数的内容。

- `SCDS_PMF_TYPE_MON` 参数用来标识要作为故障监视器停止的程序 — 此方法也可以停止数据服务或一些其它类型的应用程序。
- `SCDS_PMF_SINGLE_INSTANCE` 参数用来将此资源标识为单实例资源。

- SIGKILL 参数标识要用来停止该资源实例的信号。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，请参阅 `scds_pmf_stop(3HA)` 手册页。
- 超时值即该资源的 `Monitor_stop_timeout` 特性的值。

---

注意 - 在退出之前，`xfnts_monitor_stop` 方法将调用 `scds_close()` 来回收通过 `scds_initialize()` 分配的资源。有关详细信息，请参阅第 119 页 “`scds_initialize()` 函数”和 `scds_close(3HA)` 手册页。

---

## xfnts\_monitor\_check 方法

每当故障监视器试图将含有资源的资源组故障切换到其它节点时，RGM 都将调用 `Monitor_check` 方法。`xfnts_monitor_check` 方法将调用 `svc_validate()` 方法来检验是否已进行了适当的配置，以支持 `xfs` 守护程序（有关详细信息，请参阅第 133 页 “`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 并不直接调用 `PROBE` 方法，而是调用 `Monitor_start` 方法来启动监视器。`xfnts_monitor_start` 方法将在 `PMF` 的控制下启动故障监视器。`xfnts_monitor_stop` 方法用来停止该故障监视器。

SUNW.xfnts 故障监视器可以执行以下操作：

- 使用为检查基于 TCP 的简单服务（例如 xfs）而特别设计的公用程序来定期监视 xfs 服务器守护程序的运行情况。
- 跟踪应用程序在时间窗口定义的时间内遇到的问题（使用 `Retry_count` 和 `Retry_interval` 特性），并在应用程序完全失败的情况下决定是重启还是故障切换数据服务。`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()` 实现故障切换或重启决定。
- 更新资源状态使其可供管理工具和图形用户界面使用。

## xfnts\_probe 主循环

`xfnts_probe` 方法实现一个循环。在实现该循环之前，`xfnts_probe` 将：

- 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` 的值作为超时值传送。在探测操作之间，该探测程序将休眠，休眠时间为 `Thorough_probe_interval` 的值。

```
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` 方法将按以下方式实现该循环。

```
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。从 svc\_probe() 返回的值被传送到 scds\_fm\_action(), 以决定是否重新启动应用程序、对资源组进行故障切换或不进行任何操作。

## svc\_probe() 函数

svc\_probe() 函数将通过调用 scds\_fm\_tcp\_connect() 建立指定端口的简单套接字连接。如果连接失败, svc\_probe() 将返回值 100, 该值表明操作完全失败。如果连接成功, 但断开连接操作失败, svc\_probe() 将返回值 50, 该值表明操作部分失败。如果连接和断开连接的操作都成功, svc\_probe() 将返回值 0, 该值表明操作成功。

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);
}

```

完成操作后，`svc_probe()` 将返回表示成功的值 (0)、表示部分失败的值 (50) 或表示完全失败的值 (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()` 中的逻辑如下：

- 保留 `Retry_interval` 特性值定义的时间之内的累积失败历史记录。
- 如果累积失败值达到 100（完全失败），将重启该数据服务。如果超出 `Retry_interval` 的值，将重置该历史记录。
- 如果重启的次数超出了 `Retry_count` 特性的值，则将在 `Retry_interval` 指定的时间内对该数据服务进行故障切换。

例如，假设该探测程序建立了 xfs 服务器的连接，但是断开连接操作失败。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。如果断开连接操作失败，将向 `scds_fm_action()` 发送表明部分失败的值 (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。

如果在下一次探测中，连接服务器再次失败，值 50 将被添加到由 `scds_fm_action()` 维护的失败历史记录中。现在累积的失败值为 100，因此 `scds_fm_action()` 将重启该数据服务。

---

## xfnts\_validate 方法

创建资源时和通过管理操作更新资源的特性或资源所包含的组的特性时，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。

仅当通过管理操作更改资源或组特性时（而不是在 RGM 设置特性时或监视器设置资源特性 Status 和 Status\_msg 时），RGM 才调用 Validate。

---

**注意** – 每当 PROBE 尝试将数据服务故障切换到新节点时，Monitor\_check 方法也将明确调用 Validate 方法。

---

RGM 通过不同于传送到其它方法的变量的附加变量（包括所更新的特性和值）调用 Validate。在 xfnts\_validate 开头部分对 scds\_initialize() 的调用可分析 RGM 传送到 xfnts\_validate 的所有变量，并将信息存储在 scds\_handle 参数中。xfnts\_validate 调用的子例行程序将使用这些信息。

xfnts\_validate 方法将调用用来检验以下方面的 svc\_validate()。

- 已经为该资源设置了 Confdir\_list 特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 */
}
```

- 通过 Confdir\_list 指定的目录中包含 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);
}
```

- 该服务器守护程序二进制文件可以在该群集节点上存取。

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
    scds_syslog(LOG_ERR,
        "Cannot access XFS binary : <%s> ", strerror(errno));
    return (1);
}
```

- Port\_list 特性指定了一个端口。

```
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
```

- 包含该数据服务的资源组中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 */
```

---

注意 - 在退出之前，xfnts\_validate 方法将调用 scds\_close() 来回收通过 scds\_initialize() 分配的资源。有关详细信息，请参阅第 119 页 “scds\_initialize() 函数”和 scds\_close(3HA) 手册页。

---

---

## xfnts\_update 方法

RGM 将调用 Update 方法通知运行资源其特性已被更改。唯一可以为 xfnts 数据服务更改的特性与故障监视器有关，因此，每当更新特性时，xfnts\_update 方法都将调用 scds\_pmf\_restart\_fm() 来重启该故障监视器。

```
* 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.");
```

---

注意 - 如果存在多个实例，scds\_pmf\_restart\_fm() 的第二个参数将唯一标识要重启的故障监视器的实例。样例中的值 0 表明该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，它们是用来自动创建要在资源组管理器 (RGM) 中运行的资源类型或数据服务的工具。资源类型本质上是应用程序的封装程序，使得该应用程序可以在由 RGM 控制的群集环境中运行。

Agent Builder 提供了一个基于屏幕的界面，用于输入关于要创建的应用程序和资源类型种类的简单信息。Agent Builder 将基于您输入的信息生成以下软件：

- 为故障切换或可伸缩资源类型（与资源类型的方法回叫相对应）生成的一组源文件 — C、Korn shell (ksh) 或 GDS（普通数据服务）
- 定制的资源类型登记 (RTR) 文件（如果您生成的是 C 或 Korn shell 源文件）
- 用来启动、停止和删除资源类型的实例（资源）的定制公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
- 包含二进制代码（如果您生成的是 C 源文件）、RTR 文件（如果您生成的是 C 或 Korn shell 源文件）和公用程序脚本的 Solaris 软件包

Agent Builder 既支持支持网络的应用程序（使用网络与客户机通信的应用程序），也支持不支持网络（独立的）应用程序。使用 Agent Builder，您还可以为具有多个独立进程树（进程监视器工具 [PMF] 必须分别对其进行监视和重新启动）的应用程序生成资源类型（请参阅第 145 页“创建具有多个独立的进程树的资源类型”）。

本章包括以下主题：

- 第 138 页“使用 Agent Builder”
- 第 147 页“目录结构”
- 第 148 页“输出”
- 第 151 页“浏览 Agent Builder”
- 第 154 页“Agent Builder 的 Cluster Agent 模块”

---

## 使用 Agent Builder

本节介绍 Agent Builder 的使用方法，包括在使用 Agent Builder 之前必须完成的准备任务。本节还介绍了如何在生成资源类型代码之后充分利用 Agent Builder。

### 分析应用程序

在使用 Agent Builder 之前，您必须确定您的应用程序是否满足使其具有高可用性或伸缩性的条件。Agent Builder 无法执行仅基于应用程序的运行时特性的分析。

第 25 页“分析应用程序的适用性”介绍了关于本主题的详细信息。

虽然在大多数情况下 Agent Builder 会至少提供一个部分解决方案，但是它并不是始终能够为您的应用程序创建完整的资源类型。例如，较复杂的应用程序可能需要使用 Agent Builder 在缺省情况下不会生成的附加代码，例如为附加特性添加验证检查的代码或调整 Agent Builder 未提供的参数的代码。在这些情况下，您必须更改所生成的源代码或 RTR 文件。Agent Builder 的设计目的就是为您提供这种灵活性。

在生成的源代码中，Agent Builder 在某些位置放置了注释，您可以在这些位置添加自己的特定资源类型代码。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。

即使在编写整个资源类型代码时没有使用任何 Agent Builder 生成的代码，您也可以利用 Agent Builder 提供的 make 程序的描述文件和结构来创建您的资源类型的 Solaris 软件包。

### 安装和配置 Agent Builder

Agent Builder 不需要专门进行安装。Agent Builder 包含在 SUNWscdev 软件包中，缺省情况下在标准 Sun Cluster 软件安装过程中会安装该软件包（《Sun Cluster 软件安装指南（适用于 Solaris OS）》中包含有详细信息）。使用 Agent Builder 之前，请检验以下信息：

- Java 是否包含在 \$PATH 变量中。Agent Builder 依赖于 Java（Java 开发工具 1.3.1 版或更高版本），如果 \$PATH 中未包含 Java，则 scdsbuilder 将返回一条错误消息。
- 是否已安装了 Solaris 8 或更高版本的“开发者系统支持”软件组。
- cc 编译器是否包含在 \$PATH 变量中。Agent Builder 将使用 \$PATH 变量中 cc 的第一个引用来标识用来生成该资源类型的 C 二进制代码的编译器。如果 \$PATH 中未包含 cc，则 Agent Builder 将禁止生成 C 代码的选项（请参阅第 140 页“使用“创建”屏幕”）。

---

注意 – 除了标准 `cc` 编译器之外，您可以将其它编译器与 Agent Builder 一起使用。执行此操作的方法之一是在 `$PATH` 中创建从 `cc` 到其它编译器（例如 `gcc`）的符号链接。另一个方法是将 `make` 程序的描述文件中的编译器定义（当前为 `CC=cc`）更改为其它编译器的完整路径。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`。在这种情况下，您不能直接运行 Agent Builder，而是必须使用 `make` 和 `make pkg` 命令生成数据服务代码和软件包。

---

## 启动 Agent Builder

输入以下命令启动 Agent Builder：

```
% /usr/cluster/bin/scdsbuilder
```

将显示初始的 Sun Builder 屏幕，如下图所示。

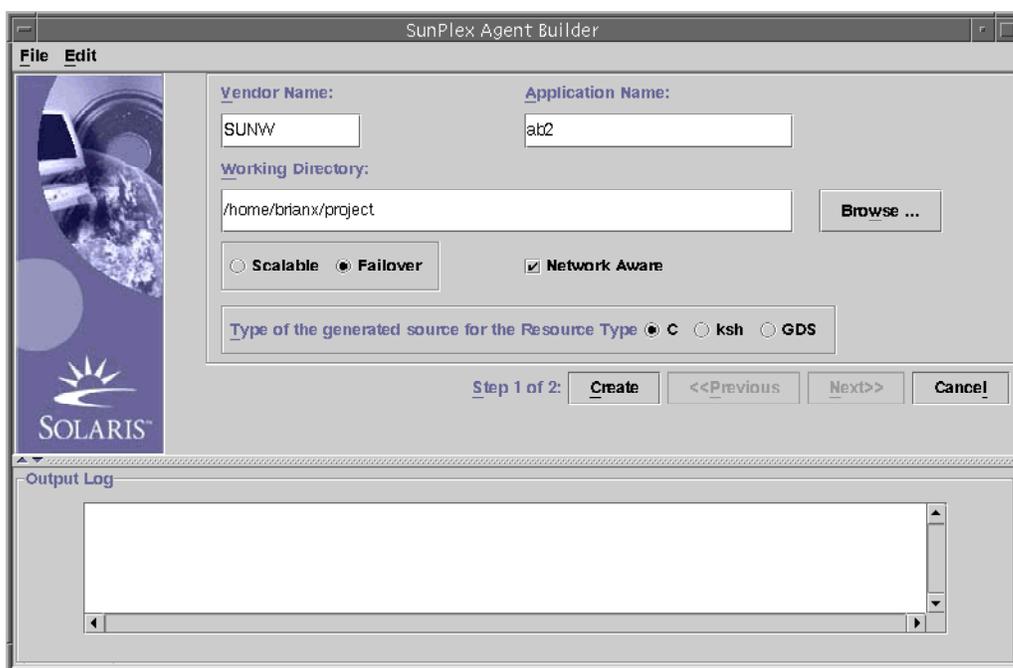


图 9-1 初始屏幕

---

注意 – 如果无法存取 GUI 版本，则可以通过命令行界面存取 Agent Builder（请参阅第 147 页“使用 Agent Builder 的命令行版本”）。

---

Agent Builder 提供了两个屏幕，用来指导您完成创建新资源类型的过程：

1. **创建** — 在此屏幕上，您需要提供关于要创建的资源类型的基本信息，例如其名称和所生成文件的工作目录（即您创建并配置资源类型模板的目录）。您还需确定要创建的资源的种类（“可伸缩”或“故障切换”）、基本应用程序是否支持网络（即该应用程序是否使用网络与其客户机通信）以及要生成的代码的类型（C、ksh 或 GDS）。有关 GDS（普通数据服务）的信息，请参阅第 10 章。您必须完成此屏幕上的信息并选择“创建”来生成相应的输出后才会显示“配置”屏幕。
2. **配置** — 在此屏幕上，您需要提供一条启动应用程序的命令。您还可以提供停止或探测应用程序的命令（可选操作）。如果您不指定这些命令，生成的输出将使用信号停止应用程序并提供缺省探测机制（请参阅第 142 页“使用“配置”屏幕”中对探测命令的说明）。此屏幕还使您可以分别更改这三个命令的超时值。

---

**注意** – 如果是从现有资源类型的工作目录启动 Agent Builder，则“创建”和“配置”屏幕将使用现有资源类型的值作为初始值。

---

如果您对如何使用 Agent Builder 各个屏幕上的任意按钮或菜单命令存在疑问，请参阅第 151 页“浏览 Agent Builder”。

## 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。下图显示的是已在各个字段输入信息的“创建”屏幕。

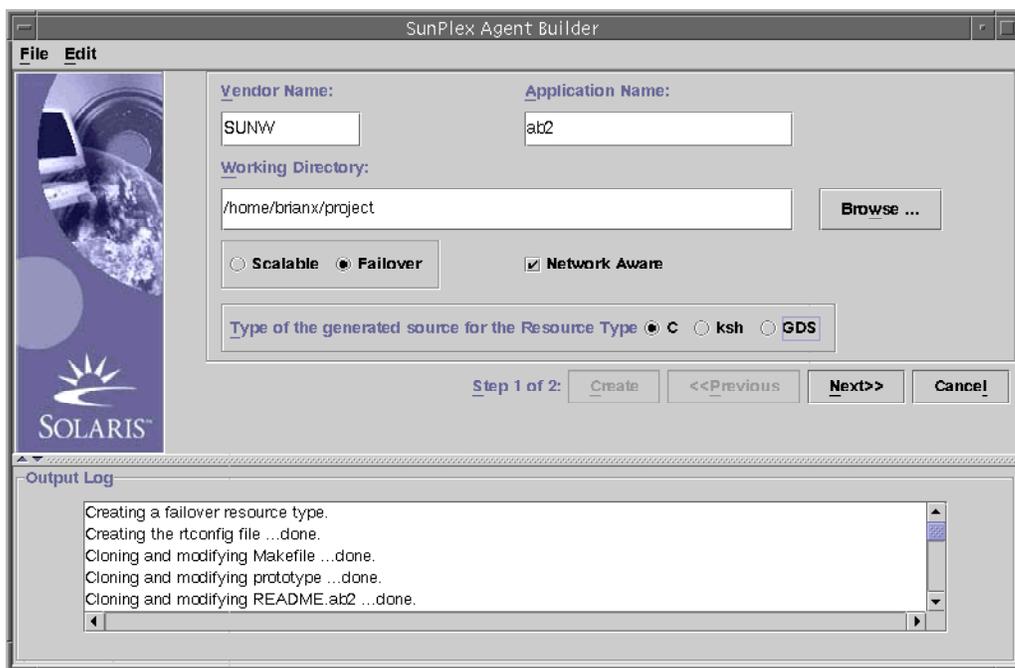


图 9-2 “创建”屏幕

“创建”屏幕中包含以下字段、单选按钮和复选框：

- **供应商名称** — 标识资源类型的供应商的名称。通常要指定供应商的股票标志，但是任何可唯一标识供应商的名称也有效。仅可使用字母数字字符。
- **应用程序名称** — 资源类型的名称。仅可使用字母数字字符。

---

**注意** – 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。全名不能超过九个字符。

---

- **工作目录** — Agent Builder 在此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 将从中启动 Agent Builder 的目录的路径作为此字段的初始值，您也可以键入其它名称或使用“浏览”按钮定位其它目录。

在工作目录下，Agent Builder 将使用资源类型的名称创建一个子目录。例如，如果供应商的名称为 SUNW，应用程序的名称为 ftp，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标资源类型的所有目录和文件（请参阅第 147 页“目录结构”）。

- **“可伸缩”或“故障切换”** — 指定目标资源类型是可伸缩类型还是故障切换类型。

- **支持网络** — 指定基本应用程序是否支持网络；即，它是否使用网络与客户进行通信。选中该框表示指定支持网络；否则表示指定不支持网络。Korn shell 代码要求应用程序必须是支持网络的。因此，如果您选中 **ksh** 或 **GDS** 按钮，则 Agent Builder 将复选该框并将其灰显。
- **C、ksh** — 指定生成的源代码的语言。尽管这些选项是互斥的，但是借助 Agent Builder 您可以先使用 ksh 类型的生成代码创建资源类型，然后重复使用相同信息创建 C 类型的生成代码（请参阅第 145 页“克隆现有的资源类型”）。
- **GDS** — 指定此服务为普通数据服务。有关创建和配置普通数据服务的信息，请参阅第 10 章。

---

**注意** – 如果 `$PATH` 中未包含 `cc` 编译器，则 Agent Builder 将灰显“C”选项按钮并在 **ksh** 按钮中加入一个选中标记。要指定其它编译器，请参阅第 138 页“安装和配置 Agent Builder”结尾处的说明。

---

输入必需信息后，请单击“**创建**”按钮。屏幕底部的“输出记录”用来显示 Agent Builder 正在执行的操作。您可以使用“编辑”菜单中的“**保存输出记录**”命令来保存输出记录中的信息。

完成后，Agent Builder 将显示一条成功消息或一条警告消息。警告消息表明 Agent Builder 无法完成此步骤，您应该检查输出记录获取详细信息。

如果 Agent Builder 成功地完成了操作，您可以单击“**下一步**”按钮进入“配置”屏幕，使用该屏幕您可以完成生成资源类型的过程。

---

**注意** – 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，您已输入的信息或 Agent Builder 已完成的工作并不会丢失（请参阅第 145 页“重复使用完成的工作”）。

---

## 使用“配置”屏幕

Agent Builder 完成创建资源类型的工作后，如果您选择“创建”屏幕上的“**下一步**”按钮，将会显示“配置”屏幕，如下图所示。必须先创建资源类型才能存取“配置”屏幕。

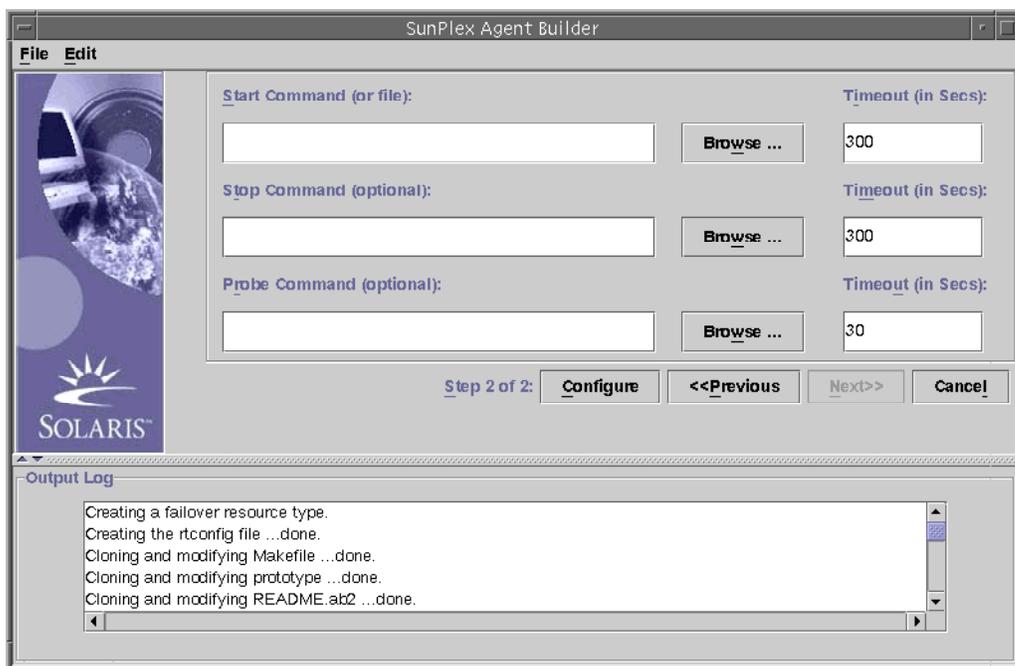


图 9-3 “配置”屏幕

“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。您必须指定此命令。您可以在提供的字段中键入该命令，也可以使用“浏览”按钮定位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必需的所有内容，例如主机名、端口号、指向配置文件的路径等。如果应用程序需要在命令行上指定一个主机名，则可以使用 Agent Builder 定义的 \$hostnames 变量（请参阅第 144 页“使用 Agent Builder \$hostnames 变量”）。

请不要将命令置于双引号 (“”) 中。

---

**注意** - 如果基本应用程序具有多个独立的进程树（每一个进程都由它自己的、由 PMF 控制的标记来启动），则您不能只指定一个命令，而是必须创建一个文本文件，其中包含启动每一个进程树的各个命令，并且还要在“启动命令”文本字段中指定指向此文件的路径。请参阅第 145 页“创建具有多个独立的进程树的资源类型”，其中列出了此文件正常工作所需的一些特殊特性。

---

- **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”按钮定位包含用来停止应用程序的命令的文件。如果应用程序需要在命令行上指定一个主机名，则可以使用 Agent

Builder 定义的 `$hostnames` 变量（请参阅第 144 页“使用 Agent Builder `$hostnames` 变量”）。

此命令是可选的。如果您不指定停止命令，则生成的代码将使用信号（位于 `Stop` 方法中）停止该应用程序，如下所示。

- `Stop` 方法发送 `SIGTERM` 以停止应用程序，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` 也未成功，则 `Stop` 方法将在未成功的状态下退出（超时值的剩余 5% 视为额外开销）。



---

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相应的退出状态值（介于 0 [成功] 和 100 [完全失败] 之间）。此命令是可选的。您可以键入指向该命令的完整路径，也可以使用“浏览”按钮定位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。如果您不指定探测命令，则生成的代码仅连接到资源所用的端口（或仅从该端口上断开连接）。如果探测成功，则表明该应用程序运行正常。探测命令仅适用于支持网络的应用程序。Agent Builder 通常会生成探测命令，但是对于不支持网络的应用程序，Agent Builder 将禁止该探测命令。

如果应用程序需要在探测命令行上指定一个主机名，则可以使用 Agent Builder 定义的 `$hostnames` 变量（请参阅第 144 页“使用 Agent Builder `$hostnames` 变量”）。

- **超时** —（针对每个命令）— 每个命令的超时值（以秒为单位）。您可以指定新值或接受 Agent Builder 提供的缺省值（启动和停止命令的缺省超时值为 300 秒，探测命令的缺省超时值为 30 秒）。

## 使用 Agent Builder `$hostnames` 变量

对于很多应用程序（特别是支持网络的应用程序），必须在命令行上将主机（在该主机上应用程序进行监听并处理服务用户请求）的名称传送给应用程序。因此，在大多数情况下，必须为目标资源类型的启动、停止和探测命令指定主机名参数（在“配置”屏幕上）。但是，应用程序进行监听的主机的名称是群集特有的 — 该主机名是资源在群集上运行时确定的，而不能在 Agent Builder 生成资源类型代码时确定。

要解决这个问题，Agent Builder 提供了您可以在启动、停止和探测命令的命令行中指定的 `$hostnames` 变量。指定 `$hostnames` 变量时就像指定实际主机名一样，例如：

```
/opt/network_aware/echo_server -pport_no-1 $hostnames
```

当目标资源类型的资源在群集上运行时，为该资源配置的 `LogicalHostname` 或 `SharedAddress` 主机名（位于资源的 `Network_resources_used` 资源特性中）将替代 `$hostnames` 变量的值。

如果您配置的是具有多个主机名的 `Network_resources_used` 特性，则 `$hostnames` 变量中将包含所有主机名（用逗号隔开）。

## 创建具有多个独立的进程树的资源类型

`Agent Builder` 可以创建具有多个独立进程树的应用程序的资源类型。从 PMF 分别监视并启动这些进程树这一方面来说，这些进程树是独立的。PMF 用进程树自身的标记启动每一个进程树。

---

**注意** – 仅当您指定生成的资源代码为 C 时，才能通过 `Agent Builder` 创建具有多个独立进程树的资源类型。当指定的代码为 `ksh` 或 `GDS` 时，您无法使用 `Agent Builder` 创建这些资源类型。要在已指定 `ksh` 或 `GDS` 的情况下创建这些资源类型，您必需以手动的方式编写代码。

---

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。此文件中不能包含任何空行。您需要在“配置”屏幕上的“启动命令”文本字段中指定此文本文件。

确保此文件没有执行权限以使 `Agent Builder` 能够区分此文件，其目的是从包含多个命令的简单可执行脚本中启动多个进程树。如果此文本文件被赋予了执行权限，则资源将在群集上正常启用，但是所有的命令都将在一个 PMF 标记下启动，这样就无法由 PMF 分别监视和重新启动进程树。

## 重复使用完成的工作

使用 `Agent Builder`，您可以通过多种方式利用已完成的工作。

- 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，然后再重新编译该代码来创建新的软件包。

## 克隆现有的资源类型

请按照以下步骤克隆已由 `Agent Builder` 生成的现有资源类型。

1. 将现有资源类型装入 `Agent Builder`。可以使用以下两种方法之一来执行此操作：
  - a. 从现有资源类型（使用 `Agent Builder` 创建）的工作目录（包含有 `rtconfig` 文件）启动 `Agent Builder`，那么在“创建”屏幕和“配置”屏幕上，`Agent Builder` 将装入该资源类型的值。

b. 使用“文件”菜单中的“装入资源类型”命令。

2. 在“创建”屏幕上更改工作目录。

您必须使用“浏览”按钮选择目录，不能键入新目录的名称。选择目录后，Agent Builder 将重新启用“创建”按钮。

3. 进行更改。

您可以使用此步骤来更改该资源类型的生成代码的类型。例如，如果您原来创建的是 ksh 版本的资源类型，但是过了一段时间后发现需要创建 C 版本的资源类型，则您可以装入现有的 ksh 资源类型，将输出更改为 C，然后用 Agent Builder 创建 C 版本的资源类型。

4. 创建克隆的资源类型。

选择“创建”以创建该资源类型。选择“下一步”进入“配置”屏幕。选择“配置”以配置该资源类型，然后选择“取消”完成操作。

## 编辑已生成的源代码

为使创建资源类型的过程保持简单，Agent Builder 将限制输出数量，这必将限定已生成的资源类型的范围。因此，为了添加更复杂的特征（例如附加特性的验证检查）或调整 Agent Builder 未提供的参数，您需要修改已生成的源代码或 RTR 文件。

源文件位于 *install\_directory/rt\_name/src* 目录中。在源代码中，Agent Builder 在您可以添加代码的位置嵌入了注释。这些注释的格式如下（针对 C 代码）：

```
/* 用户添加的代码 -- 开始 vvvvvvvvvvvvvvvvvv */  
/* 用户添加的代码 -- 结束 ^^^^^^^^^^^^^^^^^^ */
```

---

注意 – 在 Korn shell 代码中，注释的格式与以上格式完全相同，只是在命令行的开头使用的是井号 (#)。

---

例如，*rt\_name.h* 用来声明其它程序使用的所有公用程序例程。声明列表的结尾处是注释，这些注释使得您可以声明已向任意代码添加的附加例程。

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 文件的详细信息，请参阅第 29 页“设置资源和资源类型特性”；有关特性的信息，请参阅附录 A）。

## 使用 Agent Builder 的命令行版本

与图形用户界面版本相同，Agent Builder 的命令行版本也采用两步式过程。不过，您要将参数传送到 `scdscreate(1HA)` 和 `scdsconfig(1HA)` 命令，不是在图形用户界面中输入信息。

按照以下步骤使用 Agent Builder 的命令行版本：

1. 使用 `scdscreate` 创建 Sun Cluster 资源类型模板，以使应用程序具有高可用性 (HA) 或可伸缩性。
2. 使用 `scdsconfig` 配置您用 `scdscreate` 创建的资源类型模板。
3. 将目录更改为工作目录中的 `pkg` 子目录。
4. 使用 `pkgadd(1M)` 命令安装您使用 `scdscreate` 创建的软件包。
5. 如果需要，可以编辑已生成的源代码。
6. 运行启动脚本。

---

## 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。您需指定（在“创建”屏幕上）工作目录。您必须为您开发的其它资源类型分别指定安装目录。在工作目录下，Agent Builder 将创建一个子目录，其名称是将供应商名称和资源类型名称（在“创建”屏幕中指定）顺序连接而成的。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 将创建并总装下表中列出的目录。

目录名称	目录
<code>bin</code>	对于 C 输出，包含从源文件编译的二进制文件。对于 <code>ksh</code> 输出，所包含的文件与 <code>src</code> 目录中的文件相同。
<code>etc</code>	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，以形成 RTR 文件名称。例如，如果供应商的名称为 <code>SUNW</code> ，资源类型的名称为 <code>ftp</code> ，则 RTR 文件的名称为 <code>SUNW.ftp</code> 。

目录名称	目录
man	包含用于 start、stop 和 remove 公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，请使用 man -M 选项指定路径。例如 man -M <i>install_directory</i> /SUNWftp/man removeftp。
pkg	包含最终软件包。
src	包含 Agent Builder 生成的源文件。
util	包含 Agent Builder 生成的 start、stop 和 remove 公用程序脚本。请参阅第 149 页“公用程序脚本和手册页”。Agent Builder 将应用程序名称附加到各个脚本名称上；例如，startftp、stopftp 和 removeftp。

## 输出

本节介绍 Agent Builder 生成的输出。

### 源文件和二进制文件

用来管理资源组以及最终管理群集上的资源的资源组管理器 (RGM) 适用于回叫模型。当发生特定事件时，例如某个节点发生故障，RGM 将为在受影响节点上运行的每一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。（有关此模型的详细信息，请参阅第 18 页“RGM 模型”、第 20 页“回叫方法”和 `rt_callbacks(1HA)` 手册页）。

为支持此模型，Agent Builder 将在 `install_directory/rt_name/bin` 目录中生成用作回叫方法的八个可执行程序 (C) 或脚本 (ksh)。

---

**注意** – 严格地说，用来实现缺省监视器的 `rt_name_probe` 程序不是回叫程序。RGM 并不直接调用 `rt_name_probe`，而是调用 `rt_name_monitor_start` 和 `rt_name_monitor_stop`，它们调用 `rt_name_probe` 来启动和停止缺省监视器。

---

Agent Builder 生成的八个方法为：

- `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`

有关以上每一个方法的详细信息，请参阅 `rt_callbacks(1HA)` 手册页。

在 `install_directory/rt_name/src` 目录中（C 输出），Agent Builder 将生成以下文件：

- 头文件 (`rt_name.h`)。
- 包含所有方法通用的代码的源文件 (`rt_name.c`)。
- 通用代码的对象文件 (`rt_name.o`)。
- 每个方法的源文件 (`*.c`)。
- 每个方法的对象文件 (`*.o`)。

Agent Builder 将 `rt_name.o` 文件链接到每个方法的 `.o` 文件，以在 `install_directory/rt_name/bin` 目录中创建可执行文件。

对于 ksh 输出，`install_directory/rt_name/bin` 和 `install_directory/rt_name/src` 目录是一样的 — 每个都包含八个可执行脚本（对应于七个回叫方法和一个 PROBE 方法）。

---

注意 - ksh 输出包含有两个已编译的公用程序（`gettime` 和 `gethostnames`），某些回叫方法需要使用这些公用程序来获取时间并进行探测。

---

您可以编辑源代码并运行 `make` 命令来重新编译代码，完成时，可以运行 `make pkg` 命令生成新的软件包。为支持对源代码进行修改，Agent Builder 在源代码中添加代码的相应位置嵌入了注释。请参阅第 146 页“编辑已生成的源代码”。

## 公用程序脚本和手册页

生成资源类型并将其软件包安装在群集上之后，您还必须获取该资源类型在群集上运行的实例（资源），通常方法是使用管理命令或 SunPlex Manager。但是，为了方便，Agent Builder 将生成用于获取实例的定制公用程序脚本（启动脚本）以及用来停止和删除目标资源类型的资源的脚本。这三个脚本位于 `install_directory/rt_name/util` 目录中，用于执行以下操作：

- **启动脚本** — 用来登记资源类型和创建必要的资源组和资源。该脚本还可用来创建使应用程序可以与网络上的客户机进行通信的网络地址资源（`LogicalHostname` 或 `SharedAddress`）。
- **停止脚本** — 用来停止和禁止资源。
- **删除脚本** — 用来撤消启动脚本进行的工作，即它将停止和删除系统中的资源、资源组和目标资源类型。

---

注意 - 删除脚本仅适用于通过相应的启动脚本启动的资源，因为这些脚本使用内部惯例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，如果应用程序名称为 ftp，则这些脚本名称分别为 startftp、stopftp 和 removeftp。

Agent Builder 在 *install\_directory/rt\_name/man/man1m* 目录中提供了每个公用程序脚本的手册页。在启动这些脚本之前您应该首先阅读这些手册页，因为其中记录了需要传送到脚本的参数。

要查看这些手册页，请使用 -M 选项和 man 命令指定指向此手册目录的路径。例如，如果 SUNW 为供应商名称，ftp 为应用程序名称，则请使用以下命令查看 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。当 Agent Builder 生成的软件包安装到群集上之后，该公用程序脚本的手册页将被放置在 */opt/rt\_name/man* 目录中。例如，使用以下命令查看 startftp(1M) 手册页：

```
man -M /opt/SUNWftp/man startftp
```

## 支持文件

Agent Builder 将支持文件（例如 pkginfo、postinstall、postremove 和 preremove）放置在 *install\_directory/rt\_name/etc* 目录中。此目录中还包含资源类型登记 (RTR) 文件，当通过群集进行登记时，该文件用来声明目标资源类型可用的资源和资源类型特性并初始化特性值（有关详细信息，请参阅第 29 页“设置资源和资源类型特性”）。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 命令重新生成软件包。

## 软件包目录

*install\_directory/rt\_name/pkg* 目录中包含 Solaris 软件包。该软件包的名称是由供应商名称和应用程序名称顺序连接而成的，例如 SUNWftp。*install\_directory/rt\_name/src* 目录中的 Makefile 支持新软件包的创建。例如，如果您要更改源文件并重新编译代码，或要更改软件包公用程序脚本，则需使用 make pkg 命令创建新的软件包。

当您从群集中删除软件包时，如果尝试从多个节点同时运行 pkgrm 命令，则该命令可能会失败。您可以采用以下两种方法之一来解决此问题：

- 从任意节点运行 pkgrm 之前，请先从群集的一个节点上运行 removert\_name 脚本。

- 先从群集的一个节点上运行 `pkgrm`（这可进行必要的清除工作），然后再从剩余节点运行 `pkgrm`（如果需要可同时运行）该命令。

若您尝试从多个节点同时运行 `pkgrm` 而导致该命令失败，则请先从一个节点再次运行该命令，然后再从剩余节点运行该命令。

## rtconfig 文件

如果您要生成 C 或 ksh 源代码，则 Agent Builder 将在工作目录中生成一个配置文件，包含您在“创建”屏幕和“配置”屏幕上所输入的信息的 `rtconfig`。如果您从现有资源类型的工作目录启动 Agent Builder（或使用“文件”菜单的“装入资源类型”命令装入现有资源类型），Agent Builder 将读取 `rtconfig` 文件并使用您所提供的现有资源类型的信息填充“创建”和“配置”屏幕上的各项内容。当您克隆现有资源类型时，此功能非常有用（请参阅第 145 页“克隆现有的资源类型”）。

---

## 浏览 Agent Builder

浏览 Agent Builder 的操作简单且直观。Agent Builder 是一种两步式向导，每一个步骤对应一个屏幕（“创建”和“配置”屏幕）。通过以下方法在每个屏幕上输入信息：

- 在字段中键入信息。
- 浏览目录结构并选择文件或目录。
- 选中一组互斥的单选按钮之一，例如“可伸缩”或“故障切换”。
- 选中一个打开/关闭框。例如，选中 **Network Aware** 将基本应用程序标识为支持网络，而不选中此框则标识为不支持网络。

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 将相应地突出显示或灰显这些按钮。

例如，完成“创建”屏幕上的各个字段并选中所需选项后，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，而且在您完成此操作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显示进程消息。当 Agent Builder 完成操作后，它将相应地显示成功消息或提醒您查看输出记录的警告消息。“下一步”按钮将被突出显示状；如果这是最后一步，则只有“取消”按钮突出显示。

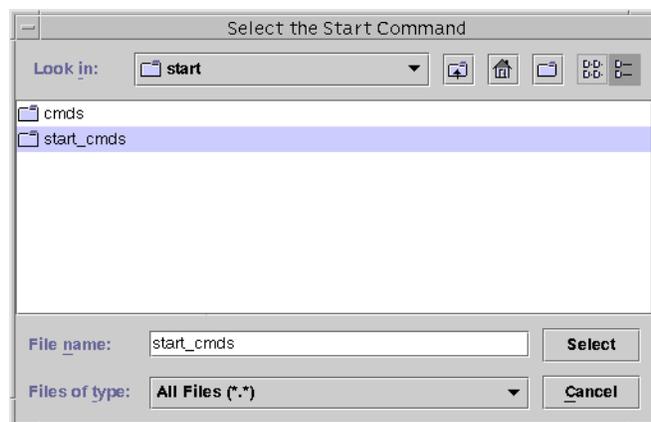
您可以随时选择“取消”退出 Agent Builder。

## “浏览”按钮

对于一些特定的 Agent Builder 字段，您既可以键入信息，也可以单击“浏览”按钮来浏览目录结构并选择文件或目录。



当您单击“浏览”时，将显示一个屏幕，如下所示：



双击某个文件夹以打开该文件夹。如果突出显示某个文件，那么其名称将显示在“文件名”框中。当您找到并突出显示相应的文件后，请单击“选择”。

---

**注意** – 如果您要浏览某个目录，请突出显示该目录并选择“打开”按钮。如果该目录没有子目录，则 Agent Builder 将关闭该浏览窗口并将您所突出显示的目录的名称放置在相应字段中。如果此目录具有子目录，请单击“关闭”按钮关闭该浏览窗口并返回上一屏。Agent Builder 将把您所突出显示的目录的名称放置在相应字段中。

---

位于屏幕右上角的图标用来执行以下操作：



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。

## 菜单

Agent Builder 提供了“文件”和“编辑”菜单。

### “文件”菜单

“文件”菜单中有两个命令：

- **装入资源类型** — 用来装入现有资源类型。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。如果从中启动 Agent Builder 的目录中存在某个资源类型，则 Agent Builder 将自动装入该资源类型。“装入资源类型”命令使您可以从任意目录启动 Agent Builder 并选择现有资源类型作为创建新资源类型的模板（请参阅第 145 页“克隆现有的资源类型”）。
- **退出** — 用来退出 Agent Builder。您也可以单击“创建”或“配置”屏幕上的“取消”退出 Agent Builder。

### “编辑”菜单

“编辑”菜单中具有用来清除和保存输出记录的命令：

- **清除输出记录** — 用来清除输出记录中的信息。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。如果您是在执行这样的一个循环操作：更改源代码然后在 Agent Builder 中重新生成输出，并且希望分开各个状态消息，则可以在每次使用之前保存并清除日志文件。
- **保存记录文件** — 用来将记录输出保存到文件中。Agent Builder 提供了浏览屏幕，用于选择目录并指定文件名。

---

## Agent Builder 的 Cluster Agent 模块

Agent Builder 的 Cluster Agent 模块是一个 NetBeans™ 模块。使用 Cluster Agent 模块，Sun Java Studio（原 Sun ONE Studio）产品的用户可以通过集成的开发环境为 Sun Cluster 软件创建资源类型或数据服务。Cluster Agent 模块提供了一种基于屏幕的界面，用来说明要创建的资源类型的种类。

---

注意 – Sun Java Studio documentation 包含有关如何设置、安装和使用 Sun Java Studio 产品的信息。

---

### ▼ 安装和设置 Cluster Agent 模块

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` 放置在 `/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。

---

注意 – 希望运行 Cluster Agent 模块的系统上必须安装或可以使用 Sun Cluster 和 Sun Java Studio 产品以及 Java™ 1.4。

---

#### 1. 您是希望使所有用户都可以使用 Cluster Agent 模块，还是希望只有您自己可以使用该模块？

- 要使所有用户都可以使用该模块，您需要成为超级用户或相当的身份，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
```

---

注意 – 如果已将 Sun Java Studio 软件安装在 `/opt/s1studio/ee` 之外的目录，请用您所用的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。

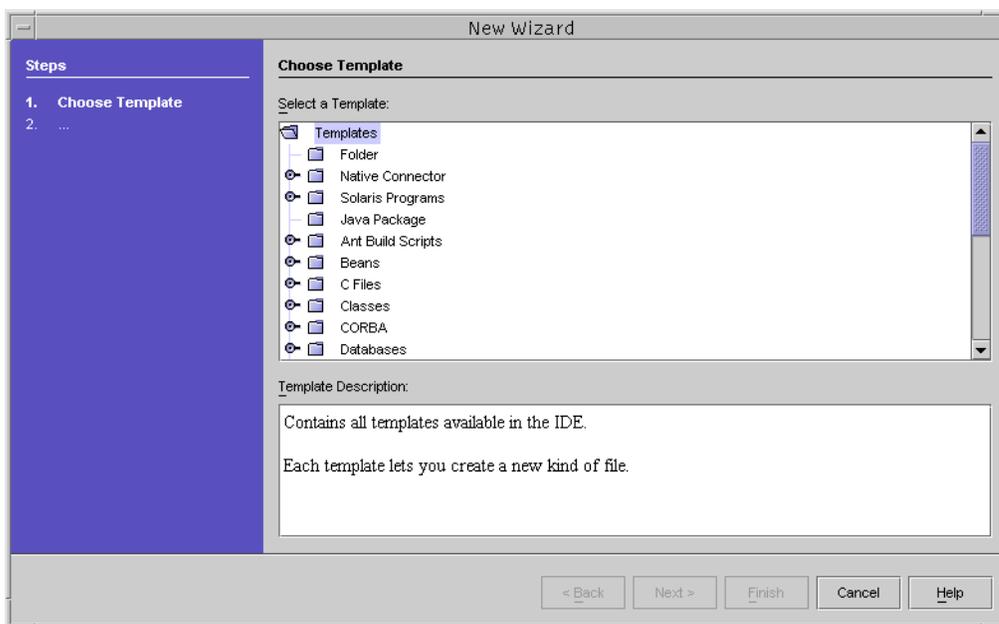
## ▼ 启动 Cluster Agent 模块

以下各个步骤介绍了如何从 Sun Java Studio 软件启动 Cluster Agent 模块。

1. 从 Sun Java Studio“文件”菜单中选择“新建”，或单击工具栏上的图标：



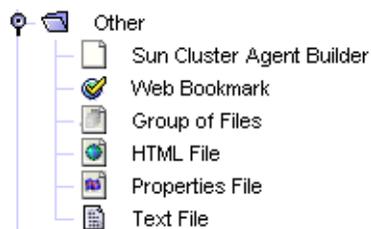
将显示“新建向导”屏幕。



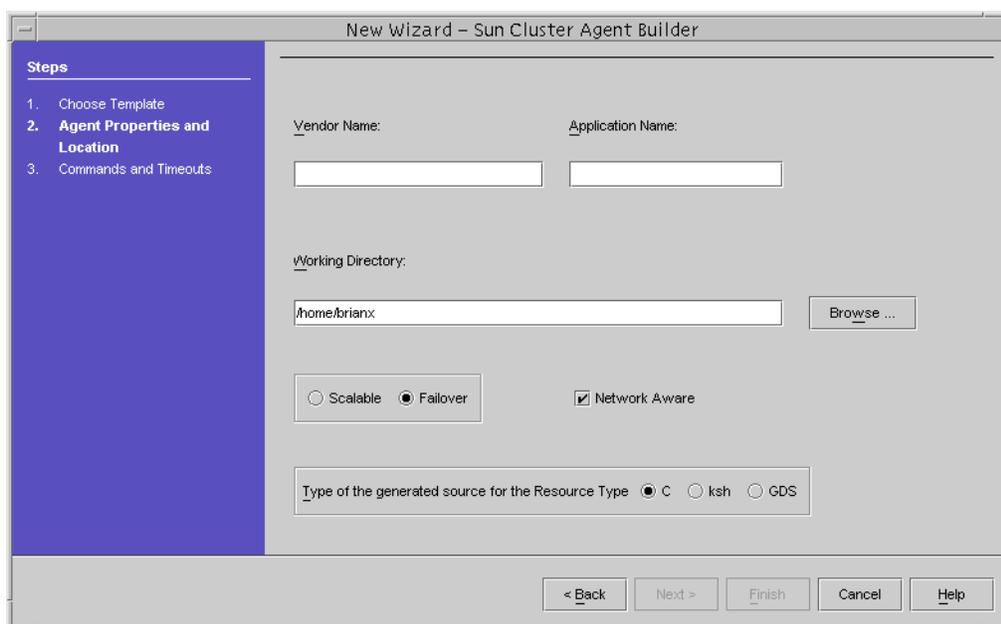
2. 在“选择模板”窗口中，向下滚动（如果有必要）并单击“其它”文件夹旁边的钥匙符号。



将打开“其它”菜单。



3. 从“其它”菜单中选择“Sun Cluster Agent Builder”并单击“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。

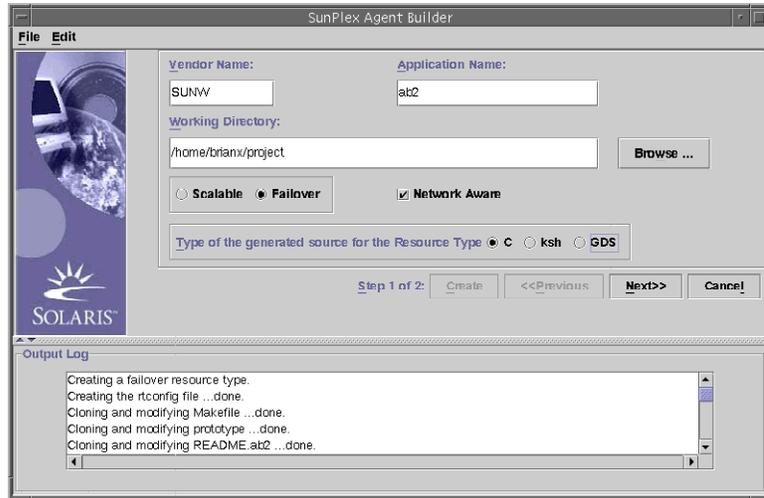


图 9-4 Agent Builder 软件中的“创建”屏幕

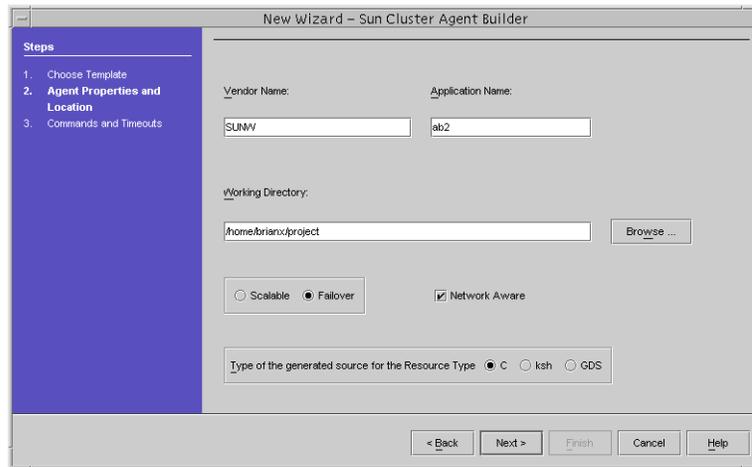


图 9-5 Cluster Agent 模块中的“新建向导 - Sun Cluster Agent Builder”屏幕

## Cluster Agent 模块和 Agent Builder 之间的区别

虽然 Cluster Agent 模块和 Agent Builder 之间很相似，但是两者之间仍然存在微小的差别：

- 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。  
而在 Agent Builder 中，当您单击位于“创建”屏幕上的“创建”之后将立即创建该资源类型；单击“配置”屏幕上的“配置”后，配置工作也将立即进行。
- 在 Sun Java Studio 产品中，显示在 Agent Builder 的“输出记录”窗口中的信息将显示在一个单独的输出窗口中。

## 第 10 章

# 普通数据服务

---

本章提供了有关普通数据服务 (GDS) 的信息，并说明了如何利用 SunPlex Agent Builder 或者标准的 Sun Cluster 管理命令创建一个使用 GDS 的服务。

- 第 159 页 “GDS 概述”
- 第 164 页 “使用 SunPlex Agent Builder 创建使用 GDS 的服务”
- 第 168 页 “使用标准的 Sun Cluster 管理命令来创建使用 GDS 的服务”
- 第 170 页 “SunPlex Agent Builder 的命令行界面”

---

## GDS 概述

GDS 是一种机制，它将简单的、能支持网络的应用程序插入 Sun Cluster 资源组管理框架中，从而使它们具有高可用性或可伸缩性。此机制不需要为代理编写代码，而编写代码是使应用程序具有高可用性或可伸缩性的通常做法。

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、回叫方法 (rt\_callbacks (1HA)) 实现和资源类型登记文件 (rt\_reg(4))。

## 预编译的资源类型

普通数据服务资源类型 SUNW.gds 包含在 SUNWscgds 软件包中。scinstall(1M) 公用程序将在群集安装过程中安装该软件包。SUNWscgds 软件包中包含以下文件：

```
# 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 的原因

通过使用 SunPlex Agent Builder 生成的源代码模型（请参阅 `scdscreate(1HA)`）或标准的 Sun Cluster 管理命令，GDS 具有以下优点：

- GDS 易于使用。
- GDS 及其方法是预编译的，因而无法进行更改。
- 使用 SunPlex Agent Builder 可为应用程序生成驱动脚本，而且这些脚本全部放在一个可在 Solaris 中安装的软件包中。该软件包可在多个群集中重复使用。

## 创建使用 GDS 的服务的方法

创建使用 GDS 的服务有两种方法：

- 使用 SunPlex Agent Builder
- 使用标准的 Sun Cluster 管理命令

## GDS 和 SunPlex Agent Builder

使用 SunPlex Agent Builder 并选择 GDS 作为生成的源代码的类型。使用用户所输入的内容来生成一组驱动脚本，而这些脚本将为给定的应用程序配置资源。

## GDS 和标准的 Sun Cluster 管理命令

此方法使用 `SUNWscgds` 中预编译的数据服务代码，但要求系统管理员使用标准的 Sun Cluster 管理命令（`scrgadm(1M)` 和 `scswitch(1M)`）创建和配置资源。

## 选择创建基于 GDS 的服务时所应使用的方法

正如第 168 页“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 169 页“标准的 Sun Cluster 管理命令（用于创建使用 GDS 的、可伸缩的服务）”等过程所示，需要在键盘上输入大量的信息才能发出有关的 `scrgadm` 命令和 `scswitch` 命令。

如果将 GDS 和 SunPlex Agent Builder 结合使用，则可简化此过程，因为它会产生驱动脚本，而这些驱动脚本可发出 `scrgadm` 命令和 `scswitch` 命令。

## 何时不宜使用 GDS 机制

虽然使用 GDS 有很多优点，但还是存在不宜使用 GDS 机制的情况。不宜使用 GDS 机制的情况有：

- 使用预编译的资源类型时所需的控制多于可用控制，例如，需要增加扩展特性或更改缺省特性时。
- 需要修改源代码以增加特殊的功能。
- 您想要使用多个进程树时。
- 您想要使用不支持网络的应用程序时。

## 所需的 GDS 特性

必须提供以下特性：

- Start\_command (扩展特性)
- Port\_list

## Start\_command 扩展特性

Start\_command 扩展特性中指定的 start 命令可启动应用程序。它必须是一条具有完整变量的 UNIX 命令，可以直接传给某个 shell 来启动应用程序。

## Port\_list 特性

Port\_list 特性可标识应用程序所侦听的端口列表。Port\_list 特性必须根据由 SunPlex Agent Builder 创建的启动脚本而指定，或是根据 scrgadm 命令而指定（如果您使用的是标准的 Sun Cluster 管理命令）。

## 可选的 GDS 特性

可选的 GDS 特性包括：

- Network\_resources\_used
- Stop\_command (扩展特性)
- Probe\_command (扩展特性)
- Start\_timeout
- Stop\_timeout
- Probe\_timeout (扩展特性)
- Child\_mon\_level (扩展特性，只用于标准的管理命令)
- Failover\_enabled (扩展特性)
- Stop\_signal (扩展特性)

## Network\_resources\_used 特性

此特性的缺省值为空。如果应用程序需要绑定到一个或多个特定的地址，则必须指定此特性。如果忽略此特性或者将其指定为 Null，则应用程序将会被认为是侦听所有地址。

在创建 GDS 资源之前，必须已经配置了一种 LogicalHostname 资源或 SharedAddress 资源。有关如何配置 LogicalHostname 或 SharedAddress 资源的信息，请参阅《Sun Cluster 数据服务规划和管理指南（适用于 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 或者包含一个或多个 SharedAddress。有关详细信息，请参阅 r\_properties (5)。

## Stop\_command 特性

stop 命令必须停止应用程序并仅在应用程序完全停止后返回。它必须是一条完整的 UNIX 命令，可以直接传给某个 shell 来停止应用程序。

如果提供了 Stop\_command，则 GDS 的 stop 方法将使用 80% 的停止超时时间来启动 stop 命令。不管发出该 stop 命令后的结果如何，GDS 的 stop 方法都将用 15% 的停止超时时间发送 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 将试图使用在 Stop\_signal 中指定的信号来停止应用程序。

## Probe\_command 特性

probe 命令定期检查给定应用程序的运行状况。它必须是一条具有完整变量的 UNIX 命令，可以直接传给某个 shell 来探测该应用程序。如果应用程序运行成功，则 probe 命令返回 0 退出状态。

probe 命令的退出状态可用于确定应用程序故障的严重程度。此退出状态（又称探测状态）必须是一个介于 0（表示成功）和 100（表示完全失败）之间的整数。探测状态也可以是一个特殊值 201，它将导致应用程序立即进行故障切换，除非 Failover\_enabled 被设置为 false。GDS 探测算法中使用探测状态（请参阅 scds\_fm\_action(3HA)）来确定是在本地重新启动应用程序还是将其故障切换到另一个节点；如果退出状态是 201，立即对应用程序进行故障切换。

如果忽略 probe 命令，则 GDS 将提供其自身的简单 probe 命令。它将与从 Network\_resources\_used 特性或 scds\_get\_netaddr\_list(3HA) 的输出结果中得到的 IP 地址集中的应用程序进行连接。如果连接成功，它将立即断开连接。如果能成功地进行连接和断开连接，则该应用程序即被认为运行状态完好。

---

**注意** – 随 GDS 一起提供 `probe` 命令的目的仅在于为正常运行的、特定于应用程序的 `probe` 命令提供简单的替代命令。

---

## Start\_timeout 特性

此特性用于指定 `start` 命令的启动超时（有关其它信息，请参阅第 161 页“`Start_command` 扩展特性”）。`Start_timeout` 的缺省值为 300 秒。

## Stop\_timeout 特性

此特性用于指定 `stop` 命令的停止超时（有关其它信息，请参阅第 162 页“`Stop_command` 特性”）。`Stop_timeout` 的缺省值为 300 秒。

## Probe\_timeout 特性

此特性用于指定 `probe` 命令的超时值（有关其它信息，请参阅第 162 页“`Probe_command` 特性”）。`Probe_timeout` 的缺省值为 30 秒。

## Child\_mon\_level 特性

此特性提供进程通过 PMF 获得监视的控制。它指定了一个级别，在该级别之内的所有派生子进程都将被监视。这类似于 `pmfadm(1M)` 命令的 `-C` 变量。

忽略此特性，或将其设置为缺省值 `-1`，与忽略 `pmfadm` 命令中的 `-C` 选项具有相同的效果；即，所有子项（和它们的子孙）都将受到监视。

---

**注意** – 此选项只能用标准的 Sun Cluster 管理命令来指定。如果您使用的是 SunPlex Agent Builder，则无法指定此选项。

---

## Failover\_enabled 特性

此布尔扩展特性用于控制资源的故障切换方式。如果此扩展特性被设置为 `true`，则当应用程序在 `retry_interval` 秒内重新启动的次数超过 `retry_count` 时，该应用程序将进行故障切换。

如果将此扩展特性设置为 `false`，则只要应用程序在 `retry_interval` 秒内重启的次数超过 `retry_count` 次，该应用程序就不再重启或向另一个节点进行故障切换。

此扩展特性可用于阻止应用程序资源启动对其资源组的故障切换。缺省值是 `true`。

## Stop\_signal 特性

GDS 使用此整数扩展特性的值来确定通过 PMF 停止应用程序时所使用的信号。有关可以指定的整数值的列表，请参阅 `signal(3HEAD)`。缺省值为 15 (`SIGTERM`)。

---

# 使用 SunPlex Agent Builder 创建使用 GDS 的服务

可以使用 SunPlex Agent Builder 创建使用 GDS 的服务。第 9 章中详细介绍了 SunPlex Agent Builder。

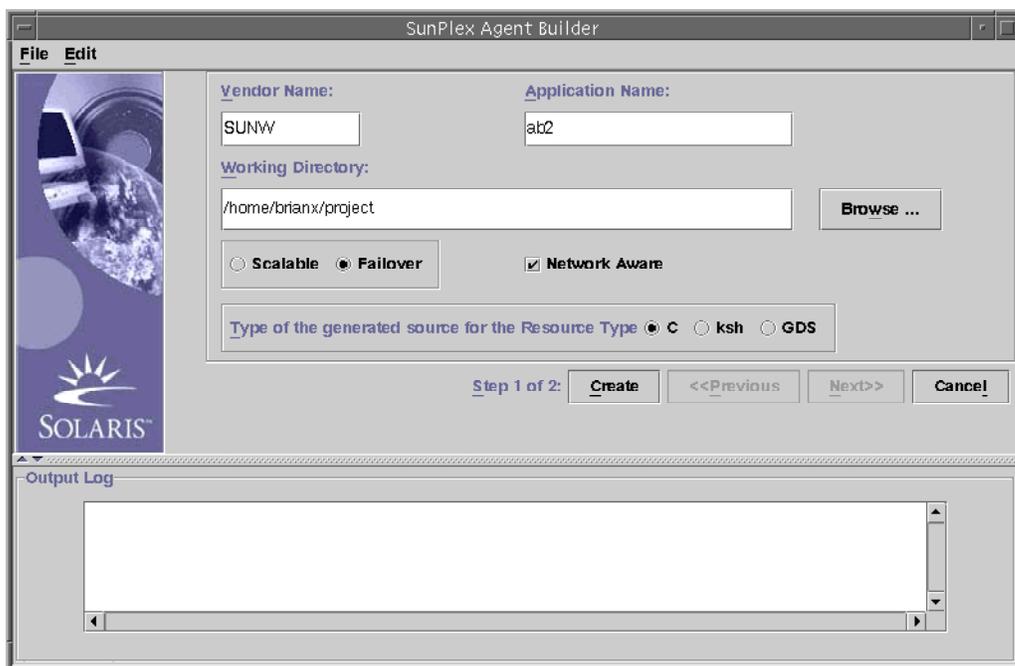
## 在 SunPlex Agent Builder 中创建使用 GDS 的服务

### ▼ 在 Agent Builder 中创建使用 GDS 的服务

1. 启动 SunPlex Agent Builder。

```
# /usr/cluster/bin/scdsbuilder
```

2. 将出现 SunPlex Agent Builder 面板。



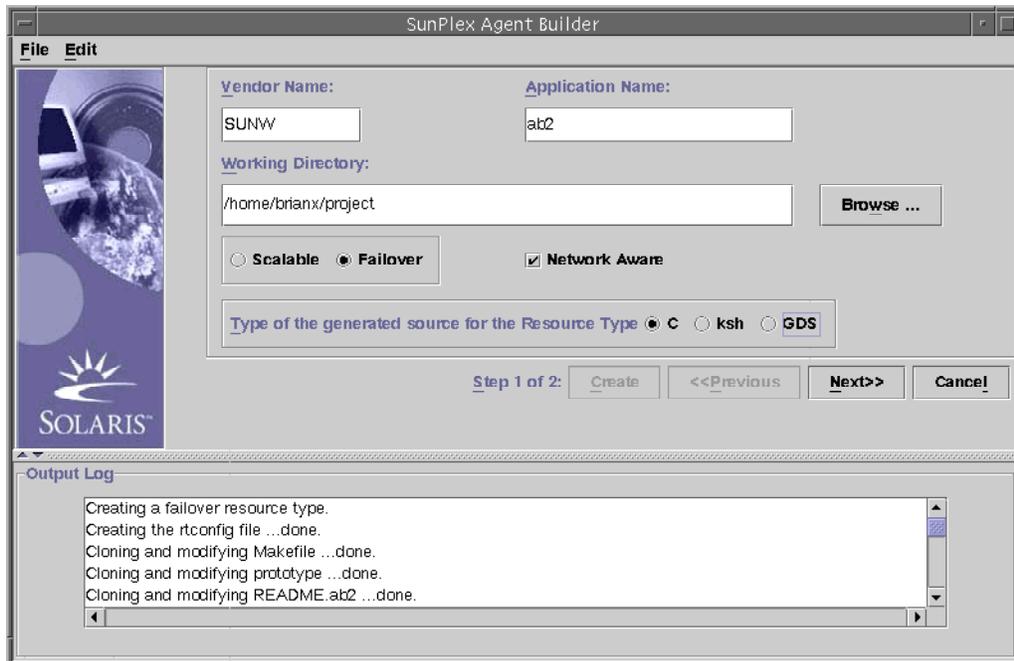
3. 在“Vendor Name”中输入供应商名称。
4. 在“Application Name”中输入应用程序名称。

---

注意 – 供应商名称和应用程序名称加起来不能超过 9 个字符。它被作为驱动脚本的软件包名称。

---

5. 转到工作目录。  
可以使用“Browse”按钮选择一个目录，而不必键入路径。
6. 选择该数据服务是可进行伸缩，还是可进行故障切换。  
不必选择“Network Aware”，因为在创建 GDS 时它是缺省值。
7. 选择“GDS”。
8. 单击“Create”按钮，以创建驱动脚本。
9. SunPlex Agent Builder 面板显示出服务的创建结果。“Create”按钮将灰显，您随即可以使用“Next”按钮。



## ▼ 配置驱动脚本

创建驱动脚本之后，需要使用 SunPlex Agent Builder 配置新服务。

1. 单击“Next”按钮，之后将出现配置面板。
2. 输入 Start 命令的位置，或者使用“Browse”按钮找到 Start 命令。
3. (可选的) 输入 Stop 命令，或者使用“Browse”按钮找到 Stop 命令。
4. (可选的) 输入 Probe 命令，或者使用“Browse”按钮找到 Probe 命令。
5. (可选的) 为 Start、Stop 和 Probe 等命令指定超时值。
6. 单击“Configure”，开始对驱动脚本进行配置。

---

注意 – 该软件包的名称将由“Vendor Name”和“Application Name”综合而成。

---

将创建驱动脚本软件包，并将其放在以下位置：

```
<working-dir>/<vendor_name><application>/pkg  
例如: /export/wdir/NETapp/pkg
```

7. 将已完成的软件包安装到群集中的所有节点上。

```
# cd /export/wdir/NETapp/pkg  
# pkgadd -d . NETapp
```

8. 在 pkgadd 过程中将会安装下列文件：

```
/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
```

---

注意 – 各手册页和脚本名称与您在上面所输入的“应用程序名称”对应，而且其前面缀以脚本名称；例如，startapp。

---

若要查看手册页，需要指定手册页的路径。例如，要查看 startapp 手册页，请使用：

```
# man -M /opt/NETapp/man startapp
```

9. 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 <logichostname> -p <port and protocol list>
```

启动脚本的变量根据资源类型的不同（故障切换或可伸缩）而有所变化。请检查定制的手册页或者运行不带任何变量的启动脚本，以获得用法语句。

```
# /opt/NETapp/util/startapp  
The resource name of LogicalHostname or SharedAddress must be  
specified.  
For failover services:  
Usage: startapp -h <logical host name>  
        -p <port and protocol list>  
        [-n <ipmgroup/adapter list>]  
For scalable services:  
Usage: startapp  
        -h <shared address name>  
        -p <port and protocol list>  
        [-l <load balancing policy>]  
        [-n <ipmgroup/adapter list>]  
        [-w <load balancing weights>]
```

## SunPlex Agent Builder 的输出

SunPlex Agent Builder 根据您在软件包创建过程中提供的输入生成三个驱动脚本和一个配置文件。该配置文件用于指定资源组和资源类型的名称。

驱动脚本分别是：

- 启动脚本：用于配置资源，以及在 RGM 控制下启动应用程序。
- 停止脚本：用于停止应用程序，以及拆分资源和资源组。
- 删除脚本：用于删除由启动脚本所创建的资源 and 资源组。

这些驱动脚本的界面和行为与 SunPlex Agent Builder 为不基于 GDS 的代理而生成的公用程序脚本的界面和行为相同。这些脚本全部放在一个可在 Solaris 中安装的软件包中。该软件包可在多个群集中重复使用。

可以定制配置文件，以便为资源组或其它参数提供定制的名称，这些名称通常在 `scrgadm` 命令的输入内容中出现。如果您不定制这些脚本，则 SunPlex Agent Builder 将为 `scrgadm` 参数提供合理的缺省值。

---

## 使用标准的 Sun Cluster 管理命令来创建使用 GDS 的服务

本节将介绍如何将参数实际输入到 GDS。我们通过现有的 Sun Cluster 管理命令（如 `scrgadm` 和 `scswitch`）来使用和管理 GDS。

如果驱动脚本的功能够用，就不必输入本节所述的低级管理命令。但是，如果您需要对基于 GDS 的资源进行更为严格的控制，则可以这样做。这些命令实际上是驱动脚本所执行的命令。

### ▼ 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

#### 1. 登记资源类型 `SUNW.gds`

```
# scrgadm -a -t SUNW.gds
```

#### 2. 创建资源组，其中包含 `LogicalHostname` 资源和故障切换服务本身。

```
# scrgadm -a -g haapp_rg
```

#### 3. 为 `LogicalHostname` 资源创建资源。

```
# scrgadm -a -L -g haapp_rs -l hhead
```

4. 为故障切换服务本身创建资源。

```
# 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/apctl/start" \  
-x Stop_command="/export/ha/apctl/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
```

5. 将资源组 haapp\_rg 联机。

```
# scswitch -Z -g haapp_rg
```

## ▼ 标准的 Sun Cluster 管理命令（用于创建使用 GDS 的、可伸缩的服务）

1. 登记资源类型 SUNW.gds。

```
# scrgadm -a -t SUNW.gds
```

2. 为 SharedAddress 资源创建资源组。

```
# scrgadm -a -g sa_rg
```

3. 在 sa\_rg 上创建 SharedAddress 资源。

```
# scrgadm -a -S -g sa_rg -l hhead
```

4. 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
```

5. 为可伸缩服务自身创建资源组。

```
# 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
```

6. 将包含网络资源的资源组联机。

```
# scswitch -Z -g sa_rg
```

7. 将资源组 app\_rg 联机。

```
# scswitch -Z -g app_rg
```

---

## SunPlex Agent Builder 的命令行界面

SunPlex Agent Builder 具有一个功能与 GUI 界面相同的命令行界面。此界面由 `scdscreate(1HA)` 和 `scdsconfig(1HA)` 命令组成。本节中的操作与基于 GUI 的过程第 170 页“使用 Agent Builder 的命令行版本创建使用 GDS 的服务”具有相同的功能，但不使用 GUI 界面。

### ▼ 使用 Agent Builder 的命令行版本创建使用 GDS 的服务

#### 1. 创建服务。

对于故障切换服务，请使用：

```
# scdscreate -g -V NET -T app -d /export/wdir
```

对于可伸缩服务，请使用：

```
# scdscreate -g -s -V NET -T app -d /export/wdir
```

---

注意 --d 参数是可选的。如果不指定此参数，则工作目录缺省为当前目录。

---

#### 2. 配置服务。

```
# scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" \  
-m "/export/app/bin/probe" -d /export/wdir
```

---

注意 - 只有 start 命令是必需的。所有其它参数都是可选的。

---

#### 3. 将已完成的软件包安装到群集中的所有节点上。

```
# cd /export/wdir/NETapp/pkg  
# pkgadd -d . NETapp
```

#### 4. 在 pkgadd 过程中将会安装下列文件：

```
/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
```

---

**注意** – 各手册页和脚本名称与您在上面所输入的“应用程序名称”对应，而且其前面缀以脚本名称；例如，startapp。

---

若要查看手册页，需要指定手册页的路径。例如，要查看 startapp 手册页，请使用：

```
# man -M /opt/NETapp/man 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 <logichostname> -p <port and protocol list>
```

启动脚本的变量根据资源类型的不同（故障切换或可伸缩）而有所变化。请检查定制的手册页或者运行不带任何变量的启动脚本，以获得用法语句。

```
# /opt/NETapp/util/startapp
```

The resource name of LogicalHostname or SharedAddress must be specified.

For failover services:

```
Usage: startapp -h <logical host name>
        -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

# 数据服务开发库参考

---

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 仅定义了一个 C 接口。没有可编写脚本的 DSDL 接口。

DSDL 提供了以下几类函数。

- 第 173 页 “通用函数”
- 第 174 页 “特性函数”
- 第 175 页 “网络资源存取函数”
- 第 176 页 “PMF 函数”
- 第 176 页 “故障监视器函数”
- 第 176 页 “公用程序函数”

---

## DSDL 函数

以下各小节简要介绍了各类 DSDL 函数。但是，各个 3HA 手册页才是 DSDL 函数的权威参考。

### 通用函数

本节中介绍的函数具有很多功能。使用这些函数，您可以：

- 初始化 DSDL 环境
- 检索资源、资源类型和资源组名称以及扩展特性值
- 进行故障切换并重启资源组以及重启资源
- 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

以下函数用来初始化调用方法。

- `scds_initialize` — 用来分配资源和初始化 DSDL 环境。
- `scds_close` — 用来释放通过 `scds_initialize` 分配的资源。

以下函数用来检索有关资源、资源类型、资源组和扩展特性的信息。

- `scds_get_resource_name` — 用来检索调用程序的资源的名称。
- `scds_get_resource_type_name` — 用来检索调用程序的资源类型的名称。
- `scds_get_resource_group_name` — 用来检索调用程序的资源组的名称。
- `scds_get_ext_property` — 用来检索指定扩展特性的值。
- `scds_free_ext_property` — 用来释放通过 `scds_get_ext_property` 分配的内存。

以下函数检索某资源所使用的 SUNW.HAStoragePlus 资源的状态信息。

- `scds_hasp_check` — 用来检索某资源所使用的 SUNW.HAStoragePlus 资源的状态信息。使用为该资源定义的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系统特性，从资源依赖的所有 SUNW.HAStoragePlus 资源的状态（联机或其它）中获取此信息。  
有关 SUNW.HAStoragePlus 的详细信息，请参阅 `SUNW.HAStoragePlus(5)`。

以下函数用来进行故障切换或重启资源或资源组。

- `scds_failover_rg` — 用来故障切换资源组。
- `scds_restart_rg` — 用来重启资源组。
- `scds_restart_resource` — 用来重启资源。

以下两个函数用来在超时值定义的时间之内执行命令，并将错误代码转换成错误消息。

- `scds_timerun` — 用来在超时值定义的时间之内执行命令。
- `scds_error_string` — 用来将错误代码转换成错误字符串。

## 特性函数

这些函数提供了公用 API，可用来存取相关资源、资源组和资源类型的具体特性，其中包括一些常用的扩展特性。DSDL 提供了 `scds_initialize` 函数，可用来分析命令行变量。然后该库将缓存相关资源、资源组和资源类型的各种特性。

单个手册页 `scds_property_functions(3HA)` 中介绍了所有这些函数。本节介绍以下函数：

- `scds_get_rs_property_name`
- `scds_get_rg_property_name`
- `scds_get_rt_property_name`
- `scds_get_ext_property_name`

## 网络资源存取函数

本节中列出的函数用来检索、打印和释放资源和资源组所使用的网络资源。本节中的 `scds_get_*` 函数提供了一种用来检索网络资源而又无需使用 `RMAPI` 函数查询具体特性（例如 `Network_resources_used` 和 `Port_list`）的简便方式。

`scds_print_name()` 函数用来打印 `scds_get_name()` 函数返回的数据结构的值。  
`scds_free_name()` 函数用来释放通过 `scds_get_name()` 函数分配的内存。

以下函数用来执行与主机名有关的操作。

- `scds_get_rg_hostnames` — 用来检索某个资源组中网络资源使用的主机名列表。
- `scds_get_rs_hostnames` — 用来检索该资源所使用的主机名列表。
- `scds_print_net_list` — 用于打印通过 `scds_get_rg_hostnames` 或 `scds_get_rs_hostnames` 返回的主机名列表的内容。
- `scds_free_net_list` — 用来释放通过 `scds_get_rg_hostnames` 或 `scds_get_rs_hostnames` 分配的内存。

以下函数用来执行与端口列表有关的操作。

- `scds_get_port_list` — 用来检索某资源所使用的端口协议对的列表。
- `scds_print_port_list` — 用来打印通过 `scds_get_port_list` 返回的端口协议对列表的内容。
- `scds_free_port_list` — 用来释放通过 `scds_get_port_list` 分配的内存。

以下函数用来执行与网络地址有关的操作。

- `scds_get_netaddr_list` — 用来检索某资源所使用的网络地址的列表。
- `scds_print_netaddr_list` — 用来打印通过 `scds_get_netaddr_list` 返回的网络地址列表的内容。
- `scds_free_netaddr_list` — 用来释放通过 `scds_get_netaddr_list` 分配的内存。

## 使用 TCP 连接进行故障监视

本节介绍的函数用来启用基于 TCP 的监视功能。通常，故障监视器使用这些函数建立服务的简单套接字连接，对该服务进行读写操作以确定其状态，然后从该服务断开连接。

本节介绍以下函数：

- `scds_tcp_connect` — 用来建立进程的 TCP 连接。
- `scds_tcp_read` — 使用 TCP 连接从所监视的进程读取数据。
- `scds_tcp_write` — 使用 TCP 连接向所监视的进程写入数据。
- `scds_simple_probe` — 用来通过建立和终止进程的 TCP 连接来探测该进程。
- `scds_tcp_disconnect` — 用来终止所监视的进程的连接。

## PMF 函数

这些函数用来封装 PMF 功能。通过 PMF 进行监视的 DSDL 模型可以创建并使用 pmfadm(1M) 的隐含 *tag* 值。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 将把 PMF 发现进程停止情况时进行的进程停止历史记录与故障监视器检测到故障情况时进行的应用程序故障历史记录结合起来, 以通过计算决定是进行重启还是进行故障切换。

本节介绍以下函数:

- `scds_pmf_get_status` — 用来确定指定的实例是否在 PMF 控制下进行监视。
- `scds_pmf_restart_fm` — 使用 PMF 重启故障监视器。
- `scds_pmf_signal` — 用来向在 PMF 控制下运行的进程树发送指定信号。
- `scds_pmf_start` — 用来在 PMF 控制下执行指定的程序 (包括故障监视器)。
- `scds_pmf_stop` — 用来终止在 PMF 控制下运行的进程。
- `scds_stop_monitoring` — 用来停止监视在 PMF 控制下运行的进程。

## 故障监视器函数

本节中介绍的函数通过保留失败历史记录, 并与 `Retry_count` 和 `Retry_interval` 特性一起对其进行计算, 提供了一种预设的故障监视模型。

本节介绍以下函数:

- `scds_fm_sleep` — 用来等待故障监视器控制套接字的消息。
- `scds_fm_action` — 用来在探测完成后执行操作。
- `scds_fm_print_probes` — 用来将探测状态信息写入系统日志。

## 公用程序函数

使用本节介绍的函数可以向系统日志写入消息和调试消息。本节介绍以下两种函数:

- `scds_syslog` — 向系统日志写入消息。
- `scds_syslog_debug` — 用来将调试消息写入系统日志。

## 第 12 章

---

# CRNP

---

本章介绍群集重配置通知协议 (CRNP)。CRNP 使故障切换和可伸缩应用程序能够“支持群集”。更重要的是，CRNP 提供了一种机制，使应用程序能够登记并接收 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登记事件通知。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。

- 第 177 页 “CRNP 概述”
- 第 180 页 “CRNP 使用的消息类型”
- 第 181 页 “客户机如何向服务器进行登记”
- 第 183 页 “服务器如何对客户机进行应答”
- 第 184 页 “服务器如何向客户机传送事件”
- 第 187 页 “CRNP 如何鉴别客户机和服务器”
- 第 188 页 “创建使用 CRNP 的 Java 应用程序”

---

## CRNP 概述

CRNP 提供了相应的机制和守护程序，可以生成群集重配置事件并通过群集将它们路由到感兴趣的客户机。

cl\_apid 守护程序与客户机交互。Sun Cluster 资源组管理器 (RGM) 生成群集重配置事件。这些守护程序使用 syseventd(1M) 传输各个本地节点上的事件。cl\_apid 守护程序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。

下图概括了 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。

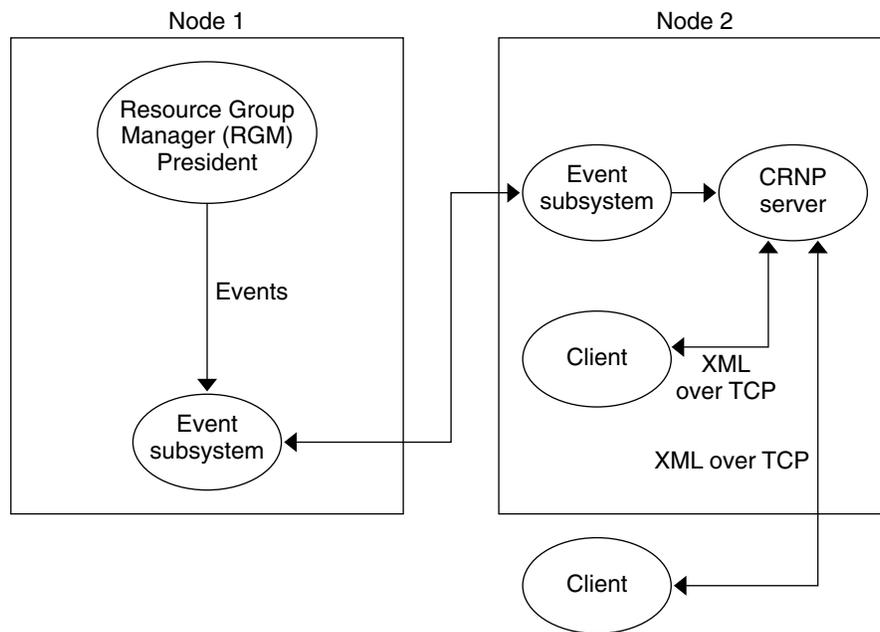


图 12-1 CRNP 的工作原理

## CRNP 协议概述

CRNP 定义了标准七层开放式系统互联 (OSI) 协议栈的应用程序层、表示层和会话层。传输层必须是传输控制协议 (TCP)，网络层必须是网间协议 (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 协议的语义

客户机通过向服务器发送一条登记消息 (SC\_CALLBACK\_RG) 来启动通信。此登记信息指定客户机要接收其通知的事件类型，还指定事件能够被传输到的端口。登记连接的源 IP 和指定的端口一起形成回叫地址。

当群集中生成某台客户机感兴趣的事件时，服务器将通过回叫地址 (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。服务器将客户机登记存储在存储器中，即使重新引导群集，登记信息也会保留在存储器中。

客户机通过向服务器发送一条登记消息 (SC\_CALLBACK\_RG，包含 REMOVE\_CLIENT 消息) 来撤销登记。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。

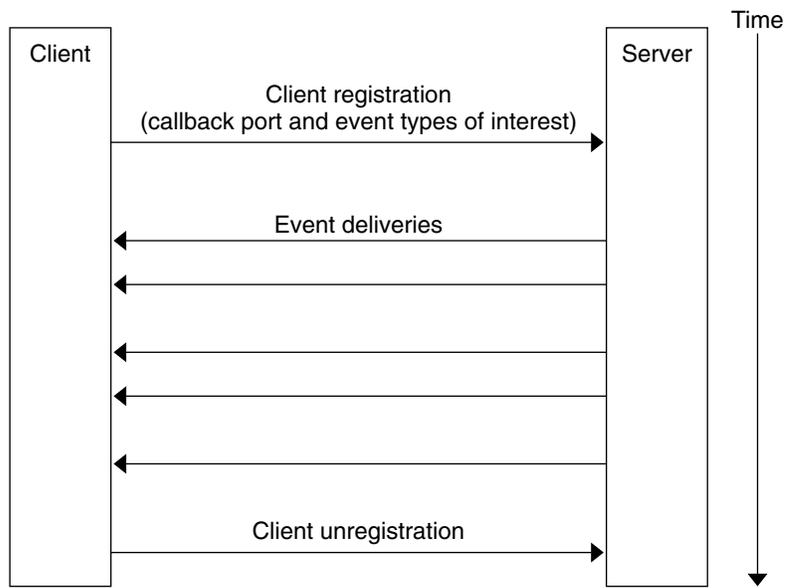


图 12-2 客户机和服务器之间的通信流程

---

## CRNP 使用的消息类型

CRNP 使用三种类型的消息（都基于 XML），如下表所示。本章后面将对这些消息类型进行详细介绍， 同时还将详细介绍这些消息类型的使用方式。

消息类型	说明
SC_CALLBACK_REG	<p>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：</p> <ul style="list-style-type: none"><li>■ 协议版本</li><li>■ ASCII 格式（而不是二进制格式）的回叫端口</li></ul> <p>ADD_CLIENT、ADD_EVENTS 和 REMOVE_EVENTS 格式还包含无界限的事件类型列表，每个类型都包括以下信息：</p> <ul style="list-style-type: none"><li>■ 事件类</li><li>■ 事件子类（可选）</li><li>■ 名称和值对列表（可选）</li></ul> <p>事件类和事件子类一同定义唯一的“事件类型”。生成 SC_CALLBACK_REG 类的 DTD（文档类型定义）是 SC_CALLBACK_REG。附录 F 中详细介绍了此 DTD。</p>
SC_EVENT	<p>此类消息包含以下信息：</p> <ul style="list-style-type: none"><li>■ 协议版本</li><li>■ 事件类</li><li>■ 事件子类</li><li>■ 供应商</li><li>■ 发行商</li><li>■ 名称和值对列表（0 或多个名称和值对数据结构）<ul style="list-style-type: none"><li>■ 名称（字符串）</li><li>■ 值（字符串或字符串数组）</li></ul></li></ul> <p>SC_EVENT 中的值未定义类型。生成 SC_EVENT 类的 DTD（文档类型定义）是 SC_EVENT。附录 F 中详细介绍了此 DTD。</p>
SC_REPLY	<p>此类消息包含以下信息：</p> <ul style="list-style-type: none"><li>■ 协议版本</li><li>■ 错误代码</li><li>■ 错误消息</li></ul> <p>生成 SC_REPLY 类的 DTD（文档类型定义）是 SC_REPLY。附录 F 中详细介绍了此 DTD。</p>

---

## 客户机如何向服务器进行登记

本节介绍管理员如何设置服务器，如何识别客户机，如何通过应用层和会话层发送信息，并介绍了错误状态。

### 管理员设置服务器的前提

系统管理员必须使用高度可用的 IP 地址（不依赖于群集中某台特定计算机的 IP 地址）和端口号来配置服务器。管理员必须向目标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。管理员可以使用命名服务，使客户机能够动态找到服务器网络地址，或者将网络名称添加到配置文件中，以便客户机读取。服务器将作为故障切换资源类型在群集中运行。

### 服务器如何标识客户机

每台客户机均由其回叫地址（IP 地址和端口号）唯一标识。端口在 SC\_CALLBACK\_REG 消息中指定，IP 地址从 TCP 登记连接获取。CRNP 假定具有相同回叫地址的后续 SC\_CALLBACK\_REG 消息来自同一台客户机，即使发送这些消息的源端口不相同。

### 如何在客户机和服务器之间传送 SC\_CALLBACK\_REG 消息

客户机通过打开一个指向服务器的 IP 地址和端口号的 TCP 连接来启动登记。建立 TCP 连接并做好写入准备后，客户机必须发送其登记消息。登记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它字节。

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，服务器将不登记客户机，并向客户机发送一条出错应答。如果客户机在服务器发出应答之前关闭套接字连接，服务器将照常登记客户机。

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。

客户机不能在发送 ADD\_CLIENT 消息之前发送 ADD\_EVENTS、REMOVE\_EVENTS 或 REMOVE\_CLIENT 消息。客户机发送 ADD\_CLIENT 消息后，才能发送 REMOVE\_CLIENT 消息。

如果某台客户机发送一条 `ADD_CLIENT` 消息，但该客户机已经登记，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登记替换旧的客户机登记，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` 消息向服务器登记一次，并通过向服务器发送一条 `REMOVE_CLIENT` 消息撤销登记一次。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

每条 `ADD_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：

- 发送一条 `REMOVE_EVENTS` 消息，其中指定的一个或多个事件类型是该客户机以前所未曾登记的，或者
- 为同一事件类型登记了两次

则服务器将静默忽略这些消息。

类和子类	名称和值对	说明
<code>EC_Cluster</code>	必需：无	登记所有群集成员更改事件（节点断开或连接）
<code>ESC_cluster_membership</code>	可选：无	
<code>EC_Cluster</code> <code>ESC_cluster_rg_state</code>	一个是必需的，如下所示： <code>rg_name</code> 值类型：字符串 可选：无	登记资源组 <i>name</i> 的所有状态更改事件
<code>EC_Cluster</code> <code>ESC_cluster_r_state</code>	一个是必需的，如下所示： <code>r_name</code> 值类型：字符串 可选：无	登记资源 <i>name</i> 的所有状态更改事件
<code>EC_Cluster</code> 无	必需：无 可选：无	登记所有 Sun Cluster 事件

---

## 服务器如何对客户机进行应答

处理登记后，服务器将发送 `SC_REPLY` 消息。服务器将通过从其上接收登记请求的那台客户机上打开的 TCP 连接发送此消息，然后关闭连接。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. 等候 `SC_REPLY` 消息
5. 接收 `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

客户机稍后将执行以下操作：

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. 等候 `SC_REPLY` 消息
5. 接收 `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 280 页“`SC_REPLY XML DTD`”包含 `SC_REPLY` 消息的 XML 文档类型定义，以及此消息可能包含的错误消息。

## SC\_REPLY 消息的内容

`SC_REPLY` 消息用于指明某个操作是否成功。它包含 CRNP 协议消息的版本、一个状态码和一条详细描述此状态码的状态消息。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	由于出现瞬态错误，客户机登记被服务器拒绝（客户机应使用其它参数尝试重新登记）。

状态码	说明
LOW_RESOURCE	群集资源不足，客户机只能以后再尝试（该群集的系统管理员也可以增加群集资源）。
SYSTEM_ERROR	发生严重问题。与该群集的系统管理员联系。
FAIL	授权失败，或者其它问题导致登记失败。
MALFORMED	XML 请求的格式不正确，无法进行分析。
INVALID	XML 请求无效（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

正常情况下，发送 SC\_CALLBACK\_REG 消息的客户机将收到一个表明登记是否成功的应答。

但是，当客户机登记时，服务器可能正处于一种错误状态，从而使服务器无法向客户机发送 SC\_REPLY 消息。在这种情况下，登记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。

由于服务器必须充当故障切换或高度可用的群集服务器，所以此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登记的客户机发送事件。

要改正这些状态，客户机应当：

- 对正在等候 SC\_REPLY 消息的登记连接强制执行一个应用程序级别的超时，随后客户机需要重试登记。
- 在登记事件回叫之前，开始在其回叫 IP 地址和端口上侦听事件传送。客户机应当同时等候登记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，客户机将静默关闭登记连接。

## 服务器如何向客户机传送事件

随着群集中事件的生成，CRNP 服务器将这些事件传送到请求这些类型的事件的所有客户机。发送过程包括向客户机的回叫地址发送一条 SC\_EVENT 消息。每个事件都是通过一个新的 TCP 连接传送的。

客户机登记事件类型后，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。这样客户机就可以了解发送后续事件的系统的当前状态。

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，然后执行全双工关闭。

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. 等候 SC\_EVENT 消息
4. 读取 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

所有客户机都登记完成后，必须始终在各自的回叫地址（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。要接收更多事件，客户机还需要通过发送另一条包含 ADD\_CLIENT 消息的 SC\_CALLBACK\_REG 消息重新登记。

## 如何保障事件的传送

群集中生成的事件具有一个全序，按照传送到每个客户机的顺序保存。换句话说，如果群集中先生成事件 A，然后生成事件 B，那么客户机 X 将先接收事件 A，然后接收事件 B。但是**不会**保存传送到**所有**客户机的事件的全序。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登记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。也就是说，允许服务器将同一个事件多次发送到一台客户机上。这在服务器发生临时故障，恢复正常后无法判断客户机是否已接收到最新信息时特别有用。

## SC\_EVENT 消息的内容

SC\_EVENT 消息包含群集中实际生成的消息，该消息已经过转换，符合 SC\_EVENT XML 消息的格式要求。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。

类和子类	发行商和供应商	名称和值对	说明
EC_Cluster	发行商: rgm	名称: node_list	<p>state_list 中数组元素的位置与 node_list 中的位置同步。即, node_list 数组中第一个列出的节点在 state_list 数组中也是第一个列出。</p> <p>state_list 仅包含以 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。</p> <p>可能会有以 ev_ 开头的其它名称及其关联的值, 但它们不供客户机使用。</p>
ESC_cluster_membership	供应商: SUNW	值类型: 字符串数组 名称: state_list 值类型: 字符串数组	
EC_Cluster	发行商: rgm	名称: rg_name	<p>state_list 中数组元素的位置与 node_list 中的位置同步。即, node_list 数组中第一个列出的节点在 state_list 数组中也是第一个列出。</p> <p>state_list 包含资源组状态的字符串表示。有效值是可以使用 scha_cmds(1HA) 命令检索的值。</p> <p>可能会有以 ev_ 开头的其它名称及其关联的值, 但它们不供客户机使用。</p>
ESC_cluster_rg_state	供应商: SUNW	值类型: 字符串 名称: node_list 值类型: 字符串数组 名称: state_list 值类型: 字符串数组	

类和子类	发行商和供应商	名称和值对	说明
EC_Cluster	发行商: <b>rgm</b>	有三个是必需的, 如下所示:	<code>state_list</code> 中数组元素的位置与 <code>node_list</code> 中的位置同步。即, <code>node_list</code> 数组中第一个列出的节点在 <code>state_list</code> 数组中也是第一个列出。  <code>state_list</code> 包含资源状态的字符串表示。有效值是可以使用 <code>scha_cmds(1HA)</code> 命令检索的值。  可能会有以 <code>ev_</code> 开头的其它名称及其关联的值, 但它们不供客户机使用。
ESC_cluster_r_state	供应商: <b>SUNW</b>	名称: <code>r_name</code>	
		值类型: 字符串	
		名称: <code>node_list</code>	
		值类型: 字符串数组	
		名称: <code>state_list</code>	
		值类型: 字符串数组	

## CRNP 如何鉴别客户机和服务器

服务器使用 TCP 包装的形式鉴别客户机。服务器上允许的客户机列表中必须包含登记消息的源 IP 地址（也作为发送事件的回叫地址）。拒绝的客户机列表中不能包含源 IP 地址和登记消息。如果源 IP 地址和登记不在列表中，服务器将拒绝请求，并向客户机发送一个出错应答。

当服务器接收到 `SC_CALLBACK_REG_ADD_CLIENT` 消息时，该客户机的后续 `SC_CALLBACK_REG` 消息必须包含一个与第一条消息中的源 IP 地址相同的源 IP 地址。如果 CRNP 服务器接收到的 `SC_CALLBACK_REG` 不满足此要求，服务器将：

- 忽略请求并向客户机发送一个出错应答，或者
- 假设该请求来自一台新客户机（取决于 `SC_CALLBACK_REG` 消息的内容）

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登记。

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登记 IP 地址和端口号相同的服务器上的事件传送。

由于 CRNP 服务的客户机需要位于保护该群集的防火墙内，因此 CRNP 不包含其它安全机制。

---

## 创建使用 CRNP 的 Java 应用程序

以下实例说明了如何开发名为 `CrnpClient` 的、使用 CRNP 的简单 Java 应用程序。应用程序通过群集上的 CRNP 服务器登记事件回叫，侦听事件回叫，并通过打印事件的内容对事件进行处理。应用程序终止前将撤销登记事件回叫请求。

查看此实例时请牢记以下几个要点：

- 样例应用程序使用 JAXP (Java API for XML Processing) 生成和分析 XML。此实例并未介绍如何使用 JAXP。有关 JAXP 的详细信息，请访问 <http://java.sun.com/xml/jaxp/index.html>。
- 此实例提供了应用程序的几个片断，完整的应用程序可以从附录 G 中找到。为了更有效地说明某些概念，本章中的实例与附录 G 中的完整应用程序略有不同。
- 为了简洁起见，本章实例中的样例代码不包含注释。附录 G 中的完整应用程序包含注释。
- 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。

### ▼ 设置环境

首先需要设置环境。

1. 下载并安装 JAXP 以及 Java 编译器和虚拟机的正确版本。

相关指令可以从 <http://java.sun.com/xml/jaxp/index.html> 上找到。

---

注意 – 此实例要求使用 Java 1.3.1 或更高版本。

---

2. 确保在编译命令行中指定 `classpath`，以使编译器能够找到 JAXP 类。在源文件所在的目录下键入以下内容：

```
% 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` 是 JAXP jar 文件所在目录的绝对路径或相对路径，`SOURCE_FILENAME` 是 Java 源文件的名称。

3. 运行应用程序时，请指定 `classpath`，以使应用程序能够装入正确的 JAXP 类文件（请注意，`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。

## ▼ 开始

在实例的这一部分，您可以使用分析命令行参数并构造 `CrnpClient` 对象的主要方法创建一个名为 `CrnpClient` 的基类。此对象将命令行参数传递给类，等候用户终止应用程序，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叫的线程。
- 联系 CRNP 服务器并登记事件回叫。

- **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。后文介绍了构造函数中引用的这四个帮助程序方法以及停机方法的实现。请注意，用来输入所需软件包的代码如下。

```
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;
}

```

后文详细介绍了成员变量。

## ▼ 分析命令行参数

- 要了解如何分析命令行参数，请参考附录 G 中的代码。

## ▼ 定义事件接收线程

在本代码中，必须确保在单独的线程中执行事件接收，这样应用程序才能在事件线程中断以及等候事件回叫时继续执行其它工作。

---

注意 - 后文介绍了如何设置 XML。

---

### 1. 在代码中定义一个名为 `EventReceptionThread` 的 `Thread` 子类，用于创建 `ServerSocket` 并等候事件到达套接字。

在实例代码的这一部分，既没有读取事件也没有处理事件。后文介绍了如何读取和处理事件。`EventReceptionThread` 在通配符网络互联网协议地址上创建一个 `ServerSocket`。`EventReceptionThread` 还保留 `CrnpClient` 对象的一个引用，这样 `EventReceptionThread` 才能够将事件发送到 `CrnpClient` 对象进行处理。

```
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. 了解 `EventReceptionThread` 类的工作原理后，就可以构造 `createEvtRecepThr` 对象了：

```

private void createEvtRecepThr() throws Exception
{
    evtThr = new EventReceptionThread(this);
    evtThr.start();
}

```

## ▼ 登记和撤销登记回叫

登记任务包括：

- 向登记网络互联协议和端口打开一个基本 TCP 套接字
- 构造 XML 登记消息
- 通过套接字发送 XML 登记消息
- 脱离套接字并读取 XML 应答消息
- 关闭套接字

### 1. 创建实现上述逻辑的 Java 代码。

以下实例显示了 CrnpClient 类的 registerCallbacks 方法的实现，该方法由 CrnpClient 构造函数调用。后文详细介绍了 createRegistrationString() 和 readRegistrationReply() 的调用。

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。此方法是由 CrnpClient 的 shutdown 方法调用的。后文详细介绍了 createUnregistrationString 的实现。

```

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();
}

```

## ▼ 生成 XML

设置应用程序结构并编写所有联网代码后，就可以编写生成和分析 XML 的代码了。第一步是编写生成 SC\_CALLBACK\_REG XML 登记消息的代码。

SC\_CALLBACK\_REG 消息包括登记类型 (ADD\_CLIENT、REMOVE\_CLIENT、ADD\_EVENTS 或 REMOVE\_EVENTS)、回叫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，需要编写一个存储登记类型、回叫端口和登记事件列表的 CallbackReg 类。此类还可以将自身串行化为一个 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。位于 <http://java.sun.com/xml/jaxp/index.html> 上的 JAXP 文档详细介绍了此方法的代码。

Event 类的实现如下所示。请注意，CallbackReg 类使用一个可以存储一个事件并可以将其转换成 XML Element 的 Event 类。

### 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 类使用一个 Event 类，该 Event 类使用一个 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;
}

```

## ▼ 创建登记消息和撤销登记消息

创建生成 XML 消息的帮助程序类后，就可以编写 `createRegistrationString` 方法的实现了。此方法是由 `registerCallbacks` 方法调用的，第 192 页“登记和撤销登记回叫”中对 `registerCallbacks` 方法进行了介绍。

`createRegistrationString` 构造一个 `CallbackReg` 对象并设置其登记类型和端口。然后 `createRegistrationString` 使用 `createAllEvent`、`createMembershipEvent`、`createRgEvent` 和 `createREvent` 帮助程序方法构造各种事件。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 成员变量可以存储用户向应用程序提供的命令行参数。第五个参数及后续参数用于指定应用程序应该登记的事件。第四个参数用于指定登记的类型，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这个参数。

### 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. 创建撤销登记字符串。

创建撤销登记字符串比创建登记字符串要简单，因为不必考虑事件：

```

private String createUnregistrationString() throws Exception
{
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
    String xmlStr = cbReg.convertToXml();
    return (xmlStr);
}

```

## ▼ 设置 XML 分析器

现在已经为应用程序创建了联网和 XML 生成代码。最后一步是分析及处理登记应答和事件回叫。CrnpClient 构造函数将调用 `setupXmlProcessing` 方法。此方法将创建 `DocumentBuilderFactory` 对象并设置该对象的各种分析特性。位于 <http://java.sun.com/xml/jaxp/index.html> 上的 JAXP 文档详细介绍了此方法。

### ● 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);
}

```

## ▼ 分析登记应答

要分析 CRNP 服务器为响应登记消息和撤销登记消息而发送的 SC\_REPLY XML 消息，需要使用 RegReply 帮助程序类。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存取程序。要分析来自服务器的 XML 数据流，需要创建一个新的 XML 文档并使用该文档的分析方法（位于 <http://java.sun.com/xml/jaxp/index.html> 上的 JAXP 文档详细介绍了此方法）。

### 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。位于 <http://java.sun.com/xml/jaxp/index.html> 上的 JAXP 文档包含了详细的信息。

```

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;
}

```

## ▼ 分析回叫事件

最后一步是分析和处理实际的回叫事件。要协助执行此任务，需要修改您在第 192 页“生成 XML”中创建的 Event 类，使其能够从 XML 文档中构造一个 Event，并创建一个 XML Element。此更改需要一个附加构造函数（调用 XML 文档）、一个 retrieveValues 方法、两个附加成员变量（vendor 和 publisher）、所有字段的存取程序方法，以及一个打印方法。

### 1. 创建实现上述逻辑的 Java 代码。

请注意，此代码与第 199 页“分析登记应答”中描述的 RegReply 类的代码相似。

```
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. 实现支持 XML 分析的 NVPair 类的其它构造函数和方法。

步骤 1 中对 Event 类的更改要求对 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. 在 `EventReceptionThread` 中实现等候事件回叫的 `while` 循环（第 191 页“定义事件接收线程”中对 `EventReceptionThread` 进行了介绍）。

```

while(true) {
    Socket sock = listeningSock.accept();
    Document doc = db.parse(sock.getInputStream());
    Event event = new Event(doc);
    client.processEvent(event);
    sock.close();
}

```

## ▼ 运行应用程序

- 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

---

# 标准特性

---

本附录介绍资源类型、资源组和资源的标准特性。此外，本附录还介绍可用来更改系统定义的特性和用来创建扩展特性的资源特性属性。

本附录包含以下几个主要部分：

- 第 205 页 “资源类型特性”
- 第 210 页 “资源特性”
- 第 217 页 “资源组特性”
- 第 219 页 “资源特性属性”

---

注意 - 特性值（例如 True 和 False）不区分大小写。

---

---

## 资源类型特性

下表介绍了 Sun Cluster 定义的资源类型特性。特性值分为以下几类（在“类别”列中）：

- **必需的** — 该特性在资源类型登记 (RTR) 文件中需要一个显式值，否则将无法创建该特性所属的对象。该值不能是空白或空字符串。
- **有条件的** — 要使特性存在，必须在 RTR 文件中声明该特性；否则，RGM 无法创建该特性，并且它不可用于管理公用程序。允许使用空白或空字符串。如果在 RTR 文件中声明特性但未指定值，RGM 将提供缺省值。
- **有条件的/显式** — 要使特性存在，必须使用显式值在 RTR 文件中声明该特性；否则，RGM 无法创建该特性，并且它不可用于管理公用程序。不允许使用空白或空字符串。
- **可选的** — 可以在 RTR 文件中声明该特性。如果在 RTR 文件中未声明特性，则 RGM 将创建该特性并提供一个缺省值。如果特性在 RTR 文件中进行了声明但未指定其值，则 RGM 所提供的缺省值与未在 RTR 文件中声明该特性时提供的缺省值相

同。

资源类型特性无法通过管理公用程序进行更新。不过 `Installed_nodes` 除外，该特性无法在 `RTR` 文件中进行声明，而是必须由管理员设置。

表 A-1 资源类型特性

特性名称	说明	是否可以更新？	类别
<code>Allow_hosts</code> (字符串数组)	<p>用于控制一组客户机，允许这些客户机通过 <code>cl_apid</code> 守护程序进行登记以接收群集重新配置事件。该特性的格式一般为 <code>ipaddress/masklength</code>，其中定义了允许其上包含的客户机进行登记的子网。例如，设置 <code>129.99.77.0/24</code> 允许位于 <code>129.99.77</code> 子网上的客户机登记事件。这里还有另外一个例子，<code>192.9.84.231/32</code> 仅允许客户机 <code>192.9.84.231</code> 登记事件。此特性可以为 <code>CRNP</code> 提供保护。<code>cl_apid</code> 守护程序在 <code>SUNW.Event(5)</code> 中介绍。</p> <p>此外，还可以识别以下特殊关键字。<code>LOCAL</code> 指的是直接连接的群集子网中的所有客户机。<code>ALL</code> 允许所有客户机进行登记。注意，如果某个客户机同时与 <code>Allow_hosts</code> 和 <code>Deny_hosts</code> 特性中的项相匹配，则该实现将阻止客户机进行登记。</p> <p>缺省值为 <code>LOCAL</code>。</p>	N	可选的
<code>API_version</code> (整数)	<p>此资源类型实现所使用的资源管理 API 的版本。 <code>Sun Cluster 3.1 4/04</code> 中，该特性的缺省值为 2。</p>	N	可选的
<code>Boot</code> (字符串)	<p>一种可选的回叫方法：节点连接或重新连接群集时，<code>RGM</code> 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）。与 <code>Init</code> 方法相似，此方法将对此类型的资源执行初始化操作。</p>	N	有条件的/显式
<code>Client_retry_count</code> (整数)	<p>控制 <code>cl_apid</code> 守护程序在与外部客户机进行通信的期间进行尝试的次数。如果经过 <code>Client_retry_count</code> 次尝试后客户机仍无响应，则该客户机超时。随后将从有资格接收群集重新配置事件的已登记客户机列表中删除该客户机。要想重新开始接收事件，必须重新登记该客户机。要了解有关该实现的重试频率，请参阅 <code>Client_retry_interval</code> 特性的描述。 <code>SUNW.Event(5)</code> 中介绍了 <code>cl_apid</code> 守护程序。 缺省值为 3。</p>	Y	可选的

表 A-1 资源类型特性 (续)

特性名称	说明	是否可以更新?	类别
Client_retry_interval (整数)	定义 cl_apid 守护程序在与未响应的外部客户机进行通信时所用的时间期限 (以秒计算)。在此间隔内最多可以进行 Client_retry_count 次尝试来连接该客户机。cl_apid 守护程序在 SUNW.Event(5) 中进行说明。 缺省值为 1800。	Y	可选的
Client_timeout (整数)	在与外部客户机通信时 cl_apid 守护程序使用的超时值 (以秒计算)。但是, cl_apid 守护程序连续尝试连接该客户机的次数可变。要了解更多关于可用于调节该特性的方法, 请参阅 Client_retry_count 和 Client_retry_interval 特性。cl_apid 守护程序在 SUNW.Event(5) 中进行说明。 缺省值为 60。	Y	可选的
Deny_hosts (字符串数组)	控制一组客户机, 不允许该组客户进行登记来接收群集重新配置事件。确定权限时, 该特性的设置优先于 Allow_hosts 列表中的其它特性的设置。此特性的格式与 Allow_hosts 特性中定义的格式相同。此特性可以为 CRNP 提供保护。 缺省值为 NULL。	Y	可选的
Failover (布尔值)	True 表示可以在多个节点上联机的任何组内无法配置此类型资源。缺省值为 False。	N	可选的
Fini (字符串)	一种可选的回叫方法: 从 RGM 管理中删除此类型的资源时, RGM 所调用的程序的路径。	N	有条件的/显式
Init (字符串)	一种可选的回叫方法: 当此类型的资源开始处于 RGM 的管理之下时, RGM 所调用的程序的路径。	N	有条件的/显式
Init_nodes (枚举值)	值可以是 RG primaries (仅适用于可以控制资源的节点) 或 RT installed_nodes (适用于安装了此资源类型的所有节点)。表明 RGM 对哪些节点调用 Init、Fini、Boot 和 Validate 方法。 缺省值为 RG primaries。	N	可选的
Installed_nodes (字符串数组)	可以在其上运行该资源类型的群集节点的名称列表。RGM 将自动创建此特性。群集管理员可以设置此值。不能在 RTR 文件中声明该特性。 缺省值是所有群集节点。	Y	可以由群集管理员进行配置

表 A-1 资源类型特性 (续)

特性名称	说明	是否可以更新?	类别
Max_clients (整数)	控制客户机的最大数量, 这些客户机可以通过 cl_apid 守护程序进行登记, 以接收群集事件通知。其它客户机为接收事件而进行的登记尝试将被您的应用程序拒绝。由于每个客户机登记中都使用群集上的资源, 则调节此特性使用户可以控制外部客户机对群集上的资源的使用。cl_apid 守护程序在 SUNW.Event(5) 中进行说明。 缺省值为 1000。	Y	可选的
Monitor_check (字符串)	一种可选的回叫方法: 在对此类型的资源执行监视器所请求的失效转移之前, RGM 调用的程序的路径。	N	有条件的/显式
Monitor_start (字符串)	一种可选的回叫方法: 为启动该类型资源的缺省监视器, RGM 调用的程序的路径。	N	有条件的/显式
Monitor_stop (字符串)	设置 Monitor_start 后需要的一种回叫方法: 为停止该类型资源的缺省监视器, RGM 调用的程序的路径。	N	有条件的/显式
每个群集节点上的 Num_resource_restarts (整数)	RGM 将此特性设置为: 在过去的 $n$ 秒内在此节点上对该资源调用 scha_control RESTART 命令的次数, 其中 $n$ 是资源的 Retry_interval 特性的值。如果某个资源类型没有声明 Retry_interval 特性, 则 Num_resource_restarts 特性不可用于此类型资源。	N	仅限于查询
Pkglist (字符串数组)	包含在资源类型安装中的软件包的可选列表。	N	有条件的/显式
Postnet_stop (字符串)	一种可选的回叫方法: 在调用该类型资源所依赖的任何网络地址资源 (Network_resources_used) 的 Stop 方法后, RGM 调用的程序的路径。此方法应该执行 STOP 操作, 在将网络接口配置为关闭之后必须执行该操作。	N	有条件的/显式
Prenet_start (字符串)	一种可选的回叫方法: 在调用该类型资源所依赖的任何网络地址资源 (Network_resources_used) 的 Start 方法前, RGM 调用的程序的路径。此方法应该执行 START 操作, 在将网络接口配置为启用之前必须执行该操作。	N	有条件的/显式

表 A-1 资源类型特性 (续)

特性名称	说明	是否可以更新?	类别
Resource_type (字符串)	<p>资源类型的名称。要查看当前已登记的资源类型的名称, 请使用:</p> <pre>scrgadm -p</pre> <p>在 Sun Cluster 3.1 及更高版本中, 资源类型名称必须包含版本:</p> <pre>vendor_id.resource_type:version</pre> <p>资源类型名称的三个组成部分是 RTR 文件中指定的以下特性: <i>Vendor_id</i>、<i>Resource_type</i> 和 <i>RT_version</i>。scrgadm 命令将插入句点和冒号分界符。资源类型名称中的 <i>RT_version</i> 后缀的值与 <i>RT_version</i> 特性的值相同。为了确保 <i>Vendor_id</i> 的唯一性, 建议使用创建该资源类型的公司的股票标志。使用 Sun Cluster 3.1 以前的版本创建的资源类型名称将继续使用以下语法:</p> <pre>vendor_id.resource_type</pre> <p>缺省值为空字符串。</p>	N	必需的
RT_basedir (字符串)	<p>用于完成回叫方法的相对路径的目录路径。此路径将设置为安装资源类型软件包的位置。它必须是完整的路径, 即, 必须以正斜杠 (/) 开始。如果所有的方法路径名称都是绝对路径, 则此特性不是必需的。</p>	N	必需的, 除非所有方法路径名称均为绝对路径
RT_description (字符串)	<p>资源类型的简单说明。</p> <p>缺省值为空字符串。</p>	N	有条件的
RT_version (字符串)	<p>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。从 Sun Cluster 3.1 开始, RT_version 特性已成为必需的特性 (在 Sun Cluster 3.0 中为可选特性)。</p>	N	有条件的/显式
Single_instance (布尔值)	<p>True 表示在群集中只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。</p> <p>缺省值为 False。</p>	N	可选的
Start (字符串)	<p>一种回叫方法: 为启动此类型的资源, RGM 调用的程序的路径。</p>	N	必需的 (除非在 RTR 文件中声明了 <i>Prenet_start</i> 方法)

表 A-1 资源类型特性 (续)

特性名称	说明	是否可以更新?	类别
Stop (字符串)	一种回叫方法: 为停止此类型的资源, RGM 调用的程序的路径。	N	必需的 (除非在 RTR 文件中声明了 Postnet_stop 方法)
Update (字符串)	一种可选的回叫方法: 当更改此类型的运行资源的特性时, RGM 所调用的程序的路径。	N	有条件的/显式
Validate (字符串)	一种可选的回叫方法: 为检查此类型的资源的特性值而调用的程序的路径。	N	有条件的/显式
Vendor_ID (字符串)	请参阅 Resource_type 特性。	N	有条件的

## 资源特性

表 A-2 中介绍了 Sun Cluster 定义的资源特性。特性值分为以下几类 (在“类别”列中) :

- **必需的** — 管理员必须在使用管理公用程序创建资源时指定一个值。
- **可选的** — 如果管理员在创建资源组时未指定值, 系统会提供缺省值。
- **有条件的** — 仅当在 RTR 文件中声明该特性之后, RGM 才会创建该特性, 否则, 该特性将不存在, 系统管理员将不能使用该特性。在 RTR 文件中声明的有条件特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, 请参阅对每个有条件的特性的说明。
- **仅限于查询** — 不能通过管理工具直接进行设置。

表 A-2 还列出了是否以及何时您可以更新资源类型 (在“是否可以更新?”列中), 如下所示:

None 或 False	从不
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

表 A-2 资源特性

特性名称	说明	是否可以更新?	类别
Affinity_timeout (整数)	<p>以秒计算的时间长度，在这段时间从给定客户机 IP 地址指向资源中的任意服务的连接将发送到同一服务器节点。</p> <p>仅当 Load_balancing_policy 的值为 Lb_sticky 或 Lb_sticky_wild 时，此特性才适用。此外，Weak_affinity 必须设置成 false (缺省值)。</p> <p>此特性只用于可伸缩服务。</p>	任何时候	可选的
Cheap_probe_interval (整数)	<p>在两次资源故障快速探测的调用之间的秒数。此特性仅可由 RGM 创建，在 RTR 文件中声明后，管理员可以使用该特性。</p> <p>如果在 RTR 文件中指定了缺省值，则此特性是可选的。如果未在资源类型文件中指定 Tunable 属性，则该特性的 Tunable 值为 When_disabled。</p> <p>如果未在 RTR 文件的特性声明中指定 Default 属性，则此特性为必需的。</p>	禁用时	有条件的
扩展特性	<p>在资源类型的 RTR 文件中声明的扩展特性。资源类型的实现中定义了这些特性。有关可为扩展特性设置的各个属性的信息，请参阅表 A-4。</p>	取决于特定的特性	有条件的
Failover_mode (枚举值)	<p>可能的设置为 NONE、SOFT 和 HARD。控制 RGM 是重新定位资源组还是中断节点来响应资源调用 Start、Stop 或 Monitor_stop 方法时发生的失败。NONE 表示 RGM 在方法失败时只需设置资源状态并等待操作员的干预。SOFT 表示 Start 方法的失败可能导致 RGM 将资源的组重新定位到另一个节点，而 Stop 或 Monitor_stop 方法的失败可能导致 RGM 将资源设置成 STOP_FAILED 状态，同时将资源组设置成 ERROR_STOP_FAILED 状态，并等待操作人员的干涉。对于 Stop 或 Monitor_stop 失败，NONE 和 SOFT 设置等效。HARD 表示 Start 方法的失败可能导致组的重新定位，通过中断群集节点来响应 Stop 或 Monitor_stop 方法的失败可能导致资源被强制停止。</p> <p>缺省值为 NONE。</p>	任何时候	可选的

表 A-2 资源特性 (续)

特性名称	说明	是否可以更新?	类别
Load_balancing_policy (字符串)	<p>定义所使用的负载平衡策略的字符串。此特性仅用于可伸缩服务。如果在 RTR 文件中声明了 Scalable 特性, 则 RGM 自动创建此特性。Load_balancing_policy 可以采用以下值:</p> <p>Lb_weighted (缺省值)。根据在 Load_balancing_weights 特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。</p> <p>Lb_sticky_wild。连接到通配符粘滞服务的 IP 地址的给定客户机 (由客户机的 IP 地址标识) 总是发送到同一群集节点, 而不管它来自哪个端口号。</p> <p>缺省值为 Lb_weighted。</p>	创建时	有条件的/ 可选的
Load_balancing_weights (字符串数组)	<p>只用于可伸缩资源。如果在 RTR 文件中声明了 Scalable 特性, 则 RGM 将自动创建此特性。其格式为 <i>weight@node,weight@node</i>, 其中 <i>weight</i> 是一个反映分配到指定 <i>node</i> 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重除以所有权重的和。例如, 1@1,3@2 表示节点 1 接收负载的 1/4, 节点 2 接收 3/4。空字符串 ("") (缺省值) 表示均匀分布。未指定显式权重的节点将接收到缺省权重 1。</p> <p>如果未在资源类型文件中指定 Tunable 属性, 则特性的 Tunable 值为 Anytime。更改该特性将仅改变新连接的分配。</p> <p>缺省值是空字符串 ("")。</p>	任何时候	有条件的/ 可选的
Type (整数) 中每个回叫方法的 <i>method_timeout</i>	<p>以秒计算的时间段, 这段时间过后, RGM 将认为对方法的调用已失败。</p> <p>如果在 RTR 文件中声明了该方法本身, 则缺省值为 3,600 (一个小时)。</p>	任何时候	有条件的/ 可选的
Monitored_switch (枚举值)	<p>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 将把此特性设置为 Enabled 或 Disabled。如果设置为 Disabled, 则在再次启用该监视器之前, 不会调用监视器的 Start 方法。如果资源没有监视器回叫方法, 则此特性不存在。</p> <p>缺省值为 Enabled。</p>	从不	仅限于查询

表 A-2 资源特性 (续)

特性名称	说明	是否可以更新?	类别
Network_resources_used (字符串数组)	<p>资源所使用的逻辑主机名或共享地址网络资源的列表。对于可伸缩服务, 此特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, 此特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 特性, 则 RGM 将自动创建此特性。如果在 RTR 文件中未声明 Scalable, Network_resources_used 将不可用, 除非在 RTR 文件中显式声明。</p> <p>如果未在资源类型文件中指定 Tunable 属性, 则该特性的 Tunable 值为 At_creation。</p>	创建时	有条件的/必需的
On_off_switch (枚举值)	<p>如果群集管理员通过管理公用程序启用或禁止资源, 则 RGM 将把此特性设置成 Enabled 或 Disabled。如果被禁止, 则在再次启用资源之前, 资源的回叫不会被调用。</p> <p>缺省值为 Disabled。</p>	从不	仅限于查询
Port_list (字符串数组)	<p>端口 (服务器在其上进行侦听) 号列表。每个端口号后接此端口使用的协议, 例如, Port_list=80/tcp。如果在 RTR 文件中声明了 Scalable 特性, 则 RGM 将自动创建 Port_list。否则, 此特性将不可用, 除非在 RTR 文件中显式声明。</p> <p>在《用于 Apache 的 Sun Cluster 数据服务指南 (适用于 Solaris OS)》中介绍了为 Apache 设置此特性的信息。</p>	创建时	有条件的/必需的
R_description (字符串)	<p>资源的简单说明。</p> <p>缺省值为空字符串。</p>	任何时候	可选的
Resource_name (字符串)	<p>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。</p>	从不	必需的

表 A-2 资源特性 (续)

特性名称	说明	是否可以更新?	类别
Resource_project_name (字符串)	<p>与资源关联的 Solaris 项目名称。使用此特性可将 Solaris 资源管理功能 (例如 CPU 共享和资源池) 应用至群集数据服务。当 RGM 使资源联机时, 它将启动此项目名下的相关进程。如果未指定此特性, 将从包含该资源的资源组的 RG_project_name 特性获取项目名称 (请参阅 rg_properties [5])。如果也未指定此特性, RGM 将使用预定义的项目名称 default。指定的项目名称必须存在于项目数据库中, 并且用户 root 必须配置为已命名项目的成员。仅从 Solaris 9 开始支持此特性。</p> <p><b>注意</b> - 对此特性的更改将在资源重新启动后生效。</p> <p>缺省值为 null。</p>	任何时候	可选的
每个群集节点上的 Resource_state (枚举值)	<p>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、Stop_failed、Start_failed、Monitor_failed 以及 Online_not_monitored。</p> <p>用户不能配置此特性。</p>	从不	仅限于查询
Retry_count (整数)	<p>当资源发生故障时, 监视器试图重新启动资源的次数。此特性仅可由 RGM 创建, 在 RTR 文件中声明后, 管理员可以使用该特性。如果在 RTR 文件中指定了缺省值, 则此特性是可选的。</p> <p>如果在资源类型文件中未指定 Tunable 属性, 此属性的 Tunable 值为 When_disabled。</p> <p>如果未在 RTR 文件的特性声明中指定 Default 属性, 则此特性为必需的。</p>	禁用时	有条件的
Retry_interval (整数)	<p>尝试重新启动失败的资源前计算的秒数。资源监视器将此特性与 Retry_count 一起使用。此特性仅可由 RGM 创建, 在 RTR 文件中声明后, 管理员可以使用该特性。如果在 RTR 文件中指定了缺省值, 则此特性是可选的。</p> <p>如果在资源类型文件中未指定 Tunable 属性, 此属性的 Tunable 值为 When_disabled。</p> <p>如果未在 RTR 文件的特性声明中指定 Default 属性, 则此特性为必需的。</p>	禁用时	有条件的

表 A-2 资源特性 (续)

特性名称	说明	是否可以更新?	类别
Scalable (布尔值)	<p>表明资源是否可伸缩。如果在 RTR 文件中声明了此特性, 则 RGM 将为该类型的资源自动创建以下可伸缩服务特性: Network_resources_used、Port_list、Load_balancing_policy 和 Load_balancing_weights。这些特性具有缺省值, 除非在 RTR 文件中对它们进行了显式声明。Scalable 的缺省值 (在 RTR 文件中声明时) 为 True。</p> <p>在 RTR 文件中声明此特性时, Tunable 属性必须设置为 At_creation, 否则资源创建将失败。</p> <p>如果在 RTR 文件中未声明此特性, 则资源不可伸缩; 群集管理员无法调整此特性并且 RGM 无法设置可伸缩服务特性。但是, 必要时您可以在 RTR 文件中显式声明 Network_resources_used 和 Port_list 特性; 因为它们在非可伸缩服务中与在可伸缩服务中都可能有用。</p>	创建时	可选的
每个群集节点上的 Status (枚举值)	<p>由资源监视器设置。值可能为: OK、degraded、faulted、unknown 和 offline。当使资源联机时, RGM 将把值设置成 unknown; 使资源脱机时, 将把值设置成 Offline。</p>	从不	仅限于查询
每个群集节点上的 Status_msg (字符串)	<p>由资源监视器在设置 Status 特性的同时进行设置。可以为每个资源每个节点都设置此特性。使资源脱机时, RGM 将把它设置成空字符串。</p>	从不	仅限于查询
Thorough_probe_interval (整数)	<p>在两次资源高开销故障探测的调用之间的秒数。此特性仅可由 RGM 创建, 在 RTR 文件中声明后, 管理员可以使用该特性。如果在 RTR 文件中指定了缺省值, 则此特性是可选的。</p> <p>如果未在资源类型文件中指定 Tunable 属性, 则该特性的 Tunable 值为 When_disabled。</p> <p>如果未在 RTR 文件的特性声明中指定 Default 属性, 则此特性为必需的。</p>	禁用时	有条件的
Type (字符串)	<p>资源类型, 此资源是该资源类型的一个实例。</p>	从不	必需的

表 A-2 资源特性 (续)

特性名称	说明	是否可以更新?	类别
Type_version (字符串)	<p>指定当前与此资源关联的资源类型的版本。RGM 将自动创建此特性，该特性不能在 RTR 文件中声明。此特性的值与资源类型的 RT_version 特性一样。创建资源时，不明确指定 Type_version 特性，尽管它可能作为资源类型名称的后缀出现。编辑资源时，可以将 Type_version 更改为新值。</p> <p>其可调性源自：</p> <ul style="list-style-type: none"> <li>■ 资源类型的当前版本</li> <li>■ RTR 文件中的 #upgrade_from 指令</li> </ul>	请参阅说明	请参阅说明
UDP_affinity (布尔值)	<p>如果为 true，则来自给定客户机的所有 UDP 通信都将发送至当前处理该客户机的所有 TCP 通信的同一服务器节点。</p> <p>仅当 Load_balancing_policy 的值是 Lb_sticky 或 Lb_sticky_wild 时，此特性才适用。此外，Weak_affinity 必须设置为 False (缺省值)。</p> <p>此特性只用于可伸缩服务。</p>	禁用时	可选的
Weak_affinity (布尔值)	<p>如果为 true，则启用较弱的客户机亲合性。这可以将来自给定客户机的连接发送到同一服务器节点，但以下情况例外：</p> <ul style="list-style-type: none"> <li>■ 由于种种原因（例如，重新启动故障监视器、资源的故障切换和切换或者节点在失败后重新连接群集）而启动服务器侦听程序。</li> <li>■ 由于管理操作而导致可伸缩资源的 Load_balancing_weights 更改。</li> </ul> <p>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亲合性所使用的系统开销较低。</p> <p>仅当 Load_balancing_policy 的值为 Lb_sticky 或 Lb_sticky_wild 时，此特性才适用。</p> <p>此特性只用于可伸缩服务。</p>	禁用时	可选的

## 资源组特性

下表介绍了 Sun Cluster 定义的资源组特性。特性值分为以下几类（在“类别”列中）：

- **必需的** — 管理员必须在使用管理公用程序创建资源组时指定一个值。
- **可选的** — 如果管理员在创建资源组时未指定值，系统会提供缺省值。
- **仅限于查询** — 不能通过管理工具直接进行设置。

“是否可以更新？”列中说明了特性在进行初始设置后是否可以更新。“Y”表示是，“N”表示否。

表 A-3 资源组特性

特性名称	说明	是否可以更新？	类别
Auto_start_on_new_cluster (布尔值)	在新的群集形成时，此特性不允许自动启动资源组。 缺省值为 TRUE。如果将其设置为 TRUE，则资源组管理器会在重新引导群集时尝试自动启动资源组，以实现 Desired primaries。如果设置为 FALSE，当群集节点重新启动时，资源组无法自动启动。	Y	可选的
Desired primaries (整数)	希望使组同时在其上处于联机状态的节点数量。 缺省值为 1。如果 RG_mode 特性为 Failover，则此特性的值必须不大于 1。如果 RG_mode 特性为 Scalable，则允许值大于 1。	Y	可选的
Failback (布尔值)	一个布尔值，表明当群集成员资格更改时，是否重新计算组联机的节点集。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。 缺省值为 False。	Y	可选的
Global_resources_used (字符串数组)	表明群集文件系统是否由此资源组中的任何资源使用。管理员可以指定的合法值为表示所有全局资源的星号 (*) 和表示非全局资源的空字符串 ("")。 缺省值为所有全局资源。	Y	可选的
Implicit_network_dependencies (布尔值)	一个布尔值，如果为 True，则表明 RGM 将把非网络地址资源固有的强大依赖性强加到组内的网络地址资源上。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，此特性不会产生影响，因为可伸缩资源组不包含任何网络地址资源。 缺省值为 True。	Y	可选的

表 A-3 资源组特性 (续)

特性名称	说明	是否可以更新?	类别
Maximum primaries (整数)	组一次可以联机的最大节点数。 缺省值为 1。如果 RG_mode 特性为 Failover, 则此特性的值必须不大于 1。如果 RG_mode 特性为 Scalable, 则允许值大于 1。	Y	可选的
Nodelist (字符串数组)	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。 缺省值为所有群集节点的列表。	Y	可选的
Pathprefix (字符串)	群集文件系统的一个目录, 在此目录中组内的资源可以写基本的管理文件。某些资源可能需要此特性。使 Pathprefix 对每个资源组都是唯一的。 缺省值为空字符串。	Y	可选的
Pingpong_interval (整数)	RGM 使用非负整数 (以秒计算) 确定: 如果重新配置, 或者执行 scha_control -O GIVEOVER 命令或执行带有 SCHA_GIVEOVER 参数的 scha_control() 函数, 何时将资源组联机。 重新配置时, 如果资源组在过去的 Pingpong_interval 秒内无法在某个特定节点上联机 (因为资源的 Start 或 Preinet_start 方法以非零状态退出或超时), 则认为此节点不是使用该资源组的合法节点, RGM 将寻找其它主节点。 如果调用资源的 scha_control 命令或 scha_control() 函数导致资源组在过去的 Pingpong_interval 秒内在某个特定节点脱机, 由于后来调用另一个节点发生的 scha_control (), 此节点不是使用该资源组的合法节点。 缺省值为 3,600 (一个小时)。	Y	可选的
Resource_list (字符串数组)	此组中包含的资源的列表。管理员不会直接设置该特性。而是当管理员向/从资源组中添加/删除资源组时, 由 RGM 更新此特性。 缺省值为空列表。	N	仅限于查询
RG_description (字符串)	资源组的简单说明。 缺省值为空字符串。	Y	可选的

表 A-3 资源组特性 (续)

特性名称	说明	是否可以更新?	类别
RG_mode (枚举值)	<p>表明资源组是故障切换组还是可伸缩组。如果值为 Failover, RGM 将组的 Maximum primaries 特性设置为 1, 并将资源组限制成由单一节点控制。</p> <p>如果此特性的值为 Scalable, 则 RGM 将允许 Maximum primaries 特性的值大于 1, 意味着该组可由多个节点同时控制。RGM 不允许将其 Failover 特性的值为 True 的资源添加到其 RG_mode 的值为 Scalable 的资源组中。</p> <p>如果 Maximum primaries 的值是 1, 则缺省值是 Failover; 如果 Maximum primaries 的值大于 1, 则缺省值是 Scalable。</p>	N	可选的
RG_name (字符串)	<p>资源组的名称。该名称在群集内必须是唯一的。</p>	N	必需的
RG_project_name (字符串)	<p>与资源组关联的 Solaris 项目名称。使用此特性可将 Solaris 资源管理功能 (例如 CPU 共享和资源池) 应用至群集数据服务。当 RGM 使资源组联机时, 它将为没有设置 Resource_project_name 特性的资源启动此项目名称下的相关进程。指定的项目名称必须存在于项目数据库中, 而且必须将用户 root 配置为已命名项目的成员。</p> <p>仅从 Solaris 9 开始支持此特性。</p> <p><b>注意</b> - 对此特性的更改将在资源重新启动后生效。</p>	任何时候	必需的
每个群集节点上的 RG_state (枚举值)	<p>由 RGM 设置为 Online、Offline、Pending_online、Pending_offline、Pending_online_blocked、Error_stop_failed 或 Online_faulted, 用于描述组在每一个群集节点上的状态。</p> <p>用户不能配置此特性。但是您可以通过调用 scswitch(1M) (或通过使用等效的 scsetup(1M) 或 SunPlex Manager 命令) 来间接设置此特性。</p>	N	仅限于查询

## 资源特性属性

下表说明了可用来更改系统定义的特性或用来创建扩展特性的资源特性属性。



**注意** - 您不能将 NULL 或空字符串 ("") 指定为 boolean、enum 或 int 类型的缺省值。

表 A-4 资源特性属性

特性	说明
Property	资源特性的名称。
Extension	如果使用，则表明 RTR 文件项声明了一个由资源类型的实现定义的扩展特性，否则，此项为系统定义的特性。
Description	一个字符串注释，用于对特性作简单说明。对于系统定义的特性，不能在 RTR 文件中设置说明属性。
特性的类型	允许的类型为：string、boolean、int、enum 和 stringarray。对于系统定义的特性，您不能在 RTR 文件项中设置类型属性。类型决定了可接受的特性值和可在 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。
Default	表示此特性的缺省值。
Tunable	表示群集管理员何时可以设置资源中此特性的值。可以设置为 None 或 False，以防止管理员设置此特性。允许管理员调节的值：True 或 Anytime（任何时候）、At_creation（仅在创建资源的时）或 When_disabled（当资源脱机时）。 缺省值是 True (Anytime)。
Enumlist	对于 enum 类型，为该特性所允许的一组字符串值。
Min	对于 int 类型，为该特性所允许的最小值。
Max	对于 int 类型，为此特性所允许的最大值。
Minlength	对于 string 和 stringarray 类型，为允许的最小字符串长度。
Maxlength	对于 string 和 stringarray 类型，为允许的最大字符串长度。
Array_minsize	对于 stringarray 类型，为允许的最小数组元素数。
Array_maxsize	对于 stringarray 类型，为允许的最大数组元素数。

## 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，并且列出了资源类型登记文件的内容。

本附录中包含以下代码列表。

- 第 221 页 “资源类型登记文件列表”
- 第 224 页 “Start 方法”
- 第 227 页 “Stop 方法”
- 第 229 页 “gettime 公用程序”
- 第 230 页 “PROBE 程序”
- 第 235 页 “Monitor\_start 方法”
- 第 237 页 “Monitor\_stop 方法”
- 第 238 页 “Monitor\_check 方法”
- 第 240 页 “Validate 方法”
- 第 244 页 “Update 方法”

---

## 资源类型登记文件列表

RTR（资源类型登记）文件包含群集管理员在登记数据服务时用来定义数据服务的初始配置的资源及资源类型特性声明。

实例 B-1 SUNW.Sample RTR 文件

```
#  
# Copyright (c) 1998-2004 by Sun Microsystems, Inc.  
# All rights reserved.  
#  
# Registration information for Domain Name Service (DNS)  
#
```

实例 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;
}
```

实例 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 = "";
}

#
```

实例 B-1 SUNW.Sample RTR 文件 (续)

```
# 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 = 30;
    TUNABLE = ANYTIME;
    DESCRIPTION = "Time out value for the probe (seconds)";
}
```

---

## Start 方法

当在群集节点使包含数据服务资源的资源组联机时或当启用该资源时，RGM 对该节点调用 Start 方法。在应用程序样例中，Start 方法用来激活该节点上的 in.named (DNS) 守护程序。

实例 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"
```

实例 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 "$@"
```

实例 B-2 dns\_svc\_start 方法 (续)

```
PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# 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 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}'`

# 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_NAME`

# 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_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.
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"
```

实例 B-2 dns\_svc\_start 方法 (续)

```
fi
exit 0
```

---

## Stop 方法

当包含 HA-DNS 资源的资源组在群集节点脱机时或资源被禁用时，对该节点调用 Stop 方法。此方法将停止该节点上的 in.named (DNS) 守护程序。

实例 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}] \

```

实例 B-3 dns\_svc\_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=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_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_NAME`

# 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] \
```

### 实例 B-3 dns\_svc\_stop 方法 (续)

```
        "${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}.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 公用程序

gettime 公用程序是一个 C 程序，PROBE 程序用于跟踪两次重新启动探测之间的时间间隔。您必须编译此程序并将其放置在与回叫方法相同的目录中，即，由 RT\_basedir 特性指定此目录。

### 实例 B-4 gettime.c 公用程序

```
#
# 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

PROBE 程序使用 nslookup(1M) 命令检查数据服务的可用性。Monitor\_start 回叫方法用来启动此程序，Monitor\_stop 回叫方法用来停止该程序。

实例 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
    done
}
}
```

实例 B-5 dns\_probe 程序 (续)

```
#####
# 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_NAME?
        STOP_METHOD=`scha_resource_get -O STOP \
            -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME?
        hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
            -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
            -T $RESOURCE_TYPE_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_NAME?
        START_METHOD=`scha_resource_get -O START \
            -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME?
        hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
            -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
            -T $RESOURCE_TYPE_NAME

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

    else
        # The absence of the TAG for the dataservice
        # implies that the dataservice has already
        # exceeded the maximum retries allowed under PMF.
    fi
}
#####
```

实例 B-5 dns\_probe 程序 (续)

```
        # 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_time`
        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
        elif [ $retries -ge $RETRY_COUNT ]; then
```

实例 B-5 dns\_probe 程序 (续)

```
# 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_NAME?

# 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}'`
```

实例 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_NAME?

# 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_NAME?

# 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_NAME?

# 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_NAME?

# 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
else
```

实例 B-5 dns\_probe 程序 (续)

```
        if [ $SERVER != $DNS_HOST ]; then
            probefail=1
        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 [ $probfail -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

此方法启动数据服务的 PROBE 程序。

实例 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

    while getopts 'R:G:T:' opt
```

实例 B-6 dns\_monitor\_start 方法 (续)

```
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_NAME`

# 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 \
    -T $RESOURCETYPE_NAME
```

实例 B-6 dns\_monitor\_start 方法 (续)

```
# 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

此方法停止数据服务的 PROBE 程序。

实例 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}] \
                    "ERROR: Option $OPTARG unknown"
        esac
    done
}
```

实例 B-7 dns\_monitor\_stop 方法 (续)

```
        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=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCE_GROUP_NAME,$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

此方法检验 Confdir 特性指定的目录是否存在。每当 PROBE 方法未能将数据服务转移到新节点时或检查作为潜在主节点的节点时，RGM 都将调用 Monitor\_check。

实例 B-8 dns\_monitor\_check 方法

```
#!/bin/ksh#
# Monitor check Method for DNS.
#
```

实例 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`

# Parse the arguments that have been passed to this method.
```

### 实例 B-8 dns\_monitor\_check 方法 (续)

```
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_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_NAME?`

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

# 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 $RESOURCE_TYPE_NAME -x Confdir=$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

此方法检验 Confdir 特性指定的目录是否存在。当创建数据服务以及群集管理员更新数据服务特性时，RGM 都将调用此方法。每当缺省监视器未能将数据服务转移到新节点时，Monitor\_check 方法将调用此方法。

### 实例 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
```

实例 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
```

实例 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

RGM 调用 Update 方法通知正在运行的资源其特性已更改。

实例 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
```

实例 B-10 dns\_update 方法 (续)

```
# 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=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCE_TYPE_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_NAME`

# 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 $RESOURCE_TYPE_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

# 数据服务开发库资源类型代码列表样例

本附录列出了 SUNW.xfnts 资源类型中的每一种方法的完整代码。其中包括 xfnts.c 的列表，该列表包含回叫方法所调用的子例行程序的代码。本附录中的代码列表如下。

- 第 247 页 “xfnts.c”
- 第 259 页 “xfnts\_monitor\_check 方法”
- 第 260 页 “xfnts\_monitor\_start 方法”
- 第 261 页 “xfnts\_monitor\_stop 方法”
- 第 262 页 “xfnts\_probe 方法”
- 第 265 页 “xfnts\_start 方法”
- 第 266 页 “xfnts\_stop 方法”
- 第 267 页 “xfnts\_update 方法”
- 第 269 页 “xfnts\_validate 方法的代码列表”

## xfnts.c

此文件用来实现 SUNW.xfnts 方法调用的子例行程序。

实例 C-1 xfnts.c

```
/*
 * Copyright (c) 1998-2004 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
 * success or failure. Action is taken based on this returned value in the
 * method found in the file xfnts_probe.c
 */
```

实例 C-1 xfnts.c (续)

```
*/

#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():
 *
 * Do HA-XFS specific validation of the resource configuration.
 */
```

实例 C-1 xfnts.c (续)

```
*
* 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
         * is missing void arg
         */
    }
}
```

实例 C-1 xfnts.c (续)

```
    */
    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.");
    rc = 1;
}
```

实例 C-1 xfnts.c (续)

```
        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/xfns -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,resourceinstance,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;

    /* get the configuration directory from the confdir_list property */
```

实例 C-1 xfnts.c (续)

```
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.
 */
```

实例 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.");
        return (1);
    }
}
```

实例 C-1 xfnts.c (续)

```
}

/*
 * 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);
    }

    /* We rely on RGM to timeout and terminate the program */

```

实例 C-1 xfnts.c (续)

```
    } 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
mon_stop(scds_handle_t scds_handle)
```

实例 C-1 xfnts.c (续)

```
{
    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;
    hrttime_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
     * in probe_timeout property before concluding that the probe failed.
     */
}
```

实例 C-1 xfnts.c (续)

```
*/

/*
 * 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"
```

实例 C-1 xfnts.c (续)

```
        " %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);
```

实例 C-1 xfnts.c (续)

```
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

此方法用来检验基本资源类型的配置是否有效。

实例 C-2 xfnts\_monitor\_check.c

```
/*
 * Copyright (c) 1998-2004 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.");
        return (1);
    }
}
```

实例 C-2 xfnts\_monitor\_check.c (续)

```
    }

    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

此方法用来启动 xfnts\_probe 方法。

实例 C-3 xfnts\_monitor\_start.c

```
/*
 * Copyright (c) 1998-2004 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;
```

实例 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

此方法用来停止 xfnts\_probe 方法。

实例 C-4 xfnts\_monitor\_stop.c

```
/*
 * Copyright (c) 1998-2004 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;
```

实例 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

xfnts\_probe 方法用来检查应用程序的可用性并决定是进行故障切换还是重启数据服务。xfnts\_monitor\_start 回叫方法用来启动此程序，而 xfnts\_monitor\_stop 回叫方法用来停止它。

实例 C-5 xfnts\_probe.c+

```
/*
 * Copyright (c) 1998-2004 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():
 * Just an infinite loop which sleep()s for sometime, waiting for
```

实例 C-5 xfnets\_probe.c+ (续)

```
* 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;

    hrttime_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.
        */

```

实例 C-5 xfnets\_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 */
}
```

---

## xfnts\_start 方法

当在群集节点使包含数据服务资源的资源组联机时或当启用该资源时，RGM 对该节点调用 Start 方法。xfnts\_start 方法将激活该节点上的 xfs 守护程序。

### 实例 C-6 xfnts\_start.c

```
/*
 * Copyright (c) 1998-2004 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) {
        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);
```

实例 C-6 xfnts\_start.c (续)

```
    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

使包含 HA-XFS 资源的资源组在群集节点上脱机时或者禁用该资源时，RGM 将对该节点调用 Stop 方法。此方法将停止该节点上的 xfs 守护程序。

实例 C-7 xfnts\_stop.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_stop.c - Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
```

### 实例 C-7 xfnets\_stop.c (续)

```
* 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);
}
```

---

## xfnests\_update 方法

RGM 调用 Update 方法通知运行资源其特性已被更改。在通过管理操作成功地设置了资源及其组的特性后，RGM 将调用 Update。

### 实例 C-8 xfnests\_update.c

```
#pragma ident "@(#)xfnests_update.c 1.10      01/01/18 SMI"

/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnests_update.c - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>
```

实例 C-8 xfnts\_update.c (续)

```
/*
 * 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 方法的代码列表

此方法用来检验 Confdir\_list 特性所指向的目录是否存在。当创建数据服务以及集群管理员更新数据服务特性时，RGM 都将调用此方法。每当故障监视器将数据服务故障切换到新节点时，Monitor\_check 方法都将调用此方法。

### 实例 C-9 xfnts\_validate.c

```
/*
 * Copyright (c) 1998-2004 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);
}
```



# 标准的 RGM 名称和值

---

本附录列出了对 RGM 名称和值所用的标准字符的要求。

---

## RGM 标准名称

RGM 名称分为五类：

- 资源组名称
- 资源类型名称
- 资源名称
- 特性名称
- 枚举文字名称

除了资源类型名称之外，所有名称都必须符合以下规则：

- 必须是 ASCII 码。
- 必须以字母开头。
- 可以包含大写和小写字母、数字、破折号 (-) 和下划线 (\_)。
- 不能超过 255 个字符。

资源类型名称可以是简单的名称（在 RTR 文件中由 `Resource_type` 特性指定）或者是完整的名称（在 RTR 文件中由 `Vendor_id` 和 `Resource_type` 特性指定）。当您指定这些特性时，RGM 将在 `Vendor_id` 和 `Resource_type` 之间添加一个句点组成一个完整名称。例如，如果是 `Vendor_id=SUNW` 和 `Resource_type=sample`，则完整的名称为 `SUNW.sample`。仅在这种情况下句点可以作为标准字符出现在 RGM 名称之中。

---

## RGM 的值

RGM 的值分为两种：特性值和描述值，这两种值都采用同一种规则，如下所示：

- 值必须都是 ASCII 码。
- 值的最大长度是 4 兆位字节减去 1 字节，即 4,194,303 字节。
- 值中不能包含以下任何字符：空字符、换行符、逗号或分号。

# 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)。第 25 页“分析应用程序的适用性”一节中列出了这些要求。本附录提供了有关其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。该应用程序的数据将存储在具有高可用性的全局文件系统中，这样即使某个服务器失败也可以由其它未失败的服务器存取这些数据。有关群集文件系统的信息，请参阅《*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存取，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切换，这样该数据服务的网络客户机可以存取其新主机上的数据服务资源。

---

## 多主机数据

具有高可用性的全局文件系统的磁盘组分布在多个主机上，这样即使某个物理主机崩溃，其它某个未崩溃主机也可以存取该磁盘。对于要具有高可用性的应用程序，其数据必须具有高可用性，因此应用程序的数据必须驻留在全局 HA 文件系统中。

全局文件系统装载在作为独立实体创建的磁盘组上。用户可以选择将一些磁盘组用作已装载的全局文件系统，而将其它磁盘组作为数据服务（例如 HA Oracle）的原始设备。

应用程序可能具有指向数据文件位置的命令行开关或换配置文件。如果该应用程序使用硬链接的路径名，您可以将该路径名更改为指向全局文件系统中的文件的符号链接，而无需更改应用程序代码。有关使用符号链接的详细介绍，请参阅第 274 页“将符号链接用于多主机数据放置”。

在最糟糕的情况下，您不得不修改应用程序的源代码，以提供一些用于指向实际数据位置的机制。您可以通过运行附加命令行开关来执行此操作。

Sun Cluster 支持使用卷管理器中所配置的 UNIX UFS 文件系统和 HA 原始设备。进行安装和配置时，系统管理员必须指定哪些磁盘资源用于 UFS 文件系统，哪些用于原始设备。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

有些时候，应用程序会对数据文件的路径名采用硬链接的方式，而没有用于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，您可以使其成为指向 `/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，假设该应用程序执行更新，方法是首先创建新的临时文件 `/etc/mydatafile.new`，然后通过使用 `rename(2)` 系统调用（或 `mv(1)` 程序）重命名临时文件以获取实际的文件名。通过创建临时文件，然后再对其进行重命名，数据服务试图确保其数据文件的目录始终保持正确的格式。

遗憾的是 `rename(2)` 操作将破坏符号链接。现在，命名为 `/etc/mydatafile` 的文件是一个正规文件，并且与 `/etc` 目录位于相同的文件系统中，而不是位于群集的全局文件系统中。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。

这种情况下的根本问题是现有应用程序不支持符号链接，并且在编写时没有考虑到符号链接的问题。要使用符号链接将数据存取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，符号链接并不能完全解决将数据放置在群集全局文件系统中的问题。

---

## 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。如果回答是肯定的，那么可能需要将该数据服务修改为使用逻辑主机名（即配置为与应用程序资源驻留在同一资源组的逻辑主机名资源的主机名），而不是使用物理主机的名称。

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切换或转移后可用，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”描述的是位于多个公共网络上的主机。这样的主机具有多个主机名和 IP 地址。其中每个网络都对应一个“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）。因为物理主机名具有多个“主机名 - IP 地址”对，所以每一个资源组可以具有多个“主机名 - IP 地址”对，分别与每个公共网络相对应。当 Sun Cluster 将资源组从一个物理主机移动到另一个物理主机时，也将同时移动该资源组的整组“主机名 - IP 地址”对。

将按照包含在该资源组中的逻辑主机名资源配置“主机名 - IP 地址”对组。这些网络地址资源是由系统管理员在创建和配置资源组时指定的。Sun Cluster 数据服务 API 中包含用来查询这些“主机名 - IP 地址”对的工具。

大多数针对 Solaris 操作系统编写的现有数据服务守护程序已经正确地处理了多地址主机。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 有效地绑定到当前为计算机配置的所有 IP 地址上。通常不需要为了处理 Sun Cluster 逻辑网络地址而更改使用 INADDR\_ANY 的数据服务守护程序。

---

## 绑定到 INADDR\_ANY 与绑定到特定的 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。对于计算机来说，会有一个 IP 地址与它自己的物理主机相对应，而其它 IP 地址则与其当前控制的每个网络地址（逻辑主机名）资源相对应。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。

就有些数据服务而言，如果将其绑定到 INADDR\_ANY，它们将无法在 Sun Cluster 环境中正常使用。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程序。

Network\_resources\_used 资源特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，必须在该资源类型的 RTR 文件中声明 Network\_resources\_used 特性。

当 RGM 使资源组联机或脱机时，它将按照特定顺序进行安装、卸载和向上/向下配置网络地址，这与它何时调用数据服务资源方法有关。请参阅第 38 页“决定使用哪种 Start 或 Stop 方法”。

数据服务的 Stop 方法返回时，该数据服务必定已停止使用资源组的网络地址。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。

如果该数据服务的停止和启动方法通过终止和启动该数据服务的守护程序完成了它们的工作，则该数据服务将在适当的时间停止和开始使用网络地址。

---

## 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处理了单个服务器崩溃和重新引导的情况，那么它们将处理该资源组被接管或转移的情况。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一段时间后重试，用户可以选择是否继续进行。

## 附录 F

---

# CRNP 的文档类型定义

---

本附录列出了群集重新配置通知协议 (CRNP) 的 DTD (文档类型定义)。

---

## SC\_CALLBACK\_REG XML DTD

---

注意 - 仅对 SC\_CALLBACK\_REG 和 SC\_EVENT 使用的 NVPAIR 数据结构定义一次。

---

```
<!-- SC_CALLBACK_REG XML format specification
      Copyright 2001-2004 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:<br>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 2001-2004 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 2001-2004 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_RESOURCE, 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          NMOKEN          #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

---

注意 - 仅对 SC\_CALLBACK\_REG 和 SC\_EVENT 使用的 NVPAIR 数据结构定义一次。

---

<!-- SC\_EVENT XML format specification

Copyright 2001-2004 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 syseventd 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

---

本附录显示完整的 CrnpClient.java 应用程序，该程序在第 12 章中进行了详细的讨论。

---

## 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 object, 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 registration 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
     * rname 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
{
    /*
     * Create 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

---

## 数字和符号

- #\$upgrade\_from 指令, 51
  - Anytime, 52
  - At\_creation, 52
  - When\_disabled, 52
  - When\_offline, 52
  - When\_unmanaged, 52
  - When\_unmonitored, 52
  - 可调性值, 52
- \$hostnames 变量, Agent Builder, 144
- “创建”屏幕, Agent Builder, 140
- “配置”屏幕, Agent Builder, 142
- “浏览”按钮, Agent Builder, 152

## A

- Agent Builder
  - \$hostnames 变量, 144
  - “创建”屏幕, 140
  - “配置”屏幕, 142
  - Cluster Agent 模块, 154
    - 区别, 157
  - GDS 输出, 168
  - rtconfig 文件, 151
  - 安装, 138
  - 编辑已生成的源代码, 146
  - 创建使用 GDS 的服务, 164
  - 创建资源类型, 145
  - 二进制文件, 148
  - 分析应用程序, 138
  - 脚本, 149
  - 克隆现有的资源类型, 145

## Agent Builder (续)

- 描述, 18, 22
- 命令行版本, 147
- 目录结构, 147
- 配置, 138
- 启动, 139
- 软件包目录, 150
- 使用, 138
- 使用命令行版本创建使用 GDS 的服务, 170
- 手册页, 149
- 用于创建 GDS, 160, 164
- 源文件, 148
- 支持文件, 150
- 重复使用完成的工作, 145
- 浏览, 151
  - “编辑”菜单, 153
  - “文件”菜单, 153
  - “浏览”按钮, 152
  - 菜单, 153
- Anytime, #upgrade\_from 指令, 52
- API, 资源管理, 请参见 RMAPI
- arraymax, 资源类型迁移, 49
- arraymin, 资源类型迁移, 49
- At\_creation, #upgrade\_from 指令, 52

## B

- Boot 方法, 使用, 40

## C

- C 程序函数, RMAPI, 65
- CCR (群集配置系统信息库), 55
- Cluster Agent 模块
  - Agent Builder 区别, 157
  - 安装, 154
  - 启动, 155
  - 设置, 154
  - 使用, 156
  - 说明, 154
- CRNP
  - Java 应用程序实例, 188
  - SC\_CALLBACK\_REG 消息, 181
  - 错误状态, 184
  - 登记客户机和服务器, 181
  - 服务器, 181
  - 服务器事件传送, 184
  - 服务器应答, 183
  - 函数, 177
  - 鉴别, 187
  - 客户机, 181
  - 客户机标识进程, 181
  - 说明, 177
  - 通信, 178
  - 消息类型, 180
  - 协议, 178
  - 协议语义, 178

## D

- DSDL
  - libdsdev.so, 17
  - PMF (进程监视工具) 函数, 176
  - 存取网络地址, 105
  - 调试资源类型, 106
  - 概述, 17
  - 公用程序函数, 176
  - 故障监视, 175
  - 故障监视器函数, 176
  - 函数, 173
  - 描述, 103
  - 启动数据服务, 104
  - 启用 HA 本地文件系统, 106
  - 实现故障监视器, 105
  - 实现位置, 17
  - 特性函数, 174
  - 停止数据服务, 104

## DSDL (续)

- 通用函数, 173
- 网络资源存取函数, 175
- 资源类型实现样例
  - scds\_initialize() 函数, 119
  - SUNW.xfnts RTR 文件, 119
  - SUNW.xfnts 故障监视器, 127
  - svc\_probe() 函数, 129
  - TCP 端口号, 118
  - X Font Server, 117
  - X Font Server 配置文件, 118
  - xfnts\_monitor\_check 方法, 127
  - xfnts\_monitor\_start 方法, 125
  - xfnts\_monitor\_stop 方法, 126
  - xfnts\_probe 主循环, 128
  - xfnts\_start 方法, 120
  - xfnts\_stop 方法, 124
  - xfnts\_update 方法, 135
  - xfnts\_validate 方法, 133
  - 从 svc\_start() 返回, 122
  - 启动服务, 120
  - 确定故障监视器操作, 132
  - 验证服务, 120
- 组件, 22
- DSDL 样例
  - scds\_initialize() 函数, 119
  - SUNW.xfnts RTR 文件, 119
  - SUNW.xfnts 故障监视器, 127
  - svc\_probe() 函数, 129
  - TCP 端口号, 118
  - X Font Server, 117
  - X Font Server 配置文件, 118
  - xfnts\_monitor\_check 方法, 127
  - xfnts\_monitor\_start 方法, 125
  - xfnts\_monitor\_stop 方法, 126
  - xfnts\_probe 主循环, 128
  - xfnts\_start 方法, 120
  - xfnts\_stop 方法, 124
  - xfnts\_update 方法, 135
  - xfnts\_validate 方法, 133
  - 从 svc\_start() 返回, 122
  - 启动服务, 120
  - 确定故障监视器操作, 132
  - 验证服务, 120

## F

Fini 方法, 使用, 70  
Fini方法, 使用, 40

## G

### GDS

- Agent Builder 输出, 168
- Child\_mon\_level 特性, 163
- Failover\_enabled 特性, 163
- Network\_resources\_used 特性, 162
- Port\_list 特性, 161
- Probe\_command 特性, 162
- Probe\_timeout 特性, 163
- Start\_command 扩展特性, 161
- Start\_timeout 特性, 163
- Stop\_command 特性, 162
- Stop\_signal 特性, 164
- Stop\_timeout 特性, 163
- SUNW.gds 资源类型, 159
- 定义, 37
- 何时使用, 161
- 描述, 159
- 使用 Agent Builder, 160, 164
- 使用 Agent Builder 创建服务, 164
- 使用 Agent Builder 的命令行版本创建服务, 170
- 使用 Sun Cluster 管理命令, 160, 168
- 使用的方法, 160
- 使用原因, 160
- 所需特性, 161

## H

HA 数据服务, 测试, 46  
halockrun, 描述, 41  
hatimerun, 描述, 41

## I

Init 方法, 使用, 70  
Init方法, 使用, 40

## J

Java, 使用 CRNP 的应用程序样例, 188

## L

libdsdev.so, DSDL, 17  
libscha.so, RMAPI, 17

## M

max, 资源类型迁移, 49  
min, 资源类型迁移, 49  
Monitor\_check 方法  
兼容性, 52  
使用, 71  
Monitor\_start 方法, 使用, 71  
Monitor\_stop 方法, 使用, 71

## P

PMF  
函数, DSDL, 176  
用途, 41  
Postnet\_start 方法, 使用, 71  
Postnet\_stop, 兼容性, 52  
Prenet\_start 方法, 使用, 71

## R

Resource\_type, 迁移, 50

### RGM

- 描述, 20
- 目的, 18
- 资源的处理, 18
- 资源类型的处理, 18
- 资源组的处理, 18

### RMAPI, 17

- C 程序函数, 65
- libscha.so, 17
- shell 命令, 64
- 出口代码, 69
- 方法变量, 69
- 公用程序函数, 68
- 回叫方法, 68

## RMAPI (续)

- 群集函数, 67
- 群集命令, 65
- 实现的位置, 17
- 资源函数, 66
- 资源类型函数, 66
- 资源类型命令, 65
- 资源命令, 64
- 资源组函数, 67
- 资源组命令, 65
- 组件, 21

## RT\_Version

- 何时更改, 51
- 何时无需更改, 51
- 目的, 51

*RT\_version*, 迁移, 50

rtconfig 文件, 151

## RTR

- 描述, 21
- 文件
  - SUNW.xfnts, 119
  - 更改, 60
  - 描述, 107
  - 迁移, 50

## S

scds\_initialize() 函数, 119

shell 命令, RMAPI, 64

start 方法, 使用, 38, 69

stop 方法

- 兼容性, 52
- 使用, 38, 70

Sun Cluster

- 命令, 23
- 使用 GDS, 160

SunPlex Agent Builder, 请参见 Agent Builder

SunPlex Manager, 描述, 22

SUNW.xfnts

- RTR 文件, 119
- 故障监视器, 127

svc\_probe() 函数, 129

## T

TCP 连接, 使用 DSDL 故障监视, 175

## U

Update 方法

- 兼容性, 52
- 使用, 42, 71

## V

Validate 方法

- 检查要升级的特性值, 55
- 升级, 53
- 使用, 42, 70

*Vendor\_id*

- 迁移, 50
- 区分, 50

## W

When\_disabled, #*\$upgrade\_from* 指令, 52

When\_offline, #*\$upgrade\_from* 指令, 52

When\_unmanaged, #*\$upgrade\_from* 指令, 52

When\_unmonitored, #*\$upgrade\_from* 指令, 52

## X

X Font Server

- 定义, 117
- 配置文件, 118

xfnts\_monitor\_check, 127

xfnts\_monitor\_start, 125

xfnts\_monitor\_stop, 126

xfnts\_start, 120

xfnts\_stop, 124

xfnts\_update, 135

xfnts\_validate, 133

xfns 服务器, 端口号, 118

## 安

安装 Agent Builder, 138

安装要求, 资源类型软件包, 59

## 编

编程体系结构, 18  
编辑已生成的 Agent Builder 源代码, 146  
编写数据服务, 45

## 变

变量, RMAPI 方法, 69

## 菜

### 菜单

Agent Builder, 153  
Agent Builder “编辑”, 153  
Agent Builder 文件, 153

## 测

### 测试

HA 数据服务, 46  
数据服务, 45

## 持

持续连接机制, 使用, 45

## 出

出口代码, RMAPI, 69

## 创

创建资源类型, Agent Builder, 145

## 存

存取网络地址, 使用 DSDL, 105

## 错

错误状态, CRNP, 184

## 代

### 代码

RMAPI 出口, 69  
更改方法, 60  
更改监视器, 60

## 登

登记 CRNP 客户机和服务器, 181

## 二

二进制文件, Agent Builder, 148

## 方

### 方法

Boot, 40, 113  
Fini, 40, 70, 113  
Init, 40, 70, 113  
Monitor\_check, 71, 112  
Monitor\_check 回叫, 71  
Monitor\_start, 71, 111  
Monitor\_start 回叫, 71  
Monitor\_stop, 71, 111  
Monitor\_stop 回叫, 71  
Postnet\_start, 71  
Postnet\_start 回叫, 71  
Prenet\_start, 71  
Prenet\_start 回叫, 71  
Start, 38, 69, 109  
Stop, 38, 70, 110  
Update, 42, 71, 112  
Update 回叫, 71  
Validate, 42, 70, 108  
Validate 回叫, 70  
xfnts\_monitor\_check, 127  
xfnts\_monitor\_start, 125  
xfnts\_monitor\_stop, 126  
xfnts\_start, 120

## 方法 (续)

- xfnts\_stop, 124
  - xfnts\_update, 135
  - xfnts\_validate, 133
  - 回叫, 42
    - 初始化, 69
    - 控制, 69
  - 幂等性, 37
- 方法变量, RMAPI, 69  
方法代码, 更改, 60

## 服

### 服务器

- CRNP, 181
- X Font Server
  - 定义, 117
  - 配置文件, 118
- xfss
  - 端口号, 118

## 公

### 公用程序函数

- DSDL, 176
- RMAPI, 68

## 故

### 故障监视器

- SUNW.xfnts, 127
  - 函数, DSDL, 176
  - 守护程序
    - 设计, 113
- 故障切换资源, 实现, 42

## 函

### 函数

- DSDL, 173
- DSDL PMF (进程监视工具), 176
- DSDL 公用程序, 176
- DSDL 故障监视器, 176
- DSDL 特性, 174

## 函数 (续)

- DSDL 网络资源存取, 175
- RMAPI C 程序, 65
- RMAPI 公用程序, 68
- RMAPI 群集, 67
- RMAPI 资源, 66
- RMAPI 资源类型, 66
- RMAPI 资源组, 67
- scds\_initialize(), 119
- svc\_probe(), 129
- 通用 DSDL, 173

## 回

### 回叫方法

- Monitor\_check, 71
- Monitor\_start, 71
- Monitor\_stop, 71
- Postnet\_start, 71
- Prenet\_start, 71
- RMAPI, 68
- Update, 71
- Validate, 70
- 初始化, 69
- 概述, 17
- 控制, 69
- 描述, 20
- 使用, 42

## 获

- 获得全限定名称, 50

## 监

- 监视器代码, 更改, 60

## 检

- 检查, 验证可伸缩服务, 45

## 脚

脚本, Agent Builder, 149

## 界

界面, 命令行, 23

## 进

进程管理, 41  
进程管理工具, 概述, 17  
进程监视工具, 请参见PMF  
进程树, 创建资源类型具有多个独立的, 145

## 可

可调性选项, 50  
    Anytime, 52  
    At\_creation, 52  
    When\_disabled, 52  
    When\_offline, 52  
    When\_unmanaged, 52  
    When\_unmonitored, 52  
可调性约束, 文档要求, 56  
可伸缩服务, 验证, 45  
可伸缩资源, 实现, 43

## 克

克隆现有的资源类型, Agent Builder, 145

## 客

客户机, CRNP, 181

## 扩

扩展特性, 声明, 35

## 幂

幂等性, 方法, 37

## 命

命令

halockrun, 41  
hatimerun, 41  
RMAPI 资源类型, 65  
Sun Cluster, 23  
用于创建 GDS, 160, 168

命令行

Agent Builder, 147  
命令, 23

## 目

目录, Agent Builder, 150  
目录结构, Agent Builder, 147

## 配

配置, Agent Builder, 138

## 屏

屏幕

创建, 140  
配置, 142

## 普

普通数据服务  
请参见GDS

## 迁

迁移资源类型, 49

## 区

区分多个登记版本, *RT\_version*, 50  
区分供应商, *Vendor\_id*, 50

## 全

全限定名称, 如何获得, 50

## 缺

缺省特性值

Sun Cluster 3.0, 56  
继承时, 55  
群集配置系统信息库, 55  
升级, 55  
用于升级的新值, 55

## 群

群集函数, *RMAPI*, 67  
群集命令, *RMAPI*, 65  
群集配置系统信息库, 55  
群集重配置通知协议, 请参见 *CRNP*

## 日

日志, 添加到资源, 41

## 软

软件包目录, *Agent Builder*, 150

## 升

升级

缺省特性值, 55  
文档要求, 56  
资源类型实例, 57  
升级支持, 定义的, 49

## 实

实例

使用 *CRNP* 的 Java 应用程序, 188  
数据服务, 73  
资源类型升级, 57

实现

*RMAPI*, 17  
使用 *DSDL* 的故障监视器, 105  
资源类型监视器, 56  
资源类型名称, 56

## 使

使用 *DSDL* 调试资源类型, 106  
使用 *DSDL* 启动数据服务, 104  
使用 *DSDL* 启用 HA 本地文件系统, 106  
使用 *DSDL* 停止数据服务, 104

## 手

手册页, *Agent Builder*, 149

## 守

守护程序, 设计故障监视器, 113

## 属

属性, 资源特性, 219

## 数

数据服务

编写, 45  
测试, 45  
测试 HA, 46  
传送到群集以进行测试, 29  
创建  
    分析适用性, 25  
    确定接口, 27  
    设置开发环境, 28  
    样例, 73

## 数据服务, 样例 (续)

- Monitor\_check 方法, 95
- Monitor\_start 方法, 93
- Monitor\_stop 方法, 94
- RTR 文件, 74
- RTR 文件中的扩展特性, 78
- RTR 文件中的资源特性, 76
- Start 方法, 83
- Stop 方法, 86
- Update 方法, 100
- Validate 方法, 96
- 处理特性更新, 96
- 定义故障监视器, 88
- 获取特性信息, 82
- 控制数据服务, 83
- 生成错误消息, 82
- 探测程序, 88
- 通用功能, 79

数据服务开发库, 请参见DSDL

## 数据服务样例

- Monitor\_check 方法, 95
- Monitor\_start 方法, 93
- Monitor\_stop 方法, 94
- RTR 文件, 74
- RTR 文件中的扩展特性, 78
- RTR 文件中的特性样例, 76
- Start 方法, 83
- Stop 方法, 86
- Update 方法, 100
- Validate 方法, 96
- 处理特性更新, 96
- 定义故障监视器, 88
- 获取特性信息, 82
- 控制数据服务, 83
- 生成错误消息, 82
- 探测程序, 88
- 通用功能, 79

## 特

### 特性

- Child\_mon\_level, 163
- Failover\_enabled, 163
- GDS, 必需的, 163
- Network\_resources\_used, 162
- Port\_list, 161
- Probe\_command, 162

## 特性 (续)

- Probe\_timeout, 163
- Start\_command 扩展, 161
- Start\_timeout, 163
- Stop\_command, 162
- Stop\_signal, 164
- Stop\_timeout, 163
- 更改资源, 42
- 设置资源, 29, 42
- 设置资源类型, 29
- 声明扩展, 35
- 声明资源, 32
- 声明资源类型, 30
- 资源, 210
- 资源类型, 205
- 资源组, 217
- 特性函数, DSDL, 17
- 特性属性, 资源, 219
- 特性值, 缺省, 55

## 网

网络资源存取函数, DSDL, 175

## 文

### 文档要求

- 可调性约束, 56
- 升级, 56

### 文件

- Agent Builder 中的二进制, 148
- Agent Builder 中的源文件, 148
- Agent Builder 中的支持, 150
- rtconfig, 151

## 消

- 消息, SC\_CALLBACK\_REG CRNP, 181
- 消息日志, 添加到资源, 41

## 协

协议, CRNP, 178

## 选

选项, 可调性, 50

## 验

验证检查, 可伸缩服务, 45

## 依

依赖性, 在资源间进行协调, 46

## 源

源代码, 编辑已生成的 Agent Builder, 145

源文件, Agent Builder, 148

## 支

支持文件, Agent Builder, 150

## 值

值, 缺省特性, 55

## 指

指令

#\$upgrade\_from, 51

放置在 RTR 文件中, 51

可调性约束, 51

缺省可调性, 51

## 重

重复使用完成的工作, Agent Builder, 145

## 主

主节点, 19

主控节点, 描述, 19

主要节点, 19

## 资

资源

监视, 40

描述, 19

启动, 38

迁移到其它版本, 53

实现故障切换, 42

实现可伸缩, 43

添加消息日志, 41

停止, 38

协调依赖性, 46

资源的 Type\_version 特性, 52

编辑, 52

可调性, 52

资源管理 API, 请参见RMAPI

资源函数, RMAPI, 66

资源类型

创建, 145

多个版本, 49

描述, 19

迁移要求, 49

升级, 53

使用 DSDL 调试, 106

资源类型登记, 请参见RTR

资源类型函数, RMAPI, 66

资源类型监视器, 实现, 56

资源类型名称

Sun Cluster 3.0, 51

版本后缀, 50

不包含版本后缀, 51

实现, 56

限定, 50

资源类型命令, RMAPI, 65

资源类型软件包, 安装要求, 59

资源类型升级, 实例, 57

资源类型特性, 205

设置, 29

声明, 30

资源命令, RMAPI, 64

资源特性, 210

访问有关信息, 36

更改, 42

设置, 29, 42

- 资源特性 (续)
  - 声明, 32
- 资源特性属性, 219
- 资源依赖性, 协调, 46
- 资源组
  - 故障切换, 20
  - 可伸缩, 20
  - 描述, 19
  - 特性, 20
- 资源组管理器, 请参见RGM
- 资源组函数, RMAPI, 67
- 资源组命令, RMAPI, 65
- 资源组特性, 217
  - 访问有关信息, 36

## 组

- 组件, RMAPI, 21

## 浏

- 浏览 Agent Builder, 151

