

080328@19860

## Sun Cluster データサービス開発ガイド (Solaris OS 版)

インストラクターノート

Sun Microsystems, Inc.

4150 Network Circle

Santa Clara, CA 95054

U.S.A.

Part No: 820-3938-10

2008 年 2 月, Revision A Copyright 2008 Sun Microsystems, Inc.

4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved. >Sun

Microsystems, Inc. (以下 Sun Microsystems 社とします) は、本書に記述されている製品らに限定されるものではありません。

U.S. Government Rights Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

この配布には、第三者によって開発された素材を含んでいることがあります。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システム Company, Ltd. が独占的にライセンスしている米国ならびに他の国における登録商標で Sun、Sun Microsystems、Sun のロゴマーク、Solaris のロゴマーク、Java Coffee Cup Java、および Solaris は、米国およびその他の国における米国 Sun Microsystems, Inc. (社とします) の商標、登録商標もしくは、サービスマークです。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用し、商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくも

OPEN LOOK および Sun Graphical User Interface は、米国 Sun Microsystems 社が自社

Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカル・ユー

社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社

Interface の非独占的ライセンスを取得しており、このライセンスは、OPEN LOOK の

Sun Microsystems 社との書面によるライセンス契約を遵守する、米国 Sun Microsystem

本書で言及されている製品や含まれている情報は、米国輸出規制法で規制されるもので

ます。米国が禁輸の対象としている国や、限定はされませんが、取引禁止顧客や特別指

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物また

続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要で

本製品に含まれる HG-MinchoL、HG-MinchoL-Sun、HG-PMinchoL-Sun、HG-GothicB、

HG-PGothicB-Sun は、株式会社リコーがリョービマジクス株式会社からライセンス付

は、株式会社リコーが財団法人日本規格協会からライセンス供与されたタイプフェース

OPENLOOK、OpenBoot、JLE は、サン・マイクロシステムズ株式会社の登録商標です

Wnn は、京都大学、株式会社アステック、オムロン株式会社で共同開発されたソフト

Wnn6 は、オムロン株式会社、オムロンソフトウェア株式会社で共同開発されたソフト

All Rights Reserved. Copyright © 1995-2008 Sun Microsystems, Inc. All Rights

Reserved.

「ATOK」は、株式会社ジャストシステムの登録商標です。

「ATOK Server/ATOK12」は、株式会社ジャストシステムの著作物であり、「ATOK S

「ATOK Server/ATOK12」に含まれる郵便番号辞書 (7 桁/5 桁) は日本郵政公社が公開し

「ATOK Server/ATOK12」に含まれるフェイスマーク辞書は、株式会社ビレッジセンタ

Unicode は、Unicode, Inc. の商標です。

本書で参照されている製品やサービスに関しては、該当する会社または組織に直接お問

原典: Sun Cluster 3.2 Release Notes for Solaris OS

Part No: 820-3938-10

Revision A

# 目次

---

はじめに .....	13
<b>1 リソース管理の概要 .....</b>	<b>19</b>
Sun Cluster アプリケーション環境 .....	19
RGM モデル .....	21
リソースタイプの説明 .....	21
リソースの説明 .....	22
リソースグループの説明 .....	22
リソースグループマネージャー .....	23
コールバックメソッド .....	24
プログラミングインタフェース .....	25
Resource Management API .....	25
Data Service Development Library .....	25
Sun Cluster Agent Builder .....	26
Resource Group Manager の管理インタフェース .....	26
Sun Cluster Manager .....	26
clsetup ユーティリティ .....	27
管理コマンド .....	27
<b>2 データサービスの開発 .....</b>	<b>29</b>
アプリケーションの適合性の分析 .....	29
使用するインタフェースの決定 .....	31
データサービス作成用開発環境の設定 .....	32
▼ 開発環境の設定方法 .....	33
データサービスをクラスタに転送する方法 .....	34
リソースとリソースタイププロパティの設定 .....	34
リソースタイププロパティの宣言 .....	35

リソースプロパティの宣言 .....	38
拡張プロパティの宣言 .....	42
コールバックメソッドの実装 .....	44
リソースとリソースグループのプロパティ情報へのアクセス .....	44
メソッドの呼び出し回数への非依存性 .....	45
メソッドがゾーンで呼び出される仕組み .....	45
汎用データサービス .....	46
アプリケーションの制御 .....	46
リソースの起動と停止 .....	46
Init、Fini、Boot オプションメソッドの使用 .....	49
リソースの監視 .....	52
大域ゾーン内でのみ実行されるモニターおよびメソッドの実装 .....	53
メッセージログのリソースへの追加 .....	55
プロセス管理の提供 .....	55
リソースへの管理サポートの提供 .....	56
フェイルオーバーリソースの実装 .....	56
スケーラブルリソースの実装 .....	57
スケーラブルサービスの妥当性検査 .....	60
データサービスの作成と検証 .....	60
TCP キープアライブを使用したサーバーの保護 .....	60
HA データサービスの検証 .....	61
リソース間の依存関係の調節 .....	62
<b>3 Resource Management API リファレンス .....</b>	<b>65</b>
RMAPI アクセスメソッド .....	65
RMAPI シェルコマンド .....	65
C 関数 .....	67
RMAPI コールバックメソッド .....	71
コールバックメソッドに提供できる引数 .....	71
コールバックメソッドの終了コード .....	72
制御および初期化コールバックメソッド .....	72
管理サポートメソッド .....	76
ネットワーク関連コールバックメソッド .....	77
モニター制御コールバックメソッド .....	77

<b>4</b>	リソースタイプの変更 .....	79
	リソースタイプの変更の概要 .....	79
	リソースタイプ登録ファイルの内容の設定 .....	80
	リソースタイプ名 .....	80
	#\$upgrade および #\$upgrade_from ディレクティブの指定 .....	81
	RTR ファイルでの RT_version の変更 .....	83
	以前のバージョンの Sun Cluster のリソースタイプ名 .....	83
	クラスタ管理者がアップグレードする際の処理 .....	84
	リソースタイプモニターコードの実装 .....	84
	インストール要件とパッケージの決定 .....	85
	RTR ファイルを変更する前に .....	86
	モニターコードの変更 .....	86
	メソッドコードの変更 .....	86
	使用するパッケージスキーマの決定 .....	87
	変更されたリソースタイプに提供すべき文書 .....	88
	アップグレードのインストール前に実行すべき事柄に関する情報 .....	88
	リソースをアップグレードする時点に関する情報 .....	89
	リソースプロパティに対する変更に関する情報 .....	90
<b>5</b>	サンプルデータサービス .....	91
	サンプルデータサービスの概要 .....	91
	リソースタイプ登録ファイルの定義 .....	92
	RTR ファイルの概要 .....	92
	サンプル RTR ファイルのリソースタイププロパティ .....	93
	サンプル RTR ファイルのリソースプロパティ .....	95
	すべてのメソッドに共通な機能の提供 .....	98
	コマンドインタプリタの指定およびパスのエクスポート .....	99
	PMF_TAG と SYSLOG_TAG 変数の宣言 .....	99
	関数の引数の構文解析 .....	100
	エラーメッセージの生成 .....	102
	プロパティ情報の取得 .....	102
	データサービスの制御 .....	103
	Start メソッドの仕組み .....	103
	Stop メソッドの仕組み .....	107
	障害モニターの定義 .....	109

---

検証プログラムの仕組み .....	110
Monitor_start メソッドの仕組み .....	116
Monitor_stop メソッドの仕組み .....	117
Monitor_check メソッドの仕組み .....	118
プロパティ更新の処理 .....	119
Validate メソッドの仕組み .....	119
Update メソッドの仕組み .....	124
<b>6 Data Service Development Library .....</b>	<b>127</b>
DSDL の概要 .....	127
構成プロパティの管理 .....	128
データサービスの起動と停止 .....	129
障害モニターの実装 .....	129
ネットワークアドレス情報へのアクセス .....	130
実装したリソースタイプのデバッグ .....	130
高可用性ローカルファイルシステムの有効化 .....	131
<b>7 リソースタイプ的设计 .....</b>	<b>133</b>
リソースタイプ登録ファイル .....	134
Validate メソッド .....	134
Start メソッド .....	136
Stop メソッド .....	137
Monitor_start メソッド .....	139
Monitor_stop メソッド .....	139
Monitor_check メソッド .....	140
Update メソッド .....	140
Init、Fini、Boot の各メソッドの説明 .....	141
障害モニターデーモンの設計 .....	142
<b>8 サンプル DSDL リソースタイプの実装 .....</b>	<b>145</b>
X Font Server について .....	145
X Font Server の構成ファイル .....	146
TCP ポート番号 .....	146
SUNW.xfnts の RTR ファイル .....	147

関数とコールバックメソッドの命名規則 .....	147
scds_initialize() 関数 .....	148
xfnts_start メソッド .....	148
X Font Server の起動前のサービスの検証 .....	148
svc_start() によるサービスの起動 .....	149
svc_start() からの復帰 .....	150
xfnts_stop メソッド .....	153
xfnts_monitor_start メソッド .....	154
xfnts_monitor_stop メソッド .....	155
xfnts_monitor_check メソッド .....	156
SUNW.xfnts 障害モニター .....	157
xfnts_probe のメインループ .....	158
svc_probe() 関数 .....	159
障害モニターのアクションの決定 .....	162
xfnts_validate メソッド .....	163
xfnts_update メソッド .....	166
<b>9 Sun Cluster Agent Builder .....</b>	<b>167</b>
Agent Builder の概要 .....	167
Agent Builder の使用にあたって .....	168
Agent Builder の使用 .....	169
アプリケーションの分析 .....	169
Agent Builder のインストールと構成 .....	170
Agent Builder 画面 .....	170
Agent Builder の起動 .....	171
Agent Builder のナビゲーション .....	172
作成画面の使用 .....	175
構成画面の使用 .....	178
Agent Builder の Korn シェルベース \$hostnames 変数の使用 .....	180
プロパティ変数の使用 .....	181
Agent Builder で作成したコードの再利用 .....	183
▼ 既存のリソースタイプからクローンを作成する方法 .....	183
▼ コマンド行バージョンの Agent Builder を使用する方法 .....	185
Agent Builder で作成されるディレクトリ構造 .....	186
Agent Builder の出力 .....	187

ソースファイルとバイナリファイル .....	187
Sun Cluster Agent Builder で作成されるユーティリティスクリプトとマニュアル ページ .....	188
Agent Builder で作成されるサポートファイル .....	190
Agent Builder で作成されるパッケージディレクトリ .....	190
rtconfig ファイル .....	190
Agent Builder の Cluster Agent モジュール .....	191
▼ Cluster Agent モジュールをインストールし設定する方法 .....	191
▼ Cluster Agent モジュールを起動する方法 .....	192
Cluster Agent モジュールの使用 .....	194
Cluster Agent モジュールと Agent Builder の違い .....	196
<b>10 汎用データサービス .....</b>	<b>197</b>
GDS の概念 .....	197
コンパイル済みリソースタイプ .....	198
GDS を使用することの利点と欠点 .....	198
GDS を使用するサービスの作成方法 .....	199
GDS によるイベントのロギング .....	199
必須の GDS プロパティ .....	200
任意の GDS プロパティ .....	201
Agent Builder を使って、GDS を使用するサービスを作成 .....	204
GDS ベースのスクリプトの作成と構成 .....	204
▼ Agent Builder を起動し、スクリプトを作成する .....	204
▼ スクリプトを構成する方法 .....	207
Agent Builder からの出力 .....	209
Sun Cluster 管理コマンドを使って、GDS を使用するサービスを作成 .....	210
▼ Sun Cluster 管理コマンドを使って GDS ベースの高可用性サービスを作成する 方法 .....	210
▼ Sun Cluster 管理コマンドを使って GDS ベースのスケラブルサービスを作成す る方法 .....	211
Agent Builder のコマンド行インタフェース .....	212
▼ コマンド行バージョンの Agent Builder を使って、GDS を使用するサービスを作成 する .....	212
<b>11 DSDL API 関数 .....</b>	<b>215</b>
汎用関数 .....	215

初期化関数 .....	216
取得関数 .....	216
フェイルオーバー関数と再起動関数 .....	216
実行関数 .....	217
プロパティ関数 .....	217
ネットワークリソースアクセス関数 .....	217
ホスト名関数 .....	218
ポートリスト関数 .....	218
ネットワークアドレス関数 .....	218
TCP 接続関数を使用する障害監視 .....	218
PMF 関数 .....	219
障害監視関数 .....	220
ユーティリティ関数 .....	220
<b>12 クラスタ再構成通知プロトコル .....</b>	<b>221</b>
CRNP の概念 .....	221
CRNP の動作 .....	222
CRNP のセマンティクス .....	223
CRNP メッセージのタイプ .....	223
クライアントをサーバーに登録する方法 .....	225
管理者によるサーバー設定の前提 .....	225
サーバーによるクライアントの識別方法 .....	225
クライアントとサーバー間での SC_CALLBACK_REG メッセージの受け渡し方法 ...	225
クライアントに対するサーバーの応答方法 .....	227
SC_REPLY メッセージの内容 .....	228
クライアントによるエラー状況の処理 .....	228
サーバーがクライアントにイベントを配信する方法 .....	229
イベント配信の保証 .....	230
SC_EVENT メッセージの内容 .....	230
CRNP によるクライアントとサーバーの認証 .....	232
CRNP を使用する Java アプリケーションの作成例 .....	233
▼ 環境を設定する .....	233
▼ アプリケーション開発を開始する .....	234
▼ コマンド行引数を解析する .....	236
▼ イベント受信スレッドを定義する .....	236

---

▼ コールバックの登録と登録解除を行う .....	237
▼ XML を生成する .....	238
▼ 登録メッセージと登録解除メッセージを作成する .....	242
▼ XML パーサーの設定方法 .....	244
▼ 登録応答を解析する .....	245
▼ コールバックイベントを解析する .....	247
▼ アプリケーションを実行する .....	251
<b>A</b> 標準プロパティ .....	253
資源タイプのプロパティ .....	253
リソースのプロパティ .....	264
リソースグループのプロパティ .....	285
リソースプロパティの属性 .....	301
<b>B</b> データサービスのコード例 .....	303
リソースタイプ登録ファイルのリスト .....	303
Start メソッドのコードリスト .....	307
Stop メソッドのコードリスト .....	310
gettime ユーティリティのコードリスト .....	312
PROBE プログラムのコードリスト .....	313
Monitor_start メソッドのコードリスト .....	319
Monitor_stop メソッドのコードリスト .....	321
Monitor_check メソッドのコードリスト .....	323
Validate メソッドのコードリスト .....	325
Update メソッドのコードリスト .....	329
<b>C</b> サンプル <b>DSDL</b> リソースタイプのコード例 .....	333
xfnts.c File Listing .....	333
xfnts_monitor_check メソッドのコードリスト .....	347
xfnts_monitor_start メソッドのコードリスト .....	348
xfnts_monitor_stop メソッドのコードリスト .....	349
xfnts_probe メソッドのコードリスト .....	350
xfnts_start メソッドのコードリスト .....	353
xfnts_stop メソッドのコードリスト .....	354

xfnts_update メソッドのコードリスト .....	356
xfnts_validate メソッドのコードリスト .....	357
<b>D</b> 有効な RGM 名と値 .....	359
有効な RGM 名 .....	359
命名規則 (リソースタイプ名を除く) .....	359
リソースタイプ名の形式 .....	360
RGM の値 .....	361
<b>E</b> 非クラスタ対応のアプリケーションの要件 .....	363
多重ホストデータ .....	364
多重ホストデータを配置するためのシンボリックリンクの使用 .....	364
ホスト名 .....	365
多重ホームホスト .....	366
INADDR_ANY へのバインドと特定の IP アドレスへのバインド .....	366
クライアントの再試行 .....	367
<b>F</b> CRNP のドキュメントタイプ定義 .....	369
SC_CALLBACK_REG XML DTD .....	369
NVP AIR XML DTD .....	371
SC_REPLY XML DTD .....	372
SC_EVENT XML DTD .....	373
<b>G</b> CrnpClient.java アプリケーション .....	375
CrnpClient.java のコンテンツ .....	375
索引 .....	401



# はじめに

---

このマニュアルでは、RMAPI (Resource Management (リソース管理) API) を使用して、SPARC® と x86 ベースシステムの両方で Sun™ Cluster データサービスを開発する方法について説明します。

---

注- このリリースでは、SPARC および x86 系列のプロセッサアーキテクチャー (UltraSPARC、SPARC64、および AMD64) を使用するシステムをサポートします。このマニュアルでは、x86 という用語は AMD64 系列のプロセッサアーキテクチャーを使用するシステムを意味します。

---

## 対象読者

このマニュアルは、Sun のソフトウェアとハードウェアについて豊富な知識を持っている経験のある開発者を対象にしています。このマニュアルの情報は、Solaris オペレーティングシステムの知識を前提としています。

## マニュアルの構成

このマニュアルは、次の章と付録で構成されています。

**第 1 章**では、データサービスを開発するのに必要な概念について説明します。

**第 2 章**では、データサービスの開発に関する詳細な情報を説明します。

**第 3 章**では、Resource Management API (RMAPI) を構成するアクセス関数とコールバックメソッドに関する情報を説明します。

**第 4 章**では、リソースタイプを変更するために理解しておく必要がある問題点を説明します。また、クラスタ管理者がリソースを更新できるようにする手段についても説明します。

**第 5 章**では、in.named アプリケーション用の Sun Cluster データサービスの例を示します。

**第 6 章**では、Data Services Development Library (DSDL) を形成するアプリケーションプログラミングインタフェースの概要を説明します。

第7章では、リソースタイプの設計と実装における DSDL の代表的な使用例について説明します。

第8章では、DSDL により実装されるリソースタイプの例を説明します。

第9章では、Sun Cluster Agent Builder について説明します。

第10章では、一般的なデータサービスの作成方法について説明します。

第11章では、DSDL API 関数について説明します。

第12章では、Cluster Reconfiguration Notification Protocol (CRNP) について説明します。CRNP を使用することで、フェイルオーバー用のアプリケーションや拡張性のあるアプリケーションを「クラスタ対応」として設定できます。?

付録Aでは、標準リソースタイプ、リソース、およびリソースグループのプロパティについて説明します。

付録Bでは、データサービスの例について、それぞれのメソッドの完全なコードを示します。

付録Cでは、SUNW.xfnts リソースタイプにおける各メソッドの完全なコードを示します。

付録Dでは、Resource Group Manager (RGM) の名前と値についての文字の要件を説明します。

付録Eでは、クラスタに対応していない、通常のアプリケーションを高可用性に適応させる要件を説明します。

付録Fでは、CRNP のドキュメントタイプ定義を説明します。

付録Gでは、第12章で説明されている CrnpClient.java の完全なアプリケーションを示します。

## 関連マニュアル

関連する Sun Cluster トピックについての情報は、以下の表に示すマニュアルを参照してください。Sun Cluster マニュアルは、すべて <http://docs.sun.com> で参照できます。

トピック	マニュアル
概要	『Sun Cluster の概要 (Solaris OS 版)』

トピック	マニュアル
Concept	『Sun Cluster の概念 (Solaris OS 版)』
ハードウェアの設計と管理	『Sun Cluster 3.1 - 3.2 Hardware Administration Manual for Solaris OS』 各ハードウェア管理ガイド
ソフトウェアのインストール	『Sun Cluster ソフトウェアのインストール (Solaris OS 版)』
データサービスのインストールと管理	『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』 各データサービスガイド
データサービスの開発	『Sun Cluster データサービス開発ガイド (Solaris OS 版)』
システム管理	『Sun Cluster のシステム管理 (Solaris OS 版)』
エラーメッセージ	『Sun Cluster Error Messages Guide for Solaris OS』
コマンドと関数のリファレンス	『Sun Cluster Reference Manual for Solaris OS』

Sun Cluster のマニュアルの完全なリストについては、お使いの Sun Cluster ソフトウェアのリリースノートを <http://docs.sun.com> で参照してください。

## 問い合わせについて

Sun Cluster ソフトウェアのインストールや使用に関して問題がある場合は、以下の情報をご用意の上、担当のサービスプロバイダにお問い合わせください。

- 名前と電子メールアドレス
- 会社名、住所、および電話番号
- システムのモデル番号とシリアル番号
- オペレーティングシステムのバージョン番号 (例: Solaris 10 OS)
- Sun Cluster ソフトウェアのバージョン番号 (例: 3.2)

次のコマンドを使用し、システムに関して、サービスプロバイダに必要な情報を収集してください。

コマンド	機能
<code>prtconf -v</code>	システムメモリのサイズと周辺デバイス情報を表示します
<code>psrinfo -v</code>	プロセッサの情報を表示する
<code>showrev -p</code>	インストールされているパッチを報告する

コマンド	機能
SPARC:prtdiag -v	システム診断情報を表示する
/usr/cluster/bin/clnode show-rev	Sun Cluster のリリースとパッケージバージョン情報を表示する

上記の情報にあわせて、`/var/adm/messages` ファイルの内容もご購入先にお知らせください。

## マニュアル、サポート、およびトレーニング

Sun の Web サイトでは、次のサービスに関する情報も提供しています。

- マニュアル (<http://jp.sun.com/documentation/>)
- サポート (<http://jp.sun.com/support/>)
- トレーニング (<http://jp.sun.com/training/>)

## 表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P-1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
<b>AaBbCc123</b>	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% <b>su</b> password:
<i>AaBbCc123</i>	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm <i>filename</i> と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。

表 P-1 表記上の規則 (続き)

字体または記号	意味	例
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第5章「衝突の回避」を参照してください。  この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% <b>grep</b> '^#define \  XV_VERSION_STRING'

コード例は次のように表示されます。

- C シェル

```
machine_name% command y|n [filename]
```

- C シェルのスーパーユーザー

```
machine_name# command y|n [filename]
```

- Bourne シェルおよび Korn シェル

```
$ command y|n [filename]
```

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。



# リソース管理の概要

---

このマニュアルでは、Oracle®、Sun Java™ System Web Server (以前の Sun ONE Web Server)、DNSなどのソフトウェアアプリケーション用のリソースタイプを作成するためのガイドラインを示します。したがって、このマニュアルはリソースタイプの開発者を対象としています。

このマニュアルの内容を理解するため、『Sun Cluster の概念 (Solaris OS 版)』で説明している概念について十分理解しておいてください。

この章では、データサービスを開発するために理解しておく必要がある概念について説明します。この章の内容は次のとおりです。

- 19 ページの「Sun Cluster アプリケーション環境」
- 21 ページの「RGM モデル」
- 23 ページの「リソースグループマネージャー」
- 24 ページの「コールバックメソッド」
- 25 ページの「プログラミングインタフェース」
- 26 ページの「Resource Group Manager の管理インタフェース」

---

注- このマニュアルでは、「リソースタイプ」と「データサービス」という用語を同じ意味で使用しています。また、このマニュアルではほとんど使用されることはありませんが、「エージェント」という用語も「リソースタイプ」や「データサービス」と同じ意味で使用されます。

---

## Sun Cluster アプリケーション環境

Sun Cluster システムを使用すると、アプリケーションを高度な可用性とスケーラビリティを備えたリソースとして実行および管理できます。RGM (Resource Group Manager) は、高可用性とスケーラビリティを実現するための機構を提供します。

この機能を利用するためのプログラミングインタフェースを形成する要素は、次のとおりです。

- ユーザーが作成するコールバックメソッドのセット。このコールバックメソッドにより、RGMはクラスタ内のアプリケーションを制御することができます。
- Resource Management API (RMAPI)。コールバックメソッドの作成に使用する低レベルのAPI コマンドおよびAPI 関数のセットです。RMAPIは `libscha.so` ライブラリとして実装されています。
- Process Monitor Facility (PMF)。クラスタ内のプロセスを監視し、再起動します。
- データサービス開発ライブラリ(Data Service Development Library: DSDL)。低レベルAPI およびプロセス管理機能をより高レベルでカプセル化するライブラリ関数のセットです。DSDLは、コールバックメソッドの作成を支援するいくつかの機能を追加します。DSDL 関数は `libdsdev.so` ライブラリとして実装されています。

次の図は、これらの要素の相互関係を示しています。

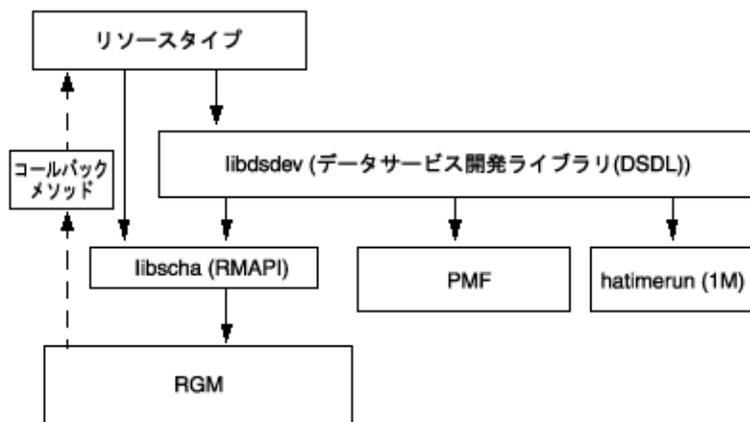


図 1-1 Sun Cluster アプリケーション環境のプログラミングアーキテクチャー

Sun Cluster Agent Builder (第9章を参照) は Sun Cluster パッケージ内のツールで、データサービスの作成プロセスを自動化します。Agent Builder は、(DSDL 関数を使用してコールバックメソッドを作成することにより)C、または(低レベルAPI コマンドを使用してコールバックメソッドを作成することにより) Korn (ksh) シェルコマンドでデータサービスコードを生成します。

RGMは各クラスタノード上でデーモンとして動作し、事前構成したポリシーに従って、選択したノードまたはゾーン上のリソースを自動的に起動および停止します。RGMは、ノードまたはゾーンの障害やリポートが発生した場合もリソースの高可用性を保ちます。これを実現するため、RGMは影響を受けたノードまたはゾーン上でリソースを停止し、別のノードまたはゾーン上でそのリソースを起動します。

また RGM は、リソース固有のモニターを自動的に起動および停止します。これらのモニターは、リソース障害を検出したり、障害が発生しているリソースを別のノードまたはゾーンに再配置したり、さまざまな視点からリソース性能を監視します。

RGM はフェイルオーバーリソースとスケーラブルリソースの両方をサポートしています。フェイルオーバーリソースは、常に単一のノードまたはゾーン上でしかオンラインにできません。スケーラブルリソースは、同時に複数のノードまたはゾーン上でオンラインにできます。ただし、共有アドレスを使用してノード間でサービスの負荷を分散しているスケーラブルリソースは、物理ノードあたり 1 つのゾーンでしかオンラインにできません。

## RGM モデル

ここでは、基本的な用語をいくつか紹介し、RGM とそれに関連するインタフェースについて詳細に説明します。

RGM は、「リソースタイプ」、「リソース」、「リソースグループ」という 3 種類の相互に関連するオブジェクトを処理します。これらのオブジェクトを紹介するために、次のような例を使用します。

開発者は、既存の Oracle DBMS アプリケーションを高可用性にするリソースタイプ `ha-oracle` を実装します。エンドユーザーは、マーケティング、エンジニアリング、および財務ごとに異なるデータベースを定義し、それぞれのリソースタイプを `ha-oracle` にします。クラスタ管理者は、上記リソースを異なるリソースグループに配置することによって、異なるノードまたはゾーン上で実行したり、個別にフェイルオーバーできるようにします。開発者は、もう 1 つのリソースタイプ `ha-calendar` を作成し、Oracle データベースを必要とする高可用性のカレンダーサーバーを実装します。クラスタ管理者は、財務カレンダー用のリソースを財務データベースリソースと同じリソースグループに配置します。そうすることで、両方のリソースが必ず同じノード上またはゾーン内で動作し、一緒にフェイルオーバーするようになります。

## リソースタイプの説明

リソースタイプは次のものから構成されます。

- クラスタ上で実行されるソフトウェアアプリケーション
- アプリケーションをクラスタリソースとして管理するために RGM がコールバックメソッドとして使用する制御プログラム
- クラスタの静的な構成の一部を形成するプロパティセット

RGM は、リソースタイププロパティを使って特定のタイプのリソースを管理します。

---

注-リソースタイプは、ソフトウェアアプリケーションだけでなく、ネットワークアドレスなど、そのほかのシステムリソースも表します。

---

開発者は、リソースタイプのプロパティを指定し、プロパティの値をリソースタイプ登録 (RTR) ファイルに設定します。RTR ファイルは、34 ページの「リソースとリソースタイププロパティの設定」、および `rt_reg(4)` のマニュアルページで説明されている形式に従います。RTR ファイルの例については、92 ページの「リソースタイプ登録ファイルの定義」も参照してください。

253 ページの「資源タイプのプロパティ」に、リソースタイププロパティのリストを示します。

クラスタ管理者は、リソースタイプの実装と実際のアプリケーションをクラスタにインストールし、登録します。登録の手続きによって、RTR ファイルの情報がクラスタ構成に入力されます。データサービスの登録手順については、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』を参照してください。

## リソースの説明

リソースは、そのリソースタイプからプロパティと値を継承します。さらに、開発者は、RTR ファイルでリソースプロパティを宣言できます。264 ページの「リソースのプロパティ」にはリソースプロパティのリストがあります。

クラスタ管理者は、RTR ファイルにプロパティがどのように指定されているかに応じて、特定のプロパティの値を変更できます。たとえば、プロパティ定義は値の許容範囲を指定できます。プロパティ定義には、いつプロパティを調整できるかを指定することもできます。つまり、「調整不可」、「常時調整可」、「作成されたとき (リソースがクラスタに追加されたとき)」または「リソースが無効にされたとき」を指定できます。このような仕様の範囲内で、クラスタ管理者は管理コマンドを使用することでプロパティを変更できます。

クラスタ管理者は、同じタイプのリソースを多数作成して、各リソースに独自の名前とプロパティ値のセットを持たせることができます。これによって、使用しているアプリケーションの複数のインスタンスをクラスタ上で実行できます。このとき、各インスタンスにはクラスタ内で一意の名前が必要です。

## リソースグループの説明

各リソースはリソースグループに構成する必要があります。RGM は、同じグループのすべてのリソースを同じノード上または同じゾーン内で一緒にオンラインかオフラインにします。RGM は、リソースグループをオンラインまたはオフラインにするときに、グループ内の個々のリソースに対してコールバックメソッドを実行しません。

リソースグループが現在オンラインになっているノードまたはゾーンのことを、「主ノード」または「主ゾーン」と呼びます。リソースグループは、その主ノードによってマスター (制御) されます。各リソースグループは、関連付けられた `Nodelist` プロパティを持っており、これによってリソースグループの「潜在的な主ノード」または「マスター」を識別します。クラスタ管理者は `Nodelist` プロパティを設定します。

リソースグループはプロパティセットも持っています。このようなプロパティには、クラスタ管理者が設定できる構成プロパティや、RGM が設定してリソースグループのアクティブな状態を反映する動的プロパティがあります。

RGM は、2 種類のリソースグループ、フェイルオーバーとスケラブルを定義します。フェイルオーバーリソースグループは、いつも 1 つのノードまたはゾーン上だけでオンラインにできます。スケラブルリソースグループは、同時に複数のノードまたはゾーン上でオンラインにできます。RGM は、各種類のリソースグループを作成するためのプロパティセットを提供します。このようなプロパティの詳細については、[34 ページの「データサービスをクラスタに転送する方法」](#)と[44 ページの「コールバックメソッドの実装」](#)を参照してください。

[285 ページの「リソースグループのプロパティ」](#)には、リソースグループプロパティのリストがあります。

## リソースグループマネージャー

Resource Group Manager (RGM) は `rgmd` デーモンとして実装され、大域ゾーン内のクラスタの各メンバー (ノード) 上で動作します。`rgmd` プロセスはすべて互いに通信し、単一のクラスタ規模の機能として動作します。

RGM は、次の機能をサポートします。

- ノードまたはゾーンがブートまたはクラッシュした場合、RGM は必ず、管理されているすべてのリソースグループの可用性を維持しようとします。そのために、RGM は正しいマスター上でそれらのリソースグループを自動的にオンラインにします。
- 特定のリソースが異常終了した場合、そのモニタープログラムはリソースグループを同じマスター上で再起動するか、新しいマスターに切り替えるかを要求できます。
- クラスタ管理者は管理コマンドを実行して、次のいずれかのアクションを要求できます。
  - リソースグループをマスターする権利の変更
  - リソースグループ内の特定のリソースの有効化または無効化
  - リソースタイプ、リソース、リソースグループの作成、削除、変更

RGMは、構成を変更するとき、そのアクションをクラスタのすべてのメンバー(ノード)間で調整します。このような動作を「再構成」と呼びます。RGMは、個々のリソースの状態を変更するため、そのリソース上でリソースタイプに固有のコールバックメソッドを実行します。

## コールバックメソッド

Sun Cluster フレームワークは、コールバックの機構を使用して、データサービスとRGM間の通信を実現します。Sun Cluster フレームワークは、引数と戻り値を含むコールバックメソッドのセットと、RGMが各メソッドを呼び出す環境を定義します。

データサービスを作成するには、個々のコールバックメソッドのセットをコーディングし、個々のメソッドをRGMから呼び出し可能な制御プログラムとして実装します。つまり、データサービスは、単一の実行可能コードではなく、多数の実行可能なスクリプト(ksh)またはバイナリ(C言語)から構成されており、それぞれをRGMから直接呼び出すことができます。

コールバックメソッドをRGMに登録するには、RTRファイルを使用します。RTRファイルでは、データサービスに対して実装した各メソッドのプログラムを指定します。クラスタ管理者がデータサービスをクラスタに登録すると、RGMはRTRファイルを読み取ります。RTRファイルには、コールバックプログラムの識別情報などの情報があります。

リソースタイプの必須コールバックメソッドは、起動メソッド(StartまたはPrenet\_start)と停止メソッド(StopまたはPostnet\_stop)だけです。

コールバックメソッドは、次のようなカテゴリに分類できます。

- 制御および初期化メソッド
  - Start および Stop メソッドは、オンラインまたはオフラインにするグループ内のリソースを起動または停止します。
  - Init、Fini、Boot メソッドは、リソース上で初期化と終了コードを実行しません。
- 管理サポートメソッド
  - Validate メソッドは、管理アクションによって設定されるプロパティーを確認します。
  - Update メソッドは、オンラインリソースのプロパティー設定を更新します。
- ネットワーク関連メソッド
  - Prenet\_start と Postnet\_stop は、同じリソースグループ内のネットワークアドレスが「起動」に構成される前、または「停止」に構成されたあとに、特別な起動処理または停止処理を実行します。
- モニター制御メソッド

- `Monitor_start` と `Monitor_stop` は、リソースのモニターを起動または停止します。
- `Monitor_check` は、リソースグループがノードまたはゾーンに移動される前に、ノードまたはゾーンの信頼性を査定します。

コールバックメソッドの詳細については、[第3章](#)、および `rt_callbacks(1HA)` のマニュアルページを参照してください。また、サンプルデータサービスでのコールバックメソッドについては、[第5章](#)と[第8章](#)を参照してください。

## プログラミングインタフェース

リソース管理アーキテクチャは、データサービス用のコードを作成するため、低レベルまたはベース API、ベース API 上に構築されるより高いレベルのライブラリを提供します。さらに、ユーザーが指定する基本的な情報からデータサービスを自動的に生成するツール、Sun Cluster Agent Builder を提供します。

### Resource Management API

RMAPI (Resource Management API) は、低レベルの関数のセットを提供します。これらの関数により、データサービスは、システム内のリソースタイプ、リソース、リソースグループに関する情報にアクセスしたり、ローカルの再起動やフェイルオーバーを要求したり、リソースの状態を設定できるようになります。これらの関数にアクセスするには、`libscha.so` ライブラリを使用します。RMAPI は、これらのコールバックメソッドを、シェルコマンドまたは C 関数の形で提供できます。RMAPI 関数の詳細については、`scha_calls(3HA)` のマニュアルページと、[第3章](#)を参照してください。また、サンプルデータサービスのコールバックメソッドにおけるこれらの関数の使用法の例については、[第5章](#)を参照してください。

### Data Service Development Library

データサービス開発ライブラリ (Data Service Development Library: DSDL) は、RMAPI 上に構築されており、基盤となる RGM の「メソッドコールバックモデル」を維持しながら、上位レベルの統合フレームワークを提供します。`libdsdev.so` ライブラリには、DSDL 関数が含まれています。

DSDL は、次のようなさまざまなデータサービス開発向けの機能を提供します。

- `libscha.so`。低レベルのリソース管理 API。
- **PMF**。プロセスとその子孫を監視し、これらが終了したときに再起動する手段を提供する、プロセス監視機能 (Process Monitor Facility: PMF)。詳細は、`pmfadm(1M)` と `rpc.pmf(1M)` のマニュアルページを参照してください。

- `hatimerun`。タイムアウトを適用してプログラムを実行するための機能。  
`hatimerun(1M)` のマニュアルページを参照してください。

DSDL は、大多数のアプリケーションに対して、データサービスの構築に必要なほとんどまたはすべての機能を提供します。DSDL は、低レベルの API の代わりになるものではなく、低レベルの API をカプセル化および拡張するためのものであることに注意してください。事実、多くの DSDL 関数は `libscha.so` 関数を呼び出します。これと同じように、開発者は、DSDL を使用しながら `libscha.so` 関数を直接呼び出すことによって、データサービスの大半を作成することができます。

DSDL の詳細については、[第 6 章](#)、および `scha_calls(3HA)` のマニュアルページを参照してください。

## Sun Cluster Agent Builder

Agent Builder は、データサービスの作成を自動化するツールです。このツールでは、ターゲットアプリケーションと作成するデータサービスについての基本的な情報を入力します。Agent Builder は、ソースコードと実行可能コード (C 言語または Korn シェル)、カスタマイズされた RTR ファイル、および Solaris パッケージを含む、データサービスを生成します。

大多数のアプリケーションでは、Agent Builder を使用することにより、わずかなコードを手作業で変更するだけで完全なデータサービスを生成できます。追加プロパティの妥当性検査を必要とするような、より要件の厳しいアプリケーションには、Agent Builder では対応できないこともあります。しかし、このような場合でも、Agent Builder によりコードの大部分を生成できるので、手作業によるコーディングは残りの部分だけで済みます。少なくとも Agent Builder を使用することにより、独自の Solaris パッケージを生成することができます。

## Resource Group Manager の管理インタフェース

Sun Cluster はクラスタを管理するために、グラフィカルユーザーインタフェース (GUI) とコマンドセットの両方を提供します。

### Sun Cluster Manager

Sun Cluster Manager (旧名 SunPlex™ Manager) は、以下の作業を実行できる Web ベースのツールです。

- クラスタのインストール
- クラスタの管理
- リソースやリソースグループの作成と構成

- Sun Cluster ソフトウェアを使ったデータサービスの構成

Sun Cluster Manager のインストール方法、および Sun Cluster Manager によるクラスタソフトウェアのインストール方法については、『Sun Cluster ソフトウェアのインストール (Solaris OS 版)』を参照してください。管理作業については、Sun Cluster Manager のオンラインヘルプを参照してください。

## clsetup ユーティリティー

clsetup(1CL) ユーティリティーを使用すると、Sun Cluster の管理作業の大部分を対話形式で行うことができます。

clsetup ユーティリティーを使用すると、Sun Cluster の次の要素を管理できます。

- 定足数 (quorum)
- リソースグループ
- データサービス
- クラスタインターコネクト
- デバイスグループとボリューム
- プライベートホスト名
- 新規ノード
- そのほかのクラスタタスク

また、clsetup ユーティリティーを使用すると、次の操作を実行できます。

- リソースグループを作成
- ネットワークリソースをリソースグループに追加
- データサービスリソースをリソースグループに追加
- リソースタイプを登録する
- リソースグループをオンラインまたはオフラインにする
- リソースグループのスイッチオーバー
- リソースグループの自動復旧アクションの中断または再開
- リソースの有効化または無効化
- リソースグループプロパティの変更
- リソースプロパティを変更する
- リソースグループからリソースを削除
- リソースグループを削除
- リソースからの Stop\_failed エラーフラグのクリア

## 管理コマンド

RGM オブジェクトを管理するための Sun Cluster コマンドは、clresourcetype、clresourcegroup、clresource、clnode、および cluster です。

`clresourcetype`、`clresourcegroup`、および `clresource` コマンドを使用すると、RGM で使用されているリソースタイプ、リソースグループ、およびリソースオブジェクトを表示、作成、構成、および削除できます。これらのコマンドはクラスタの管理インタフェースの一部であり、この章の残りで説明しているアプリケーションインタフェースと同じプログラミングコンテキストでは使用されません。しかし、`clresourcetype`、`clresourcegroup`、および `clresource` コマンドは、API が動作するクラスタ構成を構築するためのツールです。管理インタフェースを理解すると、アプリケーションインタフェースも理解しやすくなります。これらのコマンドで実行できる管理作業の詳細については、`clresourcetype(1CL)`、`clresourcegroup(1CL)`、`clresource(1CL)` の各マニュアルページを参照してください。

## データサービスの開発

---

この章では、アプリケーションの可用性とスケーラビリティを高める方法を解説し、データサービスの開発に関する詳細な情報について説明します。

この章の内容は次のとおりです。

- 29 ページの「アプリケーションの適合性の分析」
- 31 ページの「使用するインタフェースの決定」
- 32 ページの「データサービス作成用開発環境の設定」
- 34 ページの「リソースとリソースタイププロパティの設定」
- 44 ページの「コールバックメソッドの実装」
- 46 ページの「汎用データサービス」
- 46 ページの「アプリケーションの制御」
- 52 ページの「リソースの監視」
- 55 ページの「メッセージログのリソースへの追加」
- 55 ページの「プロセス管理の提供」
- 56 ページの「リソースへの管理サポートの提供」
- 56 ページの「フェイルオーバーリソースの実装」
- 57 ページの「スケーラブルリソースの実装」
- 60 ページの「データサービスの作成と検証」

### アプリケーションの適合性の分析

データサービスを作成するための最初の手順では、ターゲットアプリケーションが高可用性またはスケーラビリティを備えるための要件を満たしているかどうかを判定します。アプリケーションが一部の要件を満たしていない場合は、アプリケーションの可用性とスケーラビリティを高めるようにアプリケーションのソースコードを変更します。

次に、アプリケーションが高可用性またはスケーラビリティを備えるための要件を要約します。詳細情報を確認したい場合や、アプリケーションのソースコードを変更する必要がある場合は、[付録 B](#) を参照してください。

注- スケーラブルサービスは、次に示す高可用性の要件をすべて満たした上で、いくつかの追加要件も満たしている必要があります。

- Sun Cluster 環境では、ネットワーク対応(クライアントサーバーモデル)とネットワーク非対応(クライアントレス)のアプリケーションはどちらも、高可用性またはスケーラビリティを備えることが可能です。ただし、タイムシェアリング環境では、アプリケーションはサーバー上で動作し、telnet または rlogin 経由でアクセスされるため、Sun Cluster は可用性を強化することはできません。
- アプリケーションはクラッシュに対する耐障害性(クラッシュトレラント)を備えていなければなりません。つまりアプリケーションは、ノードの予期しない動作停止やゾーンの障害が発生したあとに起動したときに、必要に応じてディスクデータを復元する必要があります。さらに、クラッシュ後の復元時間にも制限が課せられます。ディスクを復元し、アプリケーションを再起動できる能力は、データの整合性に関わる問題であるため、クラッシュトレラントであることは、アプリケーションが高可用性を備えるための前提条件となります。データサービスは接続を復元できる必要はありません。
- アプリケーションは、自身が動作するノードの物理ホスト名に依存してはいけません。詳細は、365 ページの「ホスト名」を参照してください。
- アプリケーションは、複数の IP アドレスが「起動」状態になるよう構成されている環境で正しく動作する必要があります。たとえば、ノードが複数のパブリックネットワーク上に存在する多重ホームホスト環境や、単一のハードウェアインタフェース上に複数の論理インタフェースが「起動」状態になるよう構成されているノードが存在する環境があります。
- 高可用性を備えるには、アプリケーションデータは高可用性のローカルファイルシステムに格納されている必要があります。364 ページの「多重ホストデータ」を参照してください。

アプリケーションがデータの位置に固定されたパス名を使用している場合、アプリケーションのソースコードを変更しなくても、クラスタファイルシステム内の場所を指すシンボリックリンクにそのパスを変更できる場合があります。詳細は、364 ページの「多重ホストデータを配置するためのシンボリックリンクの使用」を参照してください。
- アプリケーションのバイナリとライブラリは、ローカルの各ノードまたはゾーン上、あるいはクラスタファイルシステムに格納できます。クラスタファイルシステム上に格納する利点は、1 箇所にインストールするだけで済む点です。
- 初回の照会がタイムアウトした場合、クライアントは自動的に照会を再試行する必要があります。アプリケーションとプロトコルがすでに単一サーバーのクラッシュと再起動に対応できている場合、関連するリソースグループのフェイルオーバーまたはスイッチオーバーにも対応します。詳細は、367 ページの「クライアントの再試行」を参照してください。
- アプリケーションは、クラスタファイルシステム内で UNIX® ドメインソケットまたは名前付きパイプを使用してはいけません。

さらに、スケーラブルサービスは、次の要件も満たしている必要があります。

- アプリケーションは、複数のインスタンスを実行でき、すべてのインスタンスがクラスタファイルシステム内の同じアプリケーションデータを処理できる必要があります。
- アプリケーションは、複数のノードまたはゾーンからの同時アクセスに対してデータの整合性を保証する必要があります。
- アプリケーションは、クラスタファイルシステムのように、グローバルに使用可能な機構を備えたロック機能を実装している必要があります。

スケーラブルサービスの場合、アプリケーションの特性により負荷均衡ポリシーが決定されます。たとえば、負荷均衡ポリシー `Lb_weighted` は、任意のインスタンスがクライアントの要求に応答できるポリシーですが、クライアント接続にサーバー上のメモリー内キャッシュを使用するアプリケーションには適用されません。この場合、特定のクライアントのトラフィックをアプリケーションの1つのインスタンスに制限する負荷均衡ポリシーを指定します。負荷均衡ポリシー `Lb_sticky` と `Lb_sticky_wild` は、クライアントからのすべての要求を同じアプリケーションインスタンスに繰り返して送信します。この場合、そのアプリケーションはメモリー内キャッシュを使用できます。異なるクライアントから複数のクライアント要求が送信された場合、RGM はサービスの複数のインスタンスに要求を分配します。スケーラブルデータサービスに対応した負荷均衡ポリシーを設定する方法については、[56 ページの「フェイルオーバーリソースの実装」](#)を参照してください。

## 使用するインタフェースの決定

Sun Cluster 開発者サポートパッケージ (SUNWscdev) は、データサービスメソッドのコーディング用に2種類のインタフェースセットを提供します。

- Resource Management API (RMAPI) - (`libscha.so` ライブラリの) 低レベルの関数セット
- データサービス開発ライブラリ (Data Service Development Library、DSDL) - RMAPI の機能をカプセル化および拡張する、より高いレベルの関数セット (`libdsdev.so` ライブラリとして実装されている)

Sun Cluster 開発者サポートパッケージには、データサービスの作成を自動化するツールである Sun Cluster Agent Builder も含まれています。

次に、データサービスを開発する際の推奨手順を示します。

1. C 言語または Korn シェルのどちらでコーディングするかを決定します。DSDL は C 言語用のインタフェースしか提供しないため、Korn シェルでコーディングする場合は使用できません。
2. Agent Builder を使用すると、必要な情報を指定するだけで、データサービスを生成できます。これには、ソースコードと実行可能コード、RTR ファイル、およびパッケージが含まれます。
3. 生成されたデータサービスをカスタマイズする必要がある場合は、生成されたソースファイルに DSDL コードを追加できます。Agent Builder は、ソースファイル内において独自のコードを追加できる場所にコメント文を埋め込みます。
4. ターゲットアプリケーションをサポートするために、さらにコードをカスタマイズする必要がある場合は、既存のソースコードに RMAPI 関数を追加できます。

実際には、データサービスを作成する方法はいくつもあります。たとえば、Agent Builder が生成したコード内の特定の場所に独自のコードを追加するのではなく、生成されたメソッド全体を書き換えたり、生成された監視プログラムを DSDL または RMAPI 関数を使って最初から作成し直したりできます。

しかし、使用方法に関わらず、ほとんどの場合は Agent Builder を使って開発作業を開始することをお勧めします。次に、その理由を示します。

- Agent Builder が生成するコードは本質的に汎用であり、数多くのデータサービスでテストされています。
- Agent Builder は、RTR ファイル、makefile、リソースのパッケージなど、データサービス用のサポートファイルを作成します。データサービスのコードをまったく使用しない場合でも、このようなファイルを使用することによってかなりの作業を省略できます。
- 生成されたコードは変更できます。

---

注 - RMAPI は C 言語用の関数セットとスクリプト用のコマンドセットを提供しますが、DSDL は C 言語用の関数インタフェースしか提供しません。DSDL は ksh コマンドを提供しないので、Agent Builder で Korn shell (ksh) 出力を指定した場合、生成されるソースコードは RMAPI を呼び出します。

---

## データサービス作成用開発環境の設定

データサービスの開発を始める前に、Sun Cluster 開発パッケージ (SUNWscdev) をインストールして、Sun Cluster のヘッダーファイルやライブラリファイルにアクセスできるようにする必要があります。このパッケージがすでにすべてのクラスタノード上にインストールされている場合でも、通常はクラスタノード上ではなく、クラス

タ外の別の開発マシンでデータサービスを開発します。このような場合、`pkgadd` コマンドを使って、開発マシンに `SUNWscdev` パッケージをインストールする必要があります。

---

注 - 開発マシンでは、必ず Solaris 9 OS または Solaris 10 OS の Developer Distribution または Entire Distribution ソフトウェアグループを使用してください。

---

コードをコンパイルおよびリンクするとき、ヘッダーファイルとライブラリファイルを識別するオプションを設定する必要があります。

---

注 - 互換モードでコンパイルされた C++ コードと標準モードでコンパイルされた C++ コードを Solaris オペレーティングシステム製品や Sun Cluster 製品で併用することはできません。

したがって、Sun Cluster で使用する C++ ベースのデータサービスを作成する場合は、そのデータサービスを次のようにコンパイルする必要があります。

- Sun Cluster 3.0 以前のバージョンで使用する場合は、互換モードでコンパイルする必要があります。
  - Sun Cluster 3.1 以降のバージョンで使用する場合は、標準モードでコンパイルする必要があります。
- 

クラスタノード以外で開発が終了すると、完成したデータサービスをクラスタに転送して、検証を行うことができます。

この節の手順では、次の作業を完了する方法を説明します。

- Sun Cluster 開発パッケージ (`SUNWscdev`) をインストールして、適切なコンパイラオプションとリンカーオプションを設定します。
- データサービスをクラスタに転送します。

## ▼ 開発環境の設定方法

`SUNWscdev` パッケージをインストールして、コンパイラオプションとリンカーオプションをデータサービス開発用に設定する方法について説明します。

- 1 スーパーユーザーになるか、RBAC 承認 `solaris.cluster.modify` を提供する役割になります。
- 2 使用する CD-ROM ディレクトリにディレクトリを変更します。

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

- 3 SUNWscdev パッケージを現在のディレクトリにインストールします。

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

- 4 makefile に、データサービスのコードが使用する **include** ファイルとライブラリファイルを示すコンパイラオプションとリンカーオプションを指定します。

-I オプションは Sun Cluster のヘッダーファイルを指定し、-L オプションは、開発システム上にあるコンパイル時ライブラリの検索パスを指定し、-R オプションはクラスタの実行時リンカーのライブラリの検索パスを指定します。

```
# サンプルデータサービスの Makefile
```

```
...
```

```
-I /usr/cluster/include
```

```
-L /usr/cluster/lib
```

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

```
...
```

## データサービスをクラスタに転送する方法

開発マシン上でデータサービスが完成した場合、データサービスをクラスタに転送して検証する必要があります。転送中のエラーの可能性を減らすため、データサービスのコードと RTR ファイルをパッケージに結合します。そして、サービスを実行するノードにそのパッケージをインストールします。

---

注 - Agent Builder は、このパッケージを自動的に作成します。

---

## リソースとリソースタイププロパティの設定

Sun Cluster は、データサービスの静的な構成を定義するために使用する、リソースタイププロパティおよびリソースプロパティのセットを提供します。リソースタイププロパティでは、リソースのタイプ、そのバージョン、API のバージョンと同時に、各コールバックメソッドへのパスも指定します。253 ページの「[資源タイプのプロパティ](#)」に、すべてのリソースタイププロパティのリストを示します。

リソースプロパティ (Failover\_mode、Thorough\_probe\_interval など) やメソッドタイムアウトも、リソースの静的な構成を定義します。動的なリソースプロパティ (Resource\_state や Status など) は、管理対象のリソースの活動状況を反映します。リソースプロパティについては、264 ページの「[リソースのプロパティ](#)」を参照してください。

リソースタイプおよびリソースプロパティは、データサービスの重要な要素であるリソースタイプ登録 (RTR) ファイルで宣言します。RTR ファイルは、クラスタ管理者が Sun Cluster ソフトウェアでデータサービスを登録するとき、データサービスの初期構成を定義します。

独自のデータサービス用の RTR ファイルを生成するには、Agent Builder を使用します。Agent Builder では、すべてのデータサービスで有益かつ必須である、一連のプロパティを宣言します。たとえば、特定のプロパティ (`Resource_type` など) は RTR ファイルで宣言する必要があります。宣言されていない場合、データサービスの登録は失敗します。必須ではなくても、そのほかのプロパティも RTR ファイルで宣言されていないければ、クラスタ管理者はそれらのプロパティを利用できません。いくつかのプロパティは宣言されているかどうかにかかわらず使用できますが、これは RGM がそのプロパティを定義し、そのデフォルト値を提供しているためです。このような複雑さを回避するためにも、Agent Builder を使用して、適切な RTR ファイルを生成するようにしてください。後に、必要であれば RTR ファイルを編集して、特定の値を変更できます。

以降では、Agent Builder で作成した RTR ファイルの例を示します。

## リソースタイププロパティの宣言

クラスタ管理者は、RTR ファイルで宣言されているリソースタイププロパティを構成することはできません。このようなリソースタイププロパティは、リソースタイプの恒久的な構成の一部を形成します。

---

注 - リソースタイププロパティ `Installed_nodes` は、クラスタ管理者のみが構成できます。RTR ファイルでは `Installed_nodes` を宣言できません。

---

リソースタイプ宣言の構文は次のようになります。

```
property-name = value;
```

---

注 - リソースグループ、リソース、およびリソースタイプのプロパティ名は大文字と小文字が区別されません。プロパティ名を指定する際には、大文字と小文字を任意に組み合わせることができます。

---

次に、サンプルのデータサービス (`smpl`) 用の RTR ファイルにおけるリソースタイプ宣言を示します。

```
# 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 = "smp1";
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          =   smp1_svc_start;
Stop           =   smp1_svc_stop;

Validate       =   smp1_validate;
Update         =   smp1_update;

Monitor_start  =   smp1_monitor_start;
Monitor_stop   =   smp1_monitor_stop;
Monitor_check  =   smp1_monitor_check;
```

---

ヒント-RTR ファイルの最初のエントリには、`Resource_type` プロパティを宣言する必要があります。最初のエントリで宣言されていない場合は、リソースタイプの登録に失敗します。

---

リソースタイプ宣言の最初のセットは、リソースタイプについての基本的な情報を提供します。

#### `Resource_type` および `Vendor_id`

リソースタイプの名前を提供します。リソースタイプ名は `Resource_type` プロパティ (この例では「`smp1`」) 単独で指定できます。 `Vendor_id` プロパティを接頭辞として使用し、リソースタイプ (この例では「`SUNW.smp1`」) との区切りにピリオド (.) を使用することもできます。 `Vendor_id` を使用する場合、リソースタイプを定義する企業の略号にします。リソースタイプ名はクラスタ内で一意である必要があります。

---

注-便宜上、リソースタイプ名 (*vendoridApplicationname*) はパッケージ名として使用されます。Solaris 9 OS 以降では、ベンダー ID とアプリケーション名の両方を合わせて 10 文字以上を指定できます。

一方、Agent Builder はどの場合でもリソースタイプ名からパッケージ名を系統だてて生成します。つまり、Agent Builder は 9 文字の制限を適用します。

---

**RT\_description**

リソースタイプの簡潔な説明です。

**RT\_version**

サンプルデータサービスのバージョンです。

**API\_version**

API のバージョンです。たとえば、`API_version = 2` は、データサービスを Sun Cluster 3.0 以降の任意のバージョンの Sun Cluster にインストールできることを示します。`API_version = 7` は、データサービスを Sun Cluster 3.2 以降の任意のバージョンの Sun Cluster にインストールできることを示します。ただし、`API_version = 7` は、Sun Cluster 3.2 よりも前にリリースされたどのバージョンの Sun Cluster にもデータサービスをインストールできないことも示します。このプロパティについては、[253 ページの「資源タイプのプロパティ」](#) の `API_version` の項目で詳しく説明しています。

**Failover = TRUE**

データサービスが、複数のノードまたはゾーン上で同時にオンラインにできるリソースグループ上では実行できないことを示します。つまり、この宣言はフェイルオーバーデータサービスを指定しています。このプロパティは、[253 ページの「資源タイプのプロパティ」](#) の `Failover` のエントリでより詳細に説明されています。

**Start, Stop, Validate**

RGM によって呼び出されるコールバックメソッドプログラムのパスを提供します。これらのパスは、`RT_basedir` で指定されたディレクトリからの相対パスです。

残りのリソースタイプ宣言は、構成情報を提供します。

**Init\_nodes = RG\_PRIMARYES**

データサービスをマスターできるノードまたはゾーン上でのみ、RGM が `Init`、`Boot`、`Fini`、および `Validate` メソッドを呼び出すことを指定します。`RG_PRIMARYES` で指定されたノードまたはゾーンは、データサービスがインストールされているすべてのノードまたはゾーンのサブセットです。この値に `RT_INSTALLED_NODES` を設定した場合、データサービスがインストールされているすべてのノードまたはゾーン上で、RGM が上記メソッドを呼び出すことを指定します。

**RT\_basedir**

コールバックメソッドパスのような完全な相対パスとして、`/opt/SUNWsample/bin`をポイントします。

**Start、Stop、Validate**

RGMによって呼び出されるコールバックメソッドプログラムのパスを提供します。これらのパスは、`RT_basedir`で指定されたディレクトリからの相対パスです。

## リソースプロパティの宣言

リソースタイププロパティと同様に、リソースプロパティもRTRファイルで宣言します。便宜上、リソースプロパティ宣言はRTRファイルのリソースタイププロパティ宣言の後に行います。リソース宣言の構文では、一連の属性と値のペアを記述して、全体を中括弧({})で囲みます。

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

Sun Clusterが提供するリソースプロパティ(つまり、「システム定義プロパティ」)の場合、特定の属性はRTRファイルで変更できます。たとえば、Sun Clusterは各コールバックメソッドのメソッドタイムアウトプロパティのデフォルト値を提供します。RTRファイルを使用すると、異なるデフォルト値を指定できます。

RGMメソッドコールバックがタイムアウトすると、メソッドのプロセスツリーが、SIGTERMシグナルではなく、SIGABRTシグナルによって消去されます。結果として、プロセスグループのすべてのメンバーが、`/var/cluster/core`ディレクトリにコアダンプファイルを生成します。このコアダンプファイルは、メソッドがタイムアウトを超過した理由を判定できるように生成されます。

注- 新しいプロセスグループを作成するデータサービスメソッドを書かないでください。データサービスメソッドで新しいプロセスグループを作成する必要がある場合は、SIGTERM および SIGABRT シグナルのシグナルハンドラを書きます。また、シグナルハンドラは、プロセスを終了する前に、単数または複数の子プロセスグループに SIGTERM または SIGABRT シグナルを転送する必要があります。これらのシグナルのシグナルハンドラを書くと、使用するメソッドによって生成されるすべてのプロセスが、正しく終了される可能性が高まります。

Sun Cluster が提供するプロパティ属性のセットを使用することにより、RTR ファイル内に新しいリソースプロパティ (拡張プロパティ) を定義することもできます。301 ページの「リソースプロパティの属性」に、リソースプロパティを変更および定義するための属性を示します。拡張プロパティ宣言は RTR ファイルのシステム定義プロパティ宣言のあとに行います。

システム定義リソースプロパティの最初のセットでは、コールバックメソッドのタイムアウト値を指定します。

...

```
# リソースプロパティ宣言は中括弧で囲まれたエントリのリストであり、
# リソースタイププロパティ宣言のあとで宣言する。
# プロパティ名宣言は、リソースプロパティエントリの左中括弧の
# 直後にある最初の属性でなければならない。
#
# メソッドタイムアウト用の最小値とデフォルト値を設定する。
{
    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;
}
```

プロパティ名 (PROPERTY = *value*) は、各リソースプロパティ宣言における最初の属性でなければなりません。リソースプロパティは、RTR ファイルのプロパティ属性で定義された制限内で構成することができます。たとえば、各メソッドタイムアウト用のデフォルト値は 300 秒です。クラスタ管理者はこの値を変更できます。ただし、指定できる最小値は (MIN 属性で指定されているように) 60 秒です。301 ページの「リソースプロパティの属性」にリソースプロパティ属性のリストを示します。

リソースプロパティの次のセットは、データサービスにおいて特定の目的に使用されるプロパティを定義します。

```
{
    PROPERTY = Failover_mode;
    DEFAULT=SOFT;
    TUNABLE = ANYTIME;
}
{
    PROPERTY = Thorough_Probe_Interval;
    MIN=1;
    MAX=3600;
    DEFAULT=60;
    TUNABLE = ANYTIME;
}
# ある期限内に再試行する回数。この回数を超えると、
# 当該ノード上ではアプリケーションが起動できないと判断される。
{
    PROPERTY = Retry_count;
    MAX=10;
    DEFAULT=2;
    TUNABLE = ANYTIME;
}
```

```
# Retry_Interval に 60 の倍数を設定する。
# この値は秒から分に変換され、切り上げられる。
# たとえば、50 秒は 1 分に変更される。このプロパティを使用して、
# 再試行回数 (Retry_Count) を指定する。
{
    PROPERTY = Retry_interval;
    MAX=3600;
    DEFAULT=300;
    TUNABLE = ANYTIME;
}

{
    PROPERTY = Network_resources_used;
    TUNABLE = WHEN_DISABLED;
    DEFAULT = "";
}

{
    PROPERTY = Scalable;
    DEFAULT = FALSE;
    TUNABLE = AT_CREATION;
}

{
    PROPERTY = Load_balancing_policy;
    DEFAULT = LB_WEIGHTED;
    TUNABLE = AT_CREATION;
}

{
    PROPERTY = Load_balancing_weights;
    DEFAULT = "";
    TUNABLE = ANYTIME;
}

{
    PROPERTY = Port_list;
    TUNABLE = ANYTIME;
    DEFAULT = ;
}
```

これらのリソースプロパティ宣言には、`TUNABLE` 属性が含まれています。この属性は、この属性が関連付けられているプロパティの値をクラスタ管理者が変更できる場合を制限します。たとえば値 `AT_CREATION` は、クラスタ管理者が値を指定できるのはリソースの作成時だけであり、あとでは値を変更できないことを示します。

上記のプロパティのほとんどは、特に理由がない限り、`Agent Builder` が生成するデフォルト値を使用しても問題ありません。こうしたプロパティのあとには、次のような情報が続きます。詳細は、[264 ページの「リソースのプロパティ」](#) または `r_properties(5)` のマニュアルページを参照してください。

**Failover\_mode**

Start または Stop メソッドの失敗時、RGM がリソースグループを再配置するか、ノードまたはゾーンを停止するかを指定します。

**Thorough\_probe\_interval, Retry\_count, and Retry\_interval**

障害モニターで使用します。Tunable は ANYTIME に等しいため、障害モニターが適切に機能していない場合、クラスタ管理者はいつでも調整できます。

**Network\_resources\_used**

データサービスで使用される論理ホスト名または共有アドレスリソースのリスト。Agent Builder がこのプロパティを宣言するため、クラスタ管理者はデータサービスを構成するとき (存在する場合) に、リソースのリストを指定できます。

**Scalable**

この値を FALSE に設定した場合、このリソースがクラスタネットワーキング (共有アドレス) 機能を使用しないことを示します。このプロパティを FALSE に設定した場合、リソースタイププロパティ Failover を TRUE に設定して、フェイルオーバーサービスを指定する必要があります。このプロパティの詳細な使用方法については、34 ページの「データサービスをクラスタに転送する方法」および 44 ページの「コールバックメソッドの実装」を参照してください。

**Load\_balancing\_policy and Load\_balancing\_weights**

これらのプロパティを自動的に宣言します。ただし、これらのプロパティはフェイルオーバーリソースタイプでは使用されません。

**Port\_list**

サーバーが待機するポートのリストです。Agent Builder がこのプロパティを宣言するため、クラスタ管理者はデータサービスを構成するとき (存在する場合) に、リソースのリストを指定できます。

## 拡張プロパティの宣言

拡張プロパティは、サンプル RTR ファイルの最後に出現します。

```
# 拡張プロパティ
#
# クラスタ管理者は、このプロパティに値を設定して、アプリケーション
# が使用する構成ファイルが格納されているディレクトリを指定する
# 必要がある。このアプリケーション (smp1) の場合、PXFS 上に
# ある構成ファイル (通常は named.conf) のパスを指定する。
{
    PROPERTY = Confdir_list;
    EXTENSION;
    STRINGARRAY;
    TUNABLE = AT_CREATION;
    DESCRIPTION = "The Configuration Directory Path(s)";
}
```

```

# 次の2 つのプロパティは、障害モニターの再起動を制御する。
{
    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.";
}
# 検証用のタイムアウト値 (秒)。
{
    PROPERTY = Probe_timeout;
    EXTENSION;
    INT;
    DEFAULT = 120;
    TUNABLE = ANYTIME;
    DESCRIPTION = "Time out value for the probe (seconds)";
}

# PMF 用の子プロセス監視レベル (pmfadm の -C オプション)。
# デフォルトの -1 は、pmfadm -C オプションを使用しないこと
# を示す。
# 0 より大きな値は、目的の子プロセス監視レベルを示す。
{
    PROPERTY = Child_mon_level;
    EXTENSION;
    INT;
    DEFAULT = -1;
    TUNABLE = ANYTIME;
    DESCRIPTION = "Child monitoring level for PMF";
}
# ユーザー追加コード -- BEGIN VVVVVVVVVVV
# ユーザー追加コード -- END   ^^^^^^^^^^^^^

```

Agent Builder は、ほとんどのデータサービスにとって有用な、次の拡張プロパティを作成します。

#### Confdir\_list

アプリケーション構成ディレクトリへのパスを指定します。このプロパティは多くのアプリケーションにとって有用な情報です。データサービスを構成するときに、クラスタ管理者はこのディレクトリの場所を指定できます。

#### Monitor\_retry\_count, Monitor\_retry\_interval, and Probe\_timeout

サーバーデーモンではなく、障害モニター自体の再起動を制御します。

#### Child\_mon\_level

PMFによる監視レベルを設定します。詳細は、`pmfadm(1M)`のマニュアルページを参照してください。

「ユーザー追加コード」というコメントで区切られた領域に、追加の拡張プロパティを作成できます。

## コールバックメソッドの実装

この節では、コールバックメソッドの実装に関する一般的な情報について説明します。

## リソースとリソースグループのプロパティ情報へのアクセス

一般に、コールバックメソッドはリソースのプロパティにアクセスする必要があります。RMAPIは、リソースのシステム定義プロパティと拡張プロパティにアクセスするために、コールバックメソッドで使用できるシェルコマンドとC関数の両方を提供します。詳細は、`scha_resource_get(1HA)`と`scha_resource_get(3HA)`のマニュアルページを参照してください。

DSDLは、システム定義プロパティにアクセスするためのC関数セット(プロパティごとに1つ)と、拡張プロパティにアクセスするための関数を提供します。`scds_property_functions(3HA)`および`scds_get_ext_property(3HA)`のマニュアルページを参照してください。

`Status`と`Status_msg`を除き、リソースプロパティを設定するAPI関数が存在しないため、プロパティ機構を使用して、データサービスの動的な状態情報を格納することはできません。したがって、動的な状態情報は、広域ファイルに格納するようにします。

---

注-クラスタ管理者は、`clresource` コマンド、グラフィカル管理コマンド、またはグラフィカル管理インタフェースを使用して、特定のリソースプロパティを設定することができます。ただし、`clresource` はクラスタの再構築時に (つまり、RGM がメソッドを呼び出した時点で) エラー終了するため、どのようなコールバックメソッドからも `clresource` を呼び出さないようにします。

---

## メソッドの呼び出し回数への非依存性

一般に、RGM は、同じリソース上で同じメソッドを (同じ引数で) 何回も連続して呼び出すことはありません。ただし、`Start` メソッドが失敗した場合には、リソースが起動していなくても、RGM はそのリソース上で `Stop` メソッドを呼び出すことができます。同様に、リソースデーモンが自発的に停止している場合でも、RGM はそのリソース上で `Stop` メソッドを実行できます。`Monitor_start` メソッドと `Monitor_stop` メソッドにも、同じことが当てはまります。

このような理由のため、`Stop` メソッドと `Monitor_stop` メソッドには「呼び出し回数への非依存性」を組み込む必要があります。つまり、同じリソース上で、同じ引数を指定して `Stop` または `Monitor_stop` を連続して呼び出しても、1 回だけ呼び出したときと同じ結果になる必要があります。

呼び出し回数に依存しないということは、リソースまたはモニターがすでに停止し、行うべき作業がなくても、`Stop` メソッドと `Monitor_stop` メソッドが 0 (成功) を返す必要があるということも意味します。

---

注-`Init`、`Fini`、`Boot`、`Update` の各メソッドも呼び出し回数に依存しない必要があります。`Start` メソッドは呼び出し回数に依存してもかまいません。

---

## メソッドがゾーンで呼び出される仕組み

`Global_zone` リソースタイププロパティは、RTR ファイルで宣言すると、リソースタイプのメソッドが大域ゾーン内で実行されるかどうかを示します。`Global_zone` プロパティが `TRUE` に等しい場合、リソースを含むリソースグループが非大域ゾーンで動作するように構成されているときでも、メソッドは大域ゾーンで実行されます。

`Global_zone` が `TRUE` に等しいリソースが非大域ゾーン内で構成されている場合、大域ゾーン内で呼び出されるメソッドは `-Zzonename` オプション付きで呼び出されます。`zonename` オペランドは、リソースが実際に構成されているローカルノード上のゾーンの名前を示します。このオペランドの値がメソッドに渡されます。

リソースが大域ゾーン内で構成されている場合には、`-Zzonename` オプションは呼び出されず、非大域ゾーン名がメソッドに渡されることはありません。

Global\_zone リソースタイププロパティについては、253 ページの「資源タイプのプロパティ」を参照してください。

## 汎用データサービス

汎用データサービス (GDS) は、単純なアプリケーションを Sun Cluster Resource Group Manager (RGM) フレームワークに組み込むことにより、単純なアプリケーションの高可用性とスケーラビリティを実現する機構です。この機構では、アプリケーションの可用性やスケーラビリティを高めるための一般的な方法である、データサービスのコーディングは必要ありません。

GDS モデルは、コンパイル済みのリソースタイプ SUNW.gds により、RGM フレームワークとやりとりします。詳細は、[第 10 章](#)を参照してください。

## アプリケーションの制御

コールバックメソッドを使用すると、RGM は基になるリソース (アプリケーション) を制御できるようになります。たとえば、ノードまたはゾーンがクラスタに結合するとき、またはクラスタから分離するとき、コールバックメソッドを使用することにより、RGM は影響下にあるリソースを制御できるようになります。

## リソースの起動と停止

リソースタイプを実装するには、少なくとも、Start メソッドと Stop メソッドが必要です。

### Start および Stop メソッドの使用

RGM は、リソースタイプのメソッドプログラムを、適切なノードまたはゾーン上で適切な回数だけ呼び出して、リソースグループをオフラインまたはオンラインにします。たとえば、クラスタノードまたはゾーンのクラッシュ後、RGM は、そのノードまたはゾーンがマスターしているリソースグループを新しいノードまたはゾーンに移動します。この場合、Start メソッドを実装することによって、(ほかにも提供されるものがありますが) 生き残ったホストノードまたはゾーン上で各リソースを再起動するための手段を、RGM に提供する必要があります。

Start メソッドは、ローカルノードまたはゾーン上でリソースが起動し、使用可能な状態になるまで終了してはいけません。初期化に時間がかかるリソースタイプでは、その Start メソッドに、十分な長さのタイムアウト値を設定する必要があります。十分なタイムアウトを確保するには、RTR ファイルで Start\_timeout プロパティのデフォルトと最小の値を設定します。

Stop メソッドは、RGM がリソースをオフラインにする状況に合わせて実装する必要があります。たとえば、リソースがノード 1 上のゾーン A 内でオフラインにされ、ノード 2 上のゾーン B 内でオンラインにされるとします。リソースグループをオフラインにしている間、RGM は、そのリソースグループ内のリソース上で Stop メソッドを呼び出して、ノード 1 上のゾーン A 内のすべての活動を停止しようとしています。ノード 1 上のゾーン A 内ですべてのリソースの Stop メソッドが完了したら、RGM は、ノード 2 上のゾーン B 内でそのリソースグループを再度オンラインにします。

Stop メソッドは、ローカルノードまたはゾーン上でリソースがすべての活動を完全に停止し完全にシャットダウンするまで終了してはいけません。もっとも安全な Stop メソッドの実装方法は、ローカルノードまたはゾーン上でリソースに関連するすべてのプロセスを終了することです。シャットダウンに時間がかかるリソースタイプでは、十分な長さのタイムアウト値をその Stop メソッドに設定する必要があります。Stop\_timeout プロパティは RTR ファイルで設定します。

RGM メソッドコールバックがタイムアウトすると、メソッドのプロセスツリーが、SIGTERM シグナルではなく、SIGABRT シグナルによって消去されます。結果として、プロセスグループのすべてのメンバーが、/var/cluster/core ディレクトリにコアダンプファイルを生成します。このコアダンプファイルは、メソッドがタイムアウトを超過した理由を判定できるように生成されます。

---

注- 新しいプロセスグループを作成するデータサービスメソッドを書かないください。データサービスメソッドで新しいプロセスグループを作成する必要がある場合は、SIGTERM および SIGABRT シグナルのシグナルハンドラを書きます。また、シグナルハンドラは、プロセスを終了する前に、単数または複数の子プロセスグループに SIGTERM または SIGABRT シグナルを転送する必要があります。これらのシグナルのシグナルハンドラを書くと、使用するメソッドによって生成されるすべてのプロセスが、正しく終了される可能性が高まります。

---

Stop メソッドが失敗またはタイムアウトすると、リソースグループはエラー状態になり、クラスタ管理者の介入が必要となります。この状態を回避するには、Stop および Monitor\_stop メソッドがすべてのエラー状態から回復するようにする必要があります。理想的には、これらのメソッドは 0 (成功) のエラー状態で終了し、ローカルノードまたはゾーン上でリソースとそのモニターのすべての活動を正常に停止する必要があります。

## Start および Stop メソッドを使用するかどうかの決定

この節では、Start メソッドと Stop メソッドを使用するか、または、Prenet\_start メソッドと Postnet\_stop メソッドを使用するかを決定するときのいくつかの注意事項について説明します。使用する適切なメソッドを決定するには、クライアントおよびデータサービスのクライアントサーバー型ネットワークプロトコルについて十分に理解している必要があります。

ネットワークアドレスリソースを使用するサービスでは、起動または停止の手順を特定の順序で実行しなければならない場合があります。この順序は、論理ホスト名アドレスの構成を基準とする必要があります。オプションのコールバックメソッド `Prenet_start` と `Postnet_stop` を使用してリソースタイプを実装すると、同じリソースグループ内のネットワークアドレスが「起動」に構成される前、または「停止」に構成されたあとに、特別な起動処理または停止処理を行います。

RGM は、データサービスの `Prenet_start` メソッドを呼び出す前に、ネットワークアドレスを取り付ける (`plumb`、ただし起動には構成しない) メソッドを呼び出します。RGM は、データサービスの `Postnet_stop` メソッドを呼び出したあとに、ネットワークアドレスを取り外す (`unplumb`) メソッドを呼び出します。

RGM がリソースグループをオンラインにするときは、次のような順番になります。

1. ネットワークアドレスを取り付けます。
2. データサービスの `Prenet_start` メソッドを呼び出します (存在する場合)。
3. ネットワークアドレスを起動状態に構成します。
4. データサービスの `Start` メソッドを呼び出します (存在する場合)。

RGM がリソースグループをオフラインにするときは、逆の順番になります。

1. データサービスの `Stop` メソッドを呼び出します (存在する場合)。
2. ネットワークアドレスを停止状態に構成します。
3. データサービスの `Postnet_stop` メソッドを呼び出します (存在する場合)。
4. ネットワークアドレスを取り外します。

`Start`、`Stop`、`Prenet_start`、`Postnet_stop` のうち、どのメソッドを使用するかを決定するには、まずサーバー側を考慮します。データサービスアプリケーションリソースとネットワークアドレスリソースの両方を持つリソースグループをオンラインにするとき、RGM は、データサービスリソースの `Start` メソッドを呼び出す前に、ネットワークアドレスを起動状態に構成するメソッドを呼び出します。したがって、データサービスを起動するときにネットワークアドレスが「起動」に構成されている必要がある場合は、`Start` メソッドを使用してデータサービスを起動します。

同様に、データサービスアプリケーションリソースとネットワークアドレスリソースの両方を持つリソースグループをオフラインにするとき、RGM は、データサービスリソースの `Stop` メソッドを呼び出したあとに、ネットワークアドレスを停止状態に構成するメソッドを呼び出します。したがって、データサービスを停止するときにネットワークアドレスが「起動」に構成されている必要がある場合は、`Stop` メソッドを使用してデータサービスを停止します。

たとえば、データサービスを起動または停止するために、データサービスの管理ユーティリティまたはライブラリを実行しなければならない場合があります。また、クライアントサーバー型ネットワークインタフェースを使用して管理を実行するような管理ユーティリティまたはライブラリを持っているデータサービスもあります。つまり、管理ユーティリティがサーバーデーモンを呼び出すので、管理

ユーティリティまたはライブラリを使用するためには、ネットワークアドレスが「起動」に構成されている必要があります。このような場合は、Start メソッドと Stop メソッドを使用します。

データサービスが起動および停止するときにネットワークアドレスが「停止」に構成されている必要がある場合は、Prenet\_start メソッドと Postnet\_stop メソッドを使用してデータサービスを起動および停止します。クラスタ再構成 (SCHA\_GIVEOVER 引数を指定した scha\_control() または clnode evacuate コマンドによるスイッチオーバー) のあとネットワークアドレスとデータサービスのどちらが最初にオンラインになるかによってクライアントソフトウェアの応答が異なるかどうかを考えます。たとえば、クライアントの実装が最小限の再試行を行うだけで、データサービスのポートが利用できないと判断すると、すぐにあきらめる場合もあります。

データサービスを起動するときにネットワークアドレスが「起動」に構成されている必要がない場合、ネットワークインタフェースが「起動」に構成される前に、データサービスを起動します。このようにデータサービスを起動することで、ネットワークアドレスが「起動」に構成されるとすぐに、データサービスはクライアントの要求に応答できます。その結果、クライアントが再試行を停止する可能性も減ります。このような場合は、Start ではなく、Prenet\_start メソッドを使用してデータサービスを起動します。

Postnet\_stop メソッドを使用した場合、ネットワークアドレスが「停止」に構成されている時点では、データサービスリソースは「起動」のままです。Postnet\_stop メソッドを実行するのは、ネットワークアドレスが「停止」に構成されたあとだけです。結果として、データサービスの TCP または UDP のサービスポート (つまり、その RPC プログラム番号) は、常に、ネットワーク上のクライアントから利用できます。ただし、ネットワークアドレスも応答しない場合を除きます。

---

注-クラスタに RPC サービスをインストールする場合、サービスはプログラム番号 100141、100142、および 100248 を使用できません。これらの番号は、Sun Cluster デーモン rgmd\_receptionist、fed、および pmfd 用に予約されています。インストールした RPC サービスがこれらのプログラム番号のいずれかを使用する場合は、RPC サービスのプログラム番号を変更します。

---

Start と Stop メソッドを使用するか、Prenet\_start と Postnet\_stop メソッドを使用するか、または両方を使用するかを決定するには、サーバーとクライアント両方の要件と動作を考慮に入れる必要があります。

## Init、Fini、Boot オプションメソッドの使用

3つのオプションメソッドである Init、Fini、Boot を使用すると、RGM がリソースで初期化コードと終了コードを実行できるようになります。

## Init メソッドの使用

次の条件のいずれかの結果としてリソースが管理下に置かれる場合、RGM は Init メソッドを実行して、1 回だけリソースの初期化を実行します。

- リソースが属しているリソースグループを、管理されていない状態から管理されている状態に切り替える。
- すでに管理されているリソースグループでリソースが作成される。

## Fini メソッドの使用

リソースが RGM によって管理されなくなったとき、RGM は Fini メソッドを実行して、そのリソースの使用後のクリーンアップを行います。Fini メソッドは、通常、Init メソッドにより実行された初期化を元に戻します。

RGM は、次の条件が成り立つときに、リソースが管理されなくなったノードまたはゾーン上で Fini を実行します。

- リソースを含むリソースグループが管理対象外状態に切り替わる。この場合、ノードリストのすべてのノードおよびゾーン上で、RGM は Fini メソッドを実行します。
- 管理されているリソースグループからリソースが削除される。この場合、ノードリストのすべてのノードおよびゾーン上で、RGM は Fini メソッドを実行します。
- リソースを含むリソースグループのノードリストからノードまたはゾーンが削除される。この場合、削除されたノードまたはゾーン上でのみ、RGM は Fini メソッドを実行します。

「ノードリスト」はリソースグループの `NodeList` またはリソースタイプの `Installed_nodes` リストのいずれかです。「ノードリスト」がリソースグループの `NodeList` とリソースタイプの `Installed_nodes` リストのどちらを指すかは、リソースタイプの `Init_nodes` プロパティの設定に依存します。`Init_nodes` プロパティは `RG_nodelist` または `RT_installed_nodes` に設定できます。大部分のリソースタイプでは、`Init_nodes` はデフォルトである `RG_nodelist` に設定されます。この場合、Init メソッドと Fini メソッドは両方とも、リソースグループの `NodeList` で指定されているノードおよびゾーン上で実行されます。

Init メソッドが実行する初期化の種類は、次のように、ユーザーが実装した Fini メソッドが実行する必要があるクリーンアップの種類を定義します。

- ノード固有の構成のクリーンアップ。
- クラスタ全体の構成のクリーンアップ。

## Fini メソッドを実装する際のガイドライン

実装する Fini メソッドは、ノード固有の構成だけをクリーンアップするのか、それともノード固有の構成とクラスタ全体にわたる構成の両方をクリーンアップするのかを判断する必要があります。

リソースが特定のノードまたはゾーン上でのみ、管理されなくなった場合、Fini メソッドはノード固有のローカル構成をクリーンアップできます。しかし、ほかのノード上ではリソースは引き続き管理されているため、Fini メソッドはクラスタ全体にわたるグローバル構成をクリーンアップしてはなりません。リソースがクラスタ全体にわたって管理されなくなった場合には、Fini メソッドはノード固有の構成とグローバル構成の両方についてクリーンアップを実行できます。実装する Fini メソッドのコードは、Fini メソッドを実行するローカルのノードまたはゾーンがリソースグループのノードリストに含まれているかどうかを調べることによって、これら2つの場合を区別できます。

ローカルのノードまたはゾーンがリソースグループのノードリストに出現している場合は、リソースが削除されようとしているか、管理されない状態に移行しようとしています。リソースはどのノードまたはゾーン上でもアクティブでなくなっています。この場合、実装する Fini メソッドでは、ローカルノード上のノード固有の構成だけでなく、クラスタ全体にわたる構成についてもクリーンアップする必要があります。

ローカルのノードまたはゾーンがリソースグループのノードリストに出現していない場合は、Fini メソッドでそのローカルのノードまたはゾーン上のノード固有の構成をクリーンアップできます。しかし、Fini メソッドでクラスタ全体にわたる構成をクリーンアップしてはなりません。この場合、ほかのノードまたはゾーン上でリソースが引き続きアクティブになっています。

また、Fini は呼び出し回数に依存しないようにコーディングする必要があります。つまり、Fini メソッドが以前の実行でリソースをクリーンアップした場合でも、以降の Fini 呼び出しは正常に終了します。

## Boot メソッドの使用

RGM は、クラスタに結合した (つまり、ブートまたはリブートしたばかりの) ノードまたはゾーンで、Boot メソッドを実行します。

Boot メソッドは、通常、Init と同じ初期化を実行します。Boot は呼び出し回数に依存しないようにコーディングする必要があります。つまり、Boot メソッドが以前の実行でリソースを初期化した場合でも、以降の Boot 呼び出しは正常に終了します。

## リソースの監視

通常、モニターは、リソース上で定期的に障害検証を実行し、検証したリソースが正しく動作しているかどうかを検出するように実装します。障害検証が失敗した場合、モニターはローカルでの再起動を試みるか、影響を受けるリソースグループのフェイルオーバーを要求できます。モニターは、RMAPI関数 `scha_control()` または `scha_control_zone()` を呼び出すか、あるいは DSDL 関数 `scds_fm_action()` を呼び出すことによって、フェイルオーバーを要求します。

また、リソースの性能を監視して、性能を調節または報告することもできます。リソースタイプに固有な障害モニターの作成は任意です。このような障害モニターを作成しなくても、リソースタイプは Sun Cluster により基本的なクラスタの監視が行われます。Sun Cluster は、ホストハードウェアの障害、ホストのオペレーティングシステムの全体的な障害、およびパブリックネットワーク上で通信できるホストの障害を検出します。

RGM がリソースモニターを直接呼び出すことはありませんが、RGM は自動的にリソース用のモニターを起動する準備を整えます。リソースをオフラインにするとき、RGM は、リソース自体を停止する前に、`Monitor_stop` メソッドを呼び出して、ローカルノードまたはゾーン上でリソースのモニターを停止します。リソースをオンラインにするとき、RGM は、リソース自体を起動したあとに、`Monitor_start` メソッドを呼び出します。

RMAPI 関数 `scha_control()` または `scha_control_zone()`、および DSDL 関数 `scds_fm_action()` (この関数は `scha_control()` を呼び出す) を使用することにより、リソースモニターはリソースグループを別のノードまたはゾーンにフェイルオーバーするよう要求できます。妥当性検査の 1 つとして、`scha_control()` および `scha_control_zone()` は、`Monitor_check` を呼び出して (定義されている場合)、要求されたノードまたはゾーンがリソースのあるリソースグループをマスターできるほど信頼できるかどうかを判断します。`Monitor_check` が「このノードまたはゾーンは信頼できない」と報告した場合、あるいは、メソッドがタイムアウトした場合、RGM はフェイルオーバー要求に適する別のノードまたはゾーンを探します。すべてのノードまたはゾーンで `Monitor_check` が失敗した場合、フェイルオーバーは取り消されます。

リソースモニターは、モニターから見たリソースの状態を反映するように `Status` と `Status_msg` プロパティを設定します。これらのプロパティを設定するには、RMAPI 関数 `scha_resource_setstatus()` または `scha_resource_setstatus_zone()`、`scha_resource_setstatus` コマンド、あるいは DSDL 関数 `scds_fm_action()` を使用します。

---

注 - `Status` および `Status_msg` プロパティはリソースモニターに固有の使用方法ですが、これらのプロパティは任意のプログラムで設定できます。

---

RMAPIによる障害モニターの実装例については、109ページの「障害モニターの定義」を参照してください。DSDLによる障害モニターの実装例については、157ページの「SUNW.xfnts 障害モニター」を参照してください。Sunが提供するデータサービスに組み込まれている障害モニターについては、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』を参照してください。

## 大域ゾーン内でのみ実行されるモニターおよびメソッドの実装

ほとんどのリソースタイプは、リソースグループのノードリストに出現するすべてのゾーンまたはノードでメソッドを実行します。一部の少数のリソースタイプについては、リソースグループが非大域ゾーンで構成されている場合でも、大域ゾーンですべてのメソッドを実行する必要があります。これが必要となるのは、ネットワークアドレスやディスクなど、大域ゾーンからしか管理できないシステムリソースを管理しているリソースタイプの場合です。このようなリソースタイプは、リソースタイプ登録 (RTR) ファイルの中で `Global_zone` プロパティを `TRUE` に設定することによって識別されます。



注意 - 信頼できる既知のソースであるリソースタイプを除いて、`Global_zone` プロパティに `TRUE` が設定されているリソースタイプは登録しないでください。このプロパティに `TRUE` を設定したリソースタイプは、ゾーン分離をすり抜け、危険があります。

`Global_zone` リソースタイププロパティが `TRUE` に設定されていない場合、モニターやメソッドはリソースグループのノードリストに列挙されている任意のゾーンで実行されます。

`scha_control()` および `scha_resource_setstatus()` 関数、そして `scha_control` および `scha_resource_setstatus` コマンドは、それらの関数やコマンドの実行元のゾーンで暗黙的に動作します。`Global_zone` リソースタイププロパティが `TRUE` に等しい場合、これらの関数やコマンドは、リソースが非大域ゾーンで構成されているときに、別に呼び出される必要があります。

リソースが非大域ゾーンで構成されているときは、`zonename` オペランドの値が `-Z` オプションを通じてリソースタイプメソッドに渡されます。実装するメソッドやモニターからこれらのいずれかの関数やコマンドを呼び出す場合、正しい処理を行わないと、大域ゾーンで正しく動作しません。実装するメソッドやモニターは、リソースグループのノードリストに含まれているリソースが構成されている非大域ゾーンで動作するようにする必要があります。

実装するメソッドやモニターのコードでこれらの条件を正しく処理していることを確認するため、次の作業が行われていることをチェックしてください。

- `scha_control` および `scha_resource_setstatus` コマンド呼び出しで、`-Z zonename` オプションを指定する。`zonename` には、RGM が `-z` オプションを通じてデータサービスメソッドに渡すものと同じ値を使用する。
- `scha_control()` 関数への呼び出しではなく `scha_control_zone()` 関数への呼び出しを含める。呼び出しでは、`-z` オプションにより渡された `zonename` オペランドを必ず渡す。
- `scha_resource_setstatus()` 関数への呼び出しではなく `scha_resource_setstatus_zone()` 関数への呼び出しを含める。呼び出しでは、`-z` オプションにより渡された `zonename` オペランドを必ず渡す。

`Global_zone` リソースタイププロパティが `TRUE` に等しいリソースが、`ZONE_LOCAL` の問い合わせの `optag` の値を指定して `scha_cluster_get()` を起動した場合、大域ゾーンの名前が返されます。この場合、呼び出した側のコードでは文字列 `:zonename` をローカルノード名に連結して、リソースが実際に構成されているゾーンを取得する必要があります。

同様に、呼び出した側のコードで、たとえば非大域ゾーンでのリソースの状態を問い合わせる場合は、`RESOURCE_STATE` の `optag` 値ではなく `RESOURCE_STATE_NODE` の `optag` 値を指定して、`scha_resource_get()` を呼び出す必要があります。この場合、`RESOURCE_STATE` の `optag` 値によって、リソースが実際に構成されている非大域ゾーンでの問い合わせではなく、大域ゾーンでの問い合わせが実行されます。

DSDL 関数は、その性質上、`-Z zonename` オプションを処理します。したがって、`scds_initialize()` 関数は、リソースが実際に構成されている非大域ゾーンに対応した、該当するリソースプロパティおよびリソースグループプロパティを取得します。そのほかの DSDL 問い合わせは、そのゾーンの中で暗黙的に動作します。

DSDL 関数 `scds_get_zone_name()` を使用すると、`-Z zonename` コマンド行オプションの中でメソッドに渡されたゾーンの名前を問い合わせることができます。`-Z zonename` が渡されていない場合には、`scds_get_zone_name()` 関数は `NULL` を返します。

次の条件がどちらも成り立つ場合、複数の Boot メソッドが大域ゾーン内で同時に実行されることがあります。

- リソースグループの `Nodelist` に、同じ物理ノード上の複数のゾーンが含まれている。
- その同じリソースグループに、`Global_zone` プロパティが `TRUE` に設定されたりリソースが 1 つ以上含まれている。

## メッセージログのリソースへの追加

状態メッセージをほかのクラスタメッセージと同じログファイルに記録する場合は、`scha_cluster_getlogfacility()` 関数を使用して、クラスタメッセージを記録するために使用されている機能番号を取得します。

この機能番号を通常の Solaris `syslog()` 関数で使用して、状態メッセージをクラスタログに書き込みます。`scha_cluster_get()` 汎用インタフェースからでも、クラスタログ機能情報にアクセスできます。

## プロセス管理の提供

リソースモニターとリソース制御コールバックを実装するために、プロセス管理機能が RMAPI および DSDL に提供されています。RMAPI は次の機能を定義します。

プロセス監視機能 (Process Monitor Facility: PMF): `pmfadm` および `rpc.pmfd`  
プロセスとその子孫を監視し、プロセスが終了したときに再起動する手段を提供します。この機能は、監視するプロセスを起動および制御する `pmfadm` コマンドと、`rpc.pmfd` デーモンからなります。

PMF の機能を実装するため、DSDL は (前に名前 `scds_pmf_` が付く) 関数のセットを提供します。DSDL の PMF 機能の概要と、個々の関数のリストについては、[219 ページの「PMF 関数」](#)を参照してください。

このコマンドとデーモンについては、`pmfadm(1M)` および `rpc.pmfd(1M)` のマニュアルページを参照してください。

### `halockrun`

ファイルロックを保持しながら子プログラムを実行するためのプログラムです。このコマンドはシェルスクリプトで使用すると便利です。

このコマンドについては、`halockrun(1M)` のマニュアルページを参照してください。

### `hatimerun`

タイムアウト制御下で子プログラムを実行するためのプログラムです。このコマンドはシェルスクリプトで使用すると便利です。

DSDL では、`hatimerun` コマンドの機能を実装するための `scds_hatimerun()` 関数が提供されています。

このコマンドについては、`hatimerun(1M)` のマニュアルページを参照してください。

## リソースへの管理サポートの提供

リソース上でクラスタ管理者が実行するアクションには、リソースプロパティの設定と変更があります。このような管理アクションを行うコードを作成できるように、APIはValidateとUpdateというコールバックメソッドを定義しています。

リソースが作成される時、RGMは任意のValidateメソッドを呼び出します。また、クラスタ管理者がリソースまたはそのリソースのあるグループのプロパティを更新したときにも、RGMはValidateメソッドを呼び出します。RGMは、リソースとそのリソースグループのプロパティ値をValidateメソッドに渡します。RGMは、リソースのタイプのInit\_nodesプロパティが示すクラスタノードまたはゾーンのセット上でValidateを呼び出します。Init\_nodesについては、[253 ページの「資源タイプのプロパティ」](#)またはrt\_properties(5)のマニュアルページを参照してください。RGMは、作成または更新が行われる前にValidateを呼び出します。任意のノードまたはゾーン上でメソッドから失敗の終了コードが戻ってくると、作成または更新は失敗します。

RGMがValidateを呼び出すのは、クラスタ管理者がリソースまたはリソースグループのプロパティを変更したときだけです。RGMがプロパティを設定したときや、モニターがStatusとStatus\_msgリソースプロパティを設定したときではありません。

RGMは、オプションのUpdateメソッドを呼び出して、プロパティが変更されたことを実行中のリソースに通知します。RGMは、クラスタ管理者がリソースまたはそのグループのプロパティの設定に成功したあとに、Updateを実行します。RGMは、リソースがオンラインであるノード上またはゾーン内で、このメソッドを呼び出します。このメソッドは、APIアクセス関数を使用して、アクティブなリソースに影響する可能性があるプロパティ値を読み取り、その値に従って、実行中のリソースを調節できます。

## フェイルオーバーリソースの実装

フェイルオーバーリソースグループには、ネットワークアドレス(組み込みリソースタイプであるLogicalHostnameやSharedAddressなど)やフェイルオーバーリソース(フェイルオーバーデータサービス用のデータサービスアプリケーションリソースなど)があります。ネットワークアドレスリソースは、データサービスがフェイルオーバーまたはスイッチオーバーする場合に、依存するデータサービスリソースと共に、クラスタノード間またはゾーン間を移動します。RGMは、フェイルオーバーリソースの実装をサポートするプロパティをいくつか提供します。

フェイルオーバーリソースグループは、別のノード上または同じノード上のゾーンへのフェイルオーバーを実行できます。ただし、ノードで障害が発生すると、同一ノード上のゾーンに対するこのリソースグループのフェイルオーバーから高可用性は得られません。とはいえ、同一ノード上のゾーンに対するリソースグループのフェイルオーバーは、テストまたはプロトタイプ化の際に便利な場合もあります。

ブール型の `Failover` リソースタイププロパティを `TRUE` に設定し、同時に複数のノードまたはゾーン上でオンラインになることができるリソースグループだけで構成されるようにリソースを制限します。このプロパティのデフォルト値は `FALSE` です。したがって、フェイルオーバーリソースを実現するためには、`RTR` ファイルで `TRUE` として宣言する必要があります。

`Scalable` リソースプロパティは、リソースがクラスタ共有アドレス機能を使用するかどうかを決定します。フェイルオーバーリソースの場合、フェイルオーバーリソースは共有アドレスを使用しないので、`Scalable` を `FALSE` に設定します。

`RG_mode` リソースグループプロパティを使用すると、クラスタ管理者はリソースグループがフェイルオーバーまたはスケーラブルのどちらであるかを識別できます。`RG_mode` が `FAILOVER` の場合、`RGM` はリソースグループの `Maximum primaries` プロパティを `1` に設定します。また、`RGM` は、リソースグループが単一のノードまたはゾーンでマスターされるように制限します。`Failover` プロパティが `TRUE` に設定されているリソースを、`RG_mode` が `SCALABLE` のリソースグループで作成することはできません。

`Implicit_network_dependencies` リソースグループプロパティは、グループ内におけるすべてのネットワークアドレスリソース (`LogicalHostname` や `SharedAddress`) への非ネットワークアドレスリソースの暗黙で強力な依存関係を、`RGM` が強制することを指定します。その結果、グループ内のネットワークアドレスが「起動」に構成されるまで、グループ内の非ネットワークアドレス (データサービス) リソースの `Start` メソッドは呼び出されません。`Implicit_network_dependencies` プロパティのデフォルトは `TRUE` です。

## スケーラブルリソースの実装

スケーラブルリソースは、同時に複数のノード上でオンラインになることができます。スケーラブルなリソース (ネットワーク負荷分散を使用) を、非大域ゾーンでも動作するよう構成することができます。ただし、そのようなスケーラブルなリソースを実行できるのは、物理ノードごとに `1` つのゾーン内だけです。スケーラブルリソースには、`Sun Cluster HA for Sun Java System Web Server` (以前の `Sun Cluster HA for Sun ONE Web Server`) や `Sun Cluster HA for Apache` などのデータサービスがあります。

`RGM` は、スケーラブルリソースの実装をサポートするプロパティをいくつか提供します。

ブール型の `Failover` リソースタイププロパティを `FALSE` に設定し、一度に複数のノードでオンラインにできるリソースグループ内でリソースが構成されるようにします。

`Scalable` リソースプロパティは、リソースがクラスタ共有アドレス機能を使用するかどうかを決定します。スケーラブルサービスは共有アドレスリソースを使用するので (スケーラブルサービスの複数のインスタンスが単一のサービスであるかのようクライアントに見せるため)、`Scalable` には `TRUE` を設定します。

RG\_mode プロパティを使用すると、クラスタ管理者はリソースグループがフェイルオーバーまたはスケーラブルのどちらであるかを識別できます。RG\_mode が SCALABLE の場合、RGM は Maximum primaries に 1 より大きな値が割り当てられることを許可します。同時に複数のノードまたはゾーンがリソースグループをマスターできます。RGM は、Failover プロパティが FALSE であるリソースが、RG\_mode が SCALABLE であるリソースグループ内でインスタンス化されることを許可します。

クラスタ管理者は、スケーラブルサービスリソースを含めるためのスケーラブルリソースグループを作成します。また、スケーラブルリソースが依存する共有アドレスリソースを含めるためのフェイルオーバーリソースグループも別に作成します。

クラスタ管理者は、RG\_dependencies リソースグループプロパティを使用して、あるノードまたはゾーン上でリソースグループをオンラインまたはオフラインにする順番を指定します。スケーラブルリソースとそれらが依存する共有アドレスリソースは異なるリソースグループに存在するので、この順番はスケーラブルサービスにとって重要です。スケーラブルデータサービスが起動する前に、そのネットワークアドレス (共有アドレス) リソースが構成されていることが必要です。したがって、クラスタ管理者は (スケーラブルサービスが属するリソースグループの) RG\_dependencies プロパティを設定して、共有アドレスリソースが属するリソースグループを組み込む必要があります。

リソースの RTR ファイルで Scalable プロパティを宣言した場合、RGM はそのリソースに対して、次のようなスケーラブルプロパティのセットを自動的に作成します。

#### Network\_resources\_used

このリソースによって使用される共有アドレスリソースを特定します。このプロパティのデフォルト値は空の文字列です。したがって、クラスタ管理者はリソースを作成するときに、スケーラブルサービスが使用する共有アドレスの実際のリストを提供する必要があります。clsetup コマンドと Sun Cluster Manager は、スケーラブルサービスに必要なリソースとリソースグループを自動的に設定する機能を提供します。

#### Load\_balancing\_policy

リソースの負荷均衡ポリシーを指定します。このポリシーは RTR ファイルに明示的に設定できます (デフォルトの LB\_WEIGHTED を使用してもかまいません)。どちらの場合でも、クラスタ管理者はリソースを作成するときに値を変更できます (RTR ファイルで Load\_balancing\_policy の Tunable を NONE または FALSE に設定していない場合)。使用できる有効な値は次のとおりです。

#### LB\_WEIGHTED

Load\_balancing\_weights プロパティに設定されている重みにより、さまざまなノードに負荷が分散されます。

#### LB\_STICKY

スケーラブルサービスの指定のクライアント (クライアントの IP アドレスで識別される) は、常に同じクラスタノードに送信されます。

### LB\_STICKY\_WILD

指定のクライアント(クライアントのIPアドレスで識別される)はワイルドカードスティッキーサービスのIPアドレスに接続され、送信時に使用されるポート番号とは無関係に、常に同じクラスタノードに送信されます。

LB\_STICKYまたはLB\_STICKY\_WILDのLoad\_balancing\_policyを持つスケーラブルサービスの場合、サービスがオンラインの状態ではLoad\_balancing\_weightsを変更すると、既存のクライアントとの関連がリセットされることがあります。その場合、そのクラスタ内にある別のノードまたはゾーンによりクライアントが以前にサービスを受けていた場合であっても、別のノードまたはゾーンが後続のクライアント要求にサービスを提供する場合があります。

同様に、サービスの新しいインスタンスをクラスタ上で起動すると、既存のクライアントとの関連がリセットされることがあります。

### Load\_balancing\_weights

個々のノードへ送信される負荷を指定します。書式は、*weight@node,weight@node* です。*weight*は、指定された*node*に分散される負荷の相対的な割合を反映した整数です。ノードに分散される負荷の割合は、アクティブなインスタンスのすべてのウェイトの合計でこのノードのウェイトを割った値になります。たとえば、*1@1,3@2*はノード1に負荷の1/4が割り当てられ、ノード2に負荷の3/4が割り当てられることを指定します。

### Port\_list

サーバーが待機するポートです。このプロパティのデフォルト値は空の文字列です。ポートのリストはRTRファイルに指定できます。このファイルで指定しない場合、クラスタ管理者は、リソースを作成するときに、実際のポートのリストを提供する必要があります。

クラスタ管理者がスケーラブルかフェイルオーバーのどちらかとなるように構成することが可能な、データサービスを作成できます。このためには、データサービスのRTRファイルにおいて、FailoverリソースタイププロパティとScalableリソースプロパティの両方をFALSEに宣言します。Scalableプロパティは作成時に調整できるように指定します。

Failoverプロパティの値がFALSEの場合、リソースはスケーラブルリソースグループに構成できます。クラスタ管理者はリソースを作成するときにScalableの値をTRUEに変更し、スケーラブルサービスを作成することによって、共有アドレスを有効にできます。

一方、FailoverがFALSEに設定されている場合でも、クラスタ管理者はリソースをフェイルオーバーリソースグループに構成して、フェイルオーバーサービスを実装できます。クラスタ管理者はScalableの値(FALSE)は変更しません。このような状況に対処するために、ScalableプロパティのValidateメソッドで妥当性を検査する必要があります。ScalableがFALSEの場合、リソースがフェイルオーバーリソースグループに構成されていることを確認します。

スケーラブルリソースについては、『Sun Cluster の概念 (Solaris OS 版)』を参照してください。

## スケーラブルサービスの妥当性検査

Scalable プロパティが TRUE であるリソースを作成または更新するたびに、RGM は、さまざまなリソースプロパティの妥当性を検査します。プロパティの構成が正しく行われていないと、RGM は更新や作成の試行を拒否します。

RGM は次の検査を行います。

- `Network_resources_used` プロパティは、空であってはならず、既存の共有アドレスリソースの名前を含む必要があります。スケーラブルリソースを含むリソースグループの `NodeList` にあるすべてのノードは、指定した共有アドレスリソースのそれぞれの `NetIfList` プロパティまたは `AuxNodeList` プロパティに存在する必要があります。
- スケーラブルリソースを含むリソースグループの `RG_dependencies` プロパティは、スケーラブルリソースの `Network_resources_used` プロパティに存在する、すべての共有アドレスリソースのリソースグループを含む必要があります。
- `Port_list` プロパティは空であってはならず、ポートとプロトコルのペアのリストを含む必要があります。各ポート番号にはスラッシュ (/) を追加し、そのあとにはそのポートにより使用されているプロトコルを付けます。たとえば、次のように使用します。

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

プロトコルには、次のものを指定できます。

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

## データサービスの作成と検証

この節では、データサービスの作成と検証の方法について説明します。TCP キープアライブを使用したサーバーの保護、高可用性データサービスの検証、およびリソース間の依存関係の調節などについて説明します。

### TCP キープアライブを使用したサーバーの保護

サーバー側で TCP キープアライブを使用すると、サーバーはダウン時の(または、ネットワークで分割された)クライアントのシステムリソースを浪費しません。長時

間稼働するようなサーバーでこのようリソースがクリーンアップされない場合、クライアントがクラッシュと再起動を繰り返すことにより、最終的には浪費されるリソースは無制限に大きくなります。

クライアントサーバー通信がTCPストリームを使用する場合、クライアントとサーバーは両方ともTCPキープアライブ機構を有効にしなければなりません。これは、非高可用性の単一サーバーの場合でも適用されます。

ほかにも、キープアライブ機構を持っている接続指向のプロトコルは存在します。

クライアント側でTCPキープアライブを使用すると、ある物理ホストから別の物理ホストにネットワークアドレスリソースがフェイルオーバーまたはスイッチオーバーした場合、クライアントに通知することができます。このようなネットワークアドレスリソースの転送(フェイルオーバーやスイッチオーバー)が発生すると、TCP接続が切断されます。しかし、クライアント側でTCPキープアライブを有効にしておかなければ、接続が休止したとき、必ずしも接続の切断はクライアントに通知されません。

たとえば、クライアントが、実行に時間がかかる要求に対するサーバーからの応答を待っており、また、クライアントの要求メッセージがすでにサーバーに到着しており、TCP層で認識されているものと想定します。この状況では、クライアントのTCPモジュールは要求を再転送し続ける必要はありません。また、クライアントアプリケーションはブロックされて、要求に対する応答を待ちます。

クライアントアプリケーションは、可能であれば、TCPキープアライブ機構を使用するだけでなく、独自の定期的なキープアライブをアプリケーションレベルで実行する必要もあります。TCPキープアライブ機構は必ずしもあらゆる限界状況に対応できるわけではありません。アプリケーションレベルのキープアライブを使用するには、通常、クライアントサーバー型プロトコルがNULL操作、または、少なくとも効率的な読み取り専用操作(状態操作など)をサポートする必要があります。

## HA データサービスの検証

この節では、高可用性環境におけるデータサービスの実装を検証する方法について説明します。この検証は一例であり、完全ではないことに注意してください。実際に稼働させるマシンに影響を与えないように、検証時は、検証用のSun Cluster構成にアクセスする必要があります。

クラスタ内のすべてのノード上ではなく、単一ノード上の非大域ゾーン内で、HAデータサービスを検証します。データサービスが非大域ゾーン内で想定どおりに動作していると判断した場合は、次にクラスタ全体で検証を実行できます。ノード上の非大域ゾーン内で動作しているHAデータサービスは、正常に動作していない場合でも、ほかのゾーン内またはほかのノード上で動作しているデータサービスの動作を妨げることはないと考えられます。

リソースグループが物理ホスト間で移動する場合などすべてのケースで、HA データサービスが適切に動作するかを検証します。たとえば、システムがクラッシュした場合や、`clnode` コマンドを使用した場合です。また、このような場合にクライアントマシンがサービスを受け続けられるかどうかを検証します。

メソッドの呼び出し回数への非依存性を検証します。たとえば、各メソッドを一時的に、元のメソッドを2回以上呼び出す短いシェルスクリプトに変更します。

## リソース間の依存関係の調節

あるクライアントサーバーのデータサービスが、クライアントからの要求を満たすために、別のクライアントサーバーのデータサービスに要求を行うことがあります。たとえば、データサービス A がサービスを提供するために、データサービス B のサービスが必要な場合、データサービス A はデータサービス B に依存しています。この要件を満たすために、Sun Cluster では、リソースグループ内でリソースの依存関係を構築できます。依存関係は、Sun Cluster がデータサービスを起動および停止する順番に影響します。詳細は、`r_properties(5)` のマニュアルページを参照してください。

リソースタイプのリソースが別のタイプのリソースに依存する場合、リソースとリソースグループを適切に構成するようにクラスタ管理者に指示する必要があります。または、これらを正しく構成するスクリプトまたはツールを提供します。

明示的なリソースの依存関係を使用するか、このような依存関係を省略して、HA データサービスのコードで別のデータサービスの可用性をポーリングするかを決定します。依存するリソースと依存されるリソースが異なるノードまたはゾーン上で動作できる場合は、これらのリソースを異なるリソースグループ内で構成します。この場合、グループ間ではリソースの依存関係を構成できないため、ポーリングが必要です。

データサービスによっては、データを自分自身で直接格納しないものもあります。そのようなデータサービスは、代わりに、別のバックエンドデータサービスに依存して、すべてのデータを格納してもらいます。このようなデータサービスは、すべての読み取り要求と更新要求をバックエンドデータサービスへの呼び出しに変換します。たとえば、すべてのデータを SQL データベース (Oracle など) に格納するような仮定のクライアントサーバー型のアポイントメントカレンダーサービスを考えます。このサービスは独自のクライアントサーバー型ネットワークプロトコルを使用します。たとえば、RPC 仕様言語 (ONC RPC など) を使用するプロトコルを定義している場合があります。

Sun Cluster 環境では、HA-ORACLE を使用してバックエンド Oracle データベースの可用性を高めることができます。つまり、アポイントメントカレンダーデーモンを起動および停止する簡単なメソッドを作成できます。クラスタ管理者は Sun Cluster でアポイントメントカレンダーのリソースタイプを登録します。

HA-ORACLE リソースが、アポイントメントカレンダーリソースとは別のノードまたはゾーン上で動作する必要がある場合、クラスタ管理者はこれらのリソースを2つリソースグループ内に構成します。したがって、クラスタ管理者はアポイントメントカレンダーリソースを HA-ORACLE リソースに依存するようにします。

クラスタ管理者は次のいずれかを実行して、リソースを依存するようにします。

- HA-ORACLE リソースと同じリソースグループ内にアポイントメントカレンダーリソースを構成します。
- 各リソースが存在する2つのリソースグループ間で強いポジティブアフィニティを指定します。

このアフィニティは、`clresource` コマンドで `RG_affinities` プロパティを使用して指定します。

カレンダーデータサービスデーモンは、起動後、Oracle データベースが利用可能になるまで、ポーリングしながら待機します。この場合、通常、カレンダーリソースタイプの `Start` メソッドは成功を戻します。ただし、`Start` メソッドが無限にブロックされると、そのリソースグループがビジー状態に移行します。このビジー状態になると、それ以降、リソースグループで状態の変化(編集、フェイルオーバー、スイッチオーバーなど)が行われなくなります。カレンダーリソースの `Start` メソッドがタイムアウトするか非ゼロ状態で終了すると、Oracle データベースが利用できない間、タイムアウトまたは非ゼロ終了状態により、リソースグループが複数のノードまたはゾーン間でやりとりを無限に繰り返す可能性があります。



## Resource Management API リファレンス

---

この章では、Resource Management API (RMAPI) を構成するアクセス関数やコールバックメソッドについてリストし、簡単に説明します。詳細については、RMAPIのそれぞれのマニュアルページを参照してください。

この章の内容は次のとおりです。

- 65 ページの「RMAPI アクセスメソッド」 - シェルスクリプトコマンドと C 関数
- 71 ページの「RMAPI コールバックメソッド」 - `rt_callbacks(1HA)` のマニュアルページに説明されている内容

### RMAPI アクセスメソッド

RMAPI は、リソースタイプ、リソース、リソースグループのプロパティ、およびその他のクラスタ情報にアクセスするための関数を提供します。これらの関数はシェルコマンドと C 関数の両方の形で提供されるため、開発者はシェルスクリプトまたは C プログラムのどちらでも制御プログラムを実装できます。

### RMAPI シェルコマンド

シェルコマンドは、クラスタの RGM によって制御されるサービスを表すリソースタイプのコールバックメソッドを、シェルスクリプトで実装するときに使用します。

これらのコマンドを使用すると、次の作業を行えます。

- リソースタイプ、リソース、リソースグループ、クラスタについての情報にアクセスする。
- モニターと併用し、リソースの `Status` プロパティと `Status_msg` プロパティを設定する。
- リソースグループの再起動または再配置を要求する。

---

注-この節では、シェルコマンドについて簡単に説明します。詳細については、各コマンドの(1HA)マニュアルページを参照してください。特に注記しないかぎり、各コマンドと関連付けられた同じ名前のマニュアルページがあります。

---

## RMAPI リソースコマンド

以下のコマンドを使用すると、リソースについての情報にアクセスしたり、リソースの `Status` プロパティーや `Status_msg` プロパティーを設定できます。

### `scha_resource_get`

RGM の制御下のリソースまたはリソースタイプに関する情報にアクセスできます。このコマンドは、C 関数 `scha_resource_get()` と同じ情報を提供します。詳細は、`scha_resource_get(1HA)` のマニュアルページを参照してください。

### `scha_resource_setstatus`

RGM の制御下のリソースの `Status` および `Status_msg` プロパティーを設定します。このコマンドはリソースのモニターによって使用され、モニターから見たリソースの状態を示します。このコマンドは、C 関数 `scha_resource_setstatus()` と同じ機能を提供します。このコマンドについては、`scha_resource_setstatus(1HA)` のマニュアルページを参照してください。

---

注-`scha_resource_setstatus()` はリソースモニター専用の関数ですが、任意のプログラムから呼び出すことができます。

---

## リソースタイプコマンド

### `scha_resourcetype_get`

RGM に登録されているリソースタイプについての情報にアクセスします。このコマンドは、C 関数 `scha_resourcegroup_get()` と同じ機能を提供します。このコマンドについては、`scha_resourcetype_get(1HA)` のマニュアルページを参照してください。

## リソースグループコマンド

次に示すコマンドを使用すると、リソースグループについての情報にアクセスしたり、リソースグループを再起動したりすることができます。

### `scha_resourcegroup_get`

RGM の制御下のリソースグループに関する情報にアクセスできます。このコマンドは、C 関数 `scha_resourcegroup_get()` と同じ機能を提供します。このコマンドについては、`scha_resourcegroup_get(1HA)` のマニュアルページを参照してください。

### scha\_control

RGMの制御下のリソースグループの再起動、または別のノードまたはゾーンへの再配置を要求します。このコマンドは、C関数 `scha_control()` および `scha_control_zone()` と同じ機能を提供します。このコマンドについては、`scha_control(1HA)`のマニュアルページを参照してください。

## クラスタコマンド

### scha\_cluster\_get

クラスタについての情報(クラスタ名、ノードまたはゾーン名、ID、状態、およびリソースグループ)にアクセスします。このコマンドは、C関数 `scha_cluster_get()` と同じ情報を提供します。このコマンドについては、`scha_cluster_get(1HA)`のマニュアルページを参照してください。

## C関数

C関数は、クラスタのRGMによって制御されるサービスを表すリソースタイプのコールバックメソッドを、Cプログラムで実装するときに使用します。

これらの関数を使用すると、次の作業を行えます。

- リソースタイプ、リソース、リソースグループ、クラスタについての情報にアクセスする。
- リソースの `Status` および `Status_msg` プロパティを設定する。
- リソースグループの再起動または再配置を要求する。
- エラーコードを適切なエラーメッセージに変換する。

---

注-この節では、C関数について簡単に説明します。C関数の詳細については、各関数の(3HA)マニュアルページを参照してください。特に注記しないかぎり、各関数と関連付けられた同じ名前のマニュアルページがあります。C関数の出力引数および戻りコードについては、`scha_calls(3HA)`のマニュアルページを参照してください。

---

## リソース関数

以下の関数は、RGMに管理されているリソースについての情報にアクセスしたり、モニターから見たリソースの状態を示します。

### scha\_resource\_open(),scha\_resource\_get(),and\_scha\_resource\_close()

これらの関数は、RGMによって管理されるリソースに関する情報にアクセスします。`scha_resource_open()` 関数は、リソースへのアクセスを初期化し、`scha_resource_get` のハンドルを戻します。`scha_resource_get` 関数は、リソースの情報にアクセスします。`scha_resource_close()` 関数は、ハンドルを無効にし、`scha_resource_get` の戻り値に割り当てられているメモリーを解放します。

`scha_resource_open()` 関数がリソースのハンドルを戻したあとに、クラスタの再構成や管理アクションによって、リソースが変更されることがあります。その結果、`scha_resource_get()` 関数がハンドルを通じて獲得した情報は正しくない可能性があります。リソース上でクラスタの再構成や管理アクションが行われた場合、RGM は `scha_err_seqid` エラーコードを `scha_resource_get()` 関数に戻し、リソースに関する情報が変更された可能性があることを示します。このエラーメッセージは致命的ではありません。関数は正常に終了します。メッセージを無視し、戻された情報を受け入れることを選択できます。または、現在のハンドルを閉じて新しいハンドルを開き、リソースに関する情報にアクセスしてもかまいません。

これら3つの関数は1つのマニュアルページで説明しています。このマニュアルページには、個々の関数名 `scha_resource_open(3HA)`、`scha_resource_get(3HA)`、または `scha_resource_close(3HA)` でアクセスできます。

#### `scha_resource_setstatus()`

RGM の制御下のリソースの `Status` および `Status_msg` プロパティを設定します。この関数はリソースのモニターによって使用され、モニターから見たリソースの状態を反映します。

---

注 - `scha_resource_setstatus()` はリソースモニター専用の関数ですが、任意のプログラムから呼び出すことができます。

---

#### `scha_resource_setstatus_zone()`

`scha_resource_setstatus()` 関数と同様に、RGM の制御下のリソースの `Status` および `Status_msg` プロパティを設定します。この関数はリソースのモニターによって使用され、モニターから見たリソースの状態を反映します。ただし、この関数ではメソッドを実行するように構成されたゾーンの名前も指定します。

---

注 - `scha_resource_setstatus_zone()` は特にリソースモニターが使用しますが、任意のプログラムから呼び出すことができます。

---

## リソースタイプ関数

以下の関数は、RGM に登録されているリソースタイプに関する情報にアクセスします。

`scha_resourcetype_open()`、`scha_resourcetype_get()`、and `scha_resourcetype_close()`  
`scha_resourcetype_open()` 関数は、リソースへのアクセスを初期化し、`scha_resourcetype_get()` のハンドルを戻します。`scha_resourcetype_get()` 関数は、リソースタイプの情報にアクセスします。`scha_resourcetype_close()` 関数は、ハンドルを無効にし、`scha_resourcetype_get()` の戻り値に割り当てられているメモリーを解放します。

`scha_resourcetype_open()` 関数がリソースタイプのハンドルを戻したあとに、クラスタの再構成や管理アクションによって、リソースタイプが変更されることがあります。その結果、`scha_resourcetype_get()` 関数がハンドルを通じて獲得した情報は正しくない可能性があります。リソースタイプ上でクラスタの再構成や管理アクションが行われた場合、RGM は `scha_err_seqid` エラーコードを `scha_resourcetype_get()` 関数に戻し、リソースタイプに関する情報が変更された可能性があることを示します。このエラーメッセージは致命的ではありません。関数は正常に終了します。メッセージを無視し、戻された情報を受け入れることを選択できます。または、現在のハンドルを閉じて新しいハンドルを開き、リソースタイプに関する情報にアクセスしてもかまいません。

これら3つの関数は1つのマニュアルページで説明しています。このマニュアルページには、個々の関数名 `scha_resourcetype_open(3HA)`、`scha_resourcetype_get(3HA)`、または `scha_resourcetype_close(3HA)` でアクセスできます。

## リソースグループ関数

以下の関数を使用すると、リソースグループについての情報にアクセスしたり、リソースグループを再起動できます。

`scha_resourcegroup_open()`、`scha_resourcegroup_get()`、and  
`scha_resourcegroup_close()`

これらの関数は、RGM によって管理されるリソースグループに関する情報にアクセスします。`scha_resourcegroup_open()` 関数は、リソースグループへのアクセスを初期化し、`scha_resourcegroup_get()` のハンドルを戻します。

`scha_resourcegroup_get()` 関数は、リソースグループの情報にアクセスします。

`scha_resourcegroup_close()` 関数は、ハンドルを無効にし、

`scha_resourcegroup_get()` の戻り値に割り当てられているメモリーを解放します。

`scha_resourcegroup_open()` 関数がリソースグループのハンドルを戻したあとに、クラスタの再構成や管理アクションによって、リソースグループが変更されることがあります。その結果、`scha_resourcegroup_get()` 関数がハンドルを通じて獲得した情報は正しくない可能性があります。リソースグループ上でクラスタの再構成や管理アクションが行われた場合、RGM は `scha_err_seqid` エラーコードを `scha_resourcegroup_get()` 関数に戻し、リソースグループに関する情報が変更された可能性があることを示します。このエラーメッセージは致命的ではありません。関数は正常に終了します。メッセージを無視し、戻された情報を受け入れることを選択できます。または、現在のハンドルを閉じて新しいハンドルを開き、リソースグループに関する情報にアクセスしてもかまいません。

これら3つの関数は1つのマニュアルページで説明しています。このマニュアルページには、個々の関数名 `scha_resourcegroup_open(3HA)`、`scha_resourcegroup_get(3HA)`、および `scha_resourcegroup_close(3HA)` でアクセスできます。

### `scha_control()` and `scha_control_zone()`

RGPM の制御下のリソースグループの再起動、または別のノードまたはゾーンへの再配置を要求します。これらの関数については、`scha_control(3HA)` および `scha_control_zone(3HA)` のマニュアルページを参照してください。

## クラスタ関数

次に示す関数は、クラスタについての情報にアクセスし、その情報を戻します。

### `scha_cluster_open()`, `scha_cluster_get()`, and `scha_cluster_close()`

これらの関数は、クラスタについての情報(クラスタ名、ノードまたはゾーン名、ID、状態、およびリソースグループ)にアクセスします。

`scha_cluster_open()` 関数がクラスタのハンドルを戻したあとに、再構成や管理アクションによって、クラスタが変更されることがあります。その結果、`scha_cluster_get()` 関数がハンドルを通じて獲得した情報は正しくない可能性があります。クラスタ上で再構成や管理アクションが行われた場合、RGPM は `scha_err_seqid` エラーコードを `scha_cluster_get` 関数に戻し、クラスタに関する情報が変更された可能性があることを示します。このエラーメッセージは致命的ではありません。関数は正常に終了します。メッセージを無視し、戻された情報を受け入れることを選択できます。または、現在のハンドルを閉じて新しいハンドルを開き、クラスタに関する情報にアクセスしてもかまいません。

`Global_zone` リソースタイププロパティが `TRUE` に等しいリソースが、`ZONE_LOCAL` の問い合わせの `optag` の値を指定して `scha_cluster_get()` を起動した場合、大域ゾーンの名前が返されます。この場合、呼び出した側のコードでは文字列 `:zonename` をローカルノード名に連結して、リソースが実際に構成されているゾーンを取得する必要があります。`zonename` は、`-Z zonename` コマンド行オプション内のメソッドに渡されるものと同じゾーン名です。コマンド行内に `-Z` オプションがない場合は、リソースグループが大域ゾーン内に構成されるので、ゾーン名をノード名に連結する必要はありません。

これら3つの関数は1つのマニュアルページで説明しています。このマニュアルページには、個々の関数名 `scha_cluster_open(3HA)`、`scha_cluster_get(3HA)`、および `scha_cluster_close(3HA)` でアクセスできます。

### `scha_cluster_getlogfacility()`

クラスタログとして使用されるシステムログ機能の数を戻します。戻された番号を Solaris の `syslog()` 関数で使用すると、イベントと状態メッセージをクラスタログに記録できます。この関数については、`scha_cluster_getlogfacility(3HA)` のマニュアルページを参照してください。

### `scha_cluster_getnodename()`

関数が呼び出されたクラスタノードの名前を戻します。この関数については、`scha_cluster_getnodename(3HA)` のマニュアルページを参照してください。

## ユーティリティー関数

この関数は、エラーコードをエラーメッセージに変換します。

`scha_strerror()`

`scha_` 関数のいずれかによって戻されるエラーコードを、対応するエラーメッセージに変換します。この関数を `logger` コマンドと共に使用すると、メッセージを Solaris システムログ (`syslog`) に記録できます。この関数については、`scha_strerror(3HA)` のマニュアルページを参照してください。

## RMAPI コールバックメソッド

コールバックメソッドは、リソースタイプを実装するために API が提供する重要な要素です。コールバックメソッドを使用すると、RGM は、クラスタのメンバーシップが変更されたとき(ノードあるいはゾーンが起動またはクラッシュしたとき)にクラスタ内のリソースを制御できます。

---

注-クライアントプログラムがクラスタシステムの HA サービスを制御するため、コールバックメソッドはスーパーユーザーまたは最大の RBAC ロールのアクセス権を持つ RGM によって実行されます。したがって、このようなコールバックメソッドをインストールおよび管理するときは、ファイルの所有権とアクセス権を制限します。特に、このようなメソッドには、特権付き所有者 (`bin` や `root` など) を割り当てます。また、このようなメソッドは、書き込み可能にはなりません。

---

この節では、コールバックメソッドの引数と終了コードについて説明します。

次のカテゴリのコールバックメソッドについて説明します。

- 制御および初期化メソッド
- 管理サポートメソッド
- ネットワーク関連メソッド
- モニター制御メソッド

---

注-この節では、メソッドが実行される時点や予想されるリソースへの影響など、コールバックメソッドについて簡単に説明します。コールバックメソッドについては、`rt_callbacks(1HA)` のマニュアルページを参照してください。

---

## コールバックメソッドに提供できる引数

RGM は、次のようにコールバックメソッドを実行します。

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

*method* は、Start や Stop などのコールバックメソッドとして登録されているプログラムのパス名です。リソースタイプのコールバックメソッドは、それらの登録ファイルで宣言します。

コールバックメソッドの引数はすべて、次のようにフラグ付きの値として渡されません。

- -R はリソースインスタンスの名前を示します。
- -T はリソースのタイプを示します。
- -G はリソースが構成されているグループを示します。

このような引数をアクセス関数で使用すると、リソースについての情報を取得できません。

Validate メソッドを呼び出すときは、追加の引数(リソースのプロパティ値と呼び出しが行われるリソースグループ)を使用します。

詳細は、`scha_calls(3HA)` のマニュアルページを参照してください。

## コールバックメソッドの終了コード

すべてのコールバックメソッドは、同じ終了コードを持っています。これらの終了コードは、メソッドの呼び出しによるリソースの状態への影響を示すように定義されています。これらの終了コードについては、`scha_calls(3HA)` のマニュアルページを参照してください。

終了コードには、次の主要な2つのカテゴリがあります。

- 0 - メソッドは成功しました。
- ゼロ以外の任意の値 - メソッドは失敗しました。

RGM は、タイムアウトやコアダンプなど、コールバックメソッドの実行の異常終了も処理します。

メソッドは、各ノードまたはゾーン上で `syslog()` を使用して障害情報を出力するように実装する必要があります。 `stdout` や `stderr` に書き込まれる出力は、ローカルノードのコンソール上には表示されますが、ユーザーに伝達される保証はありません。

## 制御および初期化コールバックメソッド

制御および初期化コールバックメソッドは、主に、リソースを起動および停止します。その他のメソッドは、リソース上で初期化と終了コードを実行します。

## Start

リソースを含むリソースグループがクラスタノードまたはゾーン上でオンラインになったとき、RGMはそのクラスタノードまたはゾーン上でこのメソッドを実行します。このメソッドは、そのノードまたはゾーン上でリソースを起動します。

ローカルノードまたはゾーン上でリソースが起動され、利用可能になるまで、Start メソッドは終了してはなりません。したがって、Start メソッドは終了する前にリソースをポーリングし、リソースが起動しているかどうかを判断する必要があります。さらに、このメソッドには、十分な長さのタイムアウト値を設定する必要があります。たとえば、データベースデーモンなど特定のリソースが起動するのに時間がかかる場合、そのメソッドには十分な長さのタイムアウト値が必要です。

RGMがStart メソッドの失敗に応答する方法は、Failover\_mode プロパティの設定によって異なります。

リソースタイプ登録 (RTR) ファイルの Start\_timeout プロパティが、リソースの Start メソッドのタイムアウト値を設定します。

## Stop

リソースを含むリソースグループがクラスタノードまたはゾーン上でオフラインになったとき、RGMはクラスタノードまたはゾーン上でこの必須メソッドを実行します。このメソッドは、リソースを(アクティブであれば)停止します。

ローカルノードまたはゾーン上でリソースがすべての活動を完全に停止し、すべてのファイル記述子を閉じるまで、Stop メソッドは終了してはなりません。そうしないと、RGMが(実際にはアクティブであるのに)リソースが停止したと判断するため、データが破壊されることがあります。データの破壊を防ぐために最も安全な方法は、リソースに関連するローカルノードまたはゾーン上ですべてのプロセスを停止することです。

Stop メソッドは終了する前にリソースをポーリングし、リソースが停止しているかどうかを判断する必要があります。さらに、このメソッドには、十分な長さのタイムアウト値を設定する必要があります。たとえば、特定のリソース(データベースデーモンなど)が停止するのに時間がかかる場合、そのメソッドには十分な長さのタイムアウト値が必要です。

RGMメソッドコールバックがタイムアウトすると、メソッドのプロセスツリーが、SIGTERMシグナルではなく、SIGABRTシグナルによって消去されます。結果として、プロセスグループのすべてのメンバーが、/var/cluster/coreディレクトリにコアダンプファイルを生成します。このコアダンプファイルは、メソッドがタイムアウトを超過した理由を判定できるように生成されます。

---

注- 新しいプロセスグループを作成するデータサービスメソッドを書かないでください。データサービスメソッドで新しいプロセスグループを作成する必要がある場合は、SIGTERM および SIGABRT シグナルのシグナルハンドラを書きます。また、シグナルハンドラは、プロセスを終了する前に、単数または複数の子プロセスグループに SIGTERM または SIGABRT シグナルを転送する必要があります。これらのシグナルのシグナルハンドラを書くと、使用するメソッドによって生成されるすべてのプロセスが、正しく終了される可能性が高まります。

---

RGM が Stop メソッドの失敗に応答する方法は、Failover\_mode プロパティの設定によって異なります。264 ページの「リソースのプロパティ」を参照してください。

RTR ファイルの Stop\_timeout プロパティが、リソースの Stop メソッドのタイムアウト値を設定します。

#### Init

リソースを管理下に置くとき、RGM はこのオプションメソッドを実行して、リソースの初期化を 1 回だけ実行します。リソースグループが管理されていない状態から管理されている状態に切り替えられるとき、またはすでに管理されているリソースグループでリソースが作成されるとき、RGM はこのメソッドを実行します。このメソッドは、Init\_nodes リソースプロパティにより特定されるノードまたはゾーン上で呼び出されます。

#### Fini

リソースが RGM によって管理されなくなったとき、RGM は Fini メソッドを実行して、そのリソースの使用後のクリーンアップを行います。Fini メソッドは、通常、Init メソッドにより実行された初期化を元に戻します。

次のような状態が生じた場合にリソースが管理対象外となる各ノードまたはゾーンで、RGM は Fini を実行します。

- リソースを含むリソースグループが管理対象外状態に切り替わる。この場合、ノードリストのすべてのノードおよびゾーン上で、RGM は Fini メソッドを実行します。
- 管理されているリソースグループからリソースが削除される。この場合、ノードリストのすべてのノードおよびゾーン上で、RGM は Fini メソッドを実行します。
- リソースを含むリソースグループのノードリストからノードまたはゾーンが削除される。この場合、削除されたノードまたはゾーン上でのみ、RGM は Fini メソッドを実行します。

「ノードリスト」はリソースグループの NodeList またはリソースタイプの Installed\_nodes リストのいずれかです。「ノードリスト」がリソースグループの NodeList とリソースタイプの Installed\_nodes リストのどちらを指すかは、リソー

スタイブの `Init_nodes` プロパティの設定に依存します。`Init_nodes` プロパティは `RG_nodelist` または `RT_installed_nodes` に設定できます。大部分のリソースタイプでは、`Init_nodes` はデフォルトである `RG_nodelist` に設定されます。この場合、`Init` メソッドと `Fini` メソッドは両方とも、リソースグループの `Nodelist` で指定されているノードおよびゾーン上で実行されます。

`Init` メソッドが実行する初期化の種類は、次のように、ユーザーが実装した `Fini` メソッドが実行する必要があるクリーンアップの種類を定義します。

- ノード固有の構成のクリーンアップ。
- クラスタ全体の構成のクリーンアップ。

実装する `Fini` メソッドは、ノード固有の構成だけをクリーンアップするのか、それともノード固有の構成とクラスタ全体にわたる構成の両方をクリーンアップするのかを判断する必要があります。

リソースが特定のノードまたはゾーン上でのみ、管理されなくなった場合、`Fini` メソッドはノード固有のローカル構成をクリーンアップできます。しかし、ほかのノード上ではリソースは引き続き管理されているため、`Fini` メソッドはクラスタ全体にわたるグローバル構成をクリーンアップしてはいけません。リソースがクラスタ全体にわたって管理されなくなった場合には、`Fini` メソッドはノード固有の構成とグローバル構成の両方についてクリーンアップを実行できます。実装する `Fini` メソッドのコードは、`Fini` メソッドを実行するローカルのノードまたはゾーンがリソースグループのノードリストに含まれているかどうかを調べることによって、これら2つの場合を区別できます。

ローカルのノードまたはゾーンがリソースグループのノードリストに出現している場合は、リソースが削除されようとしているか、管理されない状態に移行しようとしています。リソースはどのノードまたはゾーン上でもアクティブでなくなっています。この場合、実装する `Fini` メソッドでは、ローカルノード上のノード固有の構成だけでなく、クラスタ全体にわたる構成についてもクリーンアップする必要があります。

ローカルのノードまたはゾーンがリソースグループのノードリストに出現していない場合は、`Fini` メソッドでそのローカルのノードまたはゾーン上のノード固有の構成をクリーンアップできます。しかし、`Fini` メソッドでクラスタ全体にわたる構成をクリーンアップしてはなりません。この場合、ほかのノードまたはゾーン上でリソースが引き続きアクティブになっています。

`Fini` メソッドは順序に依存しない様子をコードを作成する必要もあります。つまり、`Fini` メソッドが以前の実行でリソースをクリーンアップした場合でも、以降の `Fini` 呼び出しは正常に終了します。

#### Boot

RGMは `Init` とよく似たこのオプションメソッドを実行し、リソースの所属リソースグループがRGMの管理下に置かれたあと、クラスタを結合するノードまたはゾーン上のリソースを初期化します。`Init_nodes` リソースプロパティにより

特定されるノードまたはゾーン上で、RGMはこのメソッドを実行します。起動または再起動の結果としてノードまたはゾーンがクラスタに結合または再結合したときに、Boot メソッドは呼び出されます。

Global\_zone リソースタイププロパティが TRUE に等しい場合、リソースを含むリソースグループが非大域ゾーンで動作するように構成されているときでも、メソッドは大域ゾーンで実行されます。

---

注-Init、Fini、またはBoot メソッドが失敗した場合は、エラーメッセージがシステムログに書き込まれます。ただし、それ以外はRGMによるリソース管理に影響しません。

---

## 管理サポートメソッド

リソース上での管理アクションには、リソースプロパティの設定と変更があります。Validate および Update コールバックメソッドを使用してリソースタイプを実装すると、このような管理アクションを実行できます。

### Validate

リソースの作成時や、クラスタ管理者によるリソースまたはリソースグループのプロパティの更新時、RGMは、このオプションメソッドを呼び出します。このメソッドは、リソースタイプの Init\_nodes プロパティにより特定されるクラスタノードまたはゾーンのセットに対して呼び出されます。Validate メソッドは作成または更新が行われる前に呼び出されます。任意のノードまたはゾーン上でメソッドから失敗の終了コードが戻ってくると、作成または更新は取り消されません。

Validate は、クラスタ管理者によってリソースプロパティまたはリソースグループプロパティが変更されたときだけ呼び出されます。RGMによってプロパティが設定されたときや、モニターによって Status および Status\_msg リソースプロパティが設定されたときは、このメソッドは呼び出されません。

### Update

RGMは、このオプションメソッドを実行して、プロパティが変更されたことを実行中のリソースに通知します。管理アクションがリソースまたはそのグループのプロパティの設定に成功したあとに、RGMは Update を実行します。このメソッドは、リソースがオンラインであるノードまたはゾーン上で呼び出されます。このメソッドは、API アクセス関数を使用し、アクティブなリソースに影響する可能性があるプロパティ値を読み取り、その値に従って実行中のリソースを調節します。

---

注-Update メソッドが失敗した場合は、エラーメッセージがシステムログに書き込まれます。ただし、それ以外はRGMによるリソース管理に影響しません。

---

## ネットワーク関連コールバックメソッド

ネットワークアドレスリソースを使用するサービスでは、ネットワークアドレス構成から始まる特定の順番で、起動手順または停止手順を実行する必要があります。任意のコールバックメソッドの `Pre-net_start` と `Post-net_stop` を使用してリソースタイプを実装すると、関連するネットワークアドレスが「起動」に構成される前、または、「停止」に構成されたあとに、特別な起動アクションとシャットダウンアクションを実行できます。

### `Pre-net_start`

このオプションメソッドを呼び出して、同じリソースグループ内のネットワークアドレスが構成される前に特殊な起動アクションを実行することができます。

### `Post-net_stop`

このオプションメソッドを呼び出して、同じリソースグループ内のネットワークアドレスを停止状態に構成したあとに特殊な終了アクションを実行することができます。

## モニター制御コールバックメソッド

リソースタイプの実装は、オプションとして、リソースの性能を監視したり、その状態を報告したり、リソースの障害に対処するようなプログラムを含むことができます。`Monitor_start`、`Monitor_stop`、`Monitor_check` メソッドは、リソースタイプへのリソースモニターの実装をサポートします。

### `Monitor_start`

このオプションメソッドを呼び出して、リソースの起動後にリソースの監視を開始することができます。

### `Monitor_stop`

この任意メソッドは、リソースが停止する前に呼び出され、リソースのモニターを停止します。

### `Monitor_check`

このオプションメソッドを呼び出して、リソースグループをノードまたはゾーンに再配置する前に、そのノードまたはゾーンの信頼性を査定することができます。`Monitor_check` メソッドは、並行して実行中のそのほかのメソッドと競合しない方法で実装する必要があります。



# ◆◆◆ 第 4 章

## リソースタイプの変更

---

この章では、リソースタイプを変更するために理解しておく必要がある問題を説明します。また、クラスタ管理者がリソースを更新できるようにする手段についても説明します。

この章の内容は次のとおりです。

- 79 ページの「リソースタイプの変更の概要」
- 80 ページの「リソースタイプ登録ファイルの内容の設定」
- 84 ページの「クラスタ管理者がアップグレードする際の処理」
- 84 ページの「リソースタイプモニターコードの実装」
- 85 ページの「インストール要件とパッケージの決定」
- 88 ページの「変更されたリソースタイプに提供すべき文書」

### リソースタイプの変更の概要

クラスタ管理者は、次の作業を実行できる必要があります。

- 既存のリソースタイプの新しいバージョンをインストールおよび登録する
- 特定のリソースタイプの複数のバージョンの登録を許可する
- リソースを削除し再作成することなく、既存のリソースを新しいバージョンのリソースタイプにアップグレードする

ユーザーがアップグレードしようとするリソースタイプは「アップグレード対応」リソースタイプと呼ばれます。

ユーザーが変更する既存のリソースタイプの要素には次のものがあります。

- リソースタイププロパティの属性
- 標準プロパティ、拡張プロパティを含む宣言済みリソースプロパティセット

- default、min、max、arraymin、arraymax、tunability などのリソースプロパティの属性
- 宣言済みメソッドのセット
- メソッドやモニターの実装

---

注-リソースタイプ開発者は、アプリケーションコードを変更する際に必ずしもリソースタイプを変更する必要はありません。

---

リソースタイプ開発者は、クラスタ管理者がリソースタイプをアップグレードできるようにするツールを提供するための要件を理解する必要があります。この章では、これらのツールを設定するために知っておく必要がある事項について説明します。

## リソースタイプ登録ファイルの内容の設定

ここでは、リソースタイプ登録ファイルの設定方法について説明します。

この章の内容は次のとおりです。

- 80 ページの「リソースタイプ名」
- 81 ページの「`#$upgrade` および `#$upgrade_from` ディレクティブの指定」
- 83 ページの「RTR ファイルでの `RT_version` の変更」
- 83 ページの「以前のバージョンの Sun Cluster のリソースタイプ名」

### リソースタイプ名

リソースタイプ名の3つのコンポーネントは、*vendor-id*、*resource-type*、*rt-version* として、RTR ファイルで指定されているプロパティです。clresourcetype(1CL) コマンドは、ピリオドとコロンの区切り文字を挿入して次のリソースタイプの名前を作成します。

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

*vendor-id* 接頭辞は、異なる会社が提供する同じ名前の2つの登録ファイルを区別する役目を果たします。*vendor-id* が一意であることを保証するためには、リソースタイプを作成した時点の会社の株式の略号を使用します。*rt-version* は、同じベースリソースタイプの複数の登録バージョン(アップグレード)を識別します。

次のコマンドを入力することで、完全修飾リソースタイプ名を取得できます。

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

Sun Cluster 3.1 以前に登録されたリソースタイプ名は、引き続き次の構文を使用します。

*vendor-id.resource-type*

リソースタイプ名の形式は、360 ページの「リソースタイプ名の形式」で説明されています。

## #\$upgrade および #\$upgrade\_from ディレクティブの指定

変更するリソースタイプがアップグレード対応であるようにするには、リソースタイプの RTR ファイルに #\$upgrade ディレクティブを含めます。#\$upgrade ディレクティブのあと、サポートするリソースタイプの各旧バージョンに対して 0 個以上の #\$upgrade\_from ディレクティブを追加します。

#\$upgrade および #\$upgrade\_from ディレクティブは、RTR ファイルのリソースタイププロパティー宣言と、リソース宣言のセクションの間に存在する必要があります。詳細は、rt\_reg(4) のマニュアルページを参照してください。

例 4-1 RTR ファイルの #\$upgrade\_from ディレクティブ

```
#$upgrade_from "1.1" WHEN_OFFLINE
#$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
```

#\$upgrade\_from ディレクティブの形式は次のとおりです。

*#\$upgrade\_from version tunability*

*version*

RT\_version。リソースタイプにバージョンがない場合、または以前に RTR ファイルで定義したバージョン以外のバージョンに対しては、空の文字列(“”)を指定します。

*tunability*

クラスタ管理者が指定の RT\_version をアップグレードできる条件または時点。

#\$upgrade\_from ディレクティブでは次の Tunable 属性の値を使用します。

#### ANYTIME

クラスタ管理者がリソースをアップグレードできる時点に対して制限がない場合に使用します。リソースは、アップグレード中完全にオンラインになることができます。

#### WHEN\_UNMONITORED

新しいリソースタイプバージョンのメソッドが次のような場合に使用します。

- Update、Stop、Monitor\_check、Postnet\_stop メソッドが、古いリソースタイプバージョンの起動メソッド (Prenet\_stop および Start) と互換性がある
- Fini メソッドが、古いバージョンの Init メソッドと互換性がある

クラスタ管理者は、アップグレードの前にリソース監視プログラムのみを停止する必要があります。

#### WHEN\_OFFLINE

新しいリソースタイプバージョンの Update、Stop、Monitor\_check、Postnet\_stop メソッドが次のような場合に使用します。

- 古いバージョンの Init メソッドと互換性がある
- 古いリソースタイプバージョンの起動メソッド (Prenet\_stop および Start) と互換性がない

クラスタ管理者は、アップグレードの前にリソースをオフラインにする必要があります。

#### WHEN\_DISABLED

WHEN\_OFFLINE と同様です。ただし、クラスタ管理者はアップグレードの前にリソースを無効にする必要があります。

#### WHEN\_UNMANAGED

新しいリソースタイプバージョンの Fini メソッドが、古いバージョンの Init メソッドと互換性がない場合に使用します。クラスタ管理者はアップグレードの前に、既存のリソースグループを管理されていない状態に切り替える必要があります。

リソースタイプのバージョンが #supgrade\_from ディレクティブのリストに存在しない場合、RGMにより WHEN\_UNMANAGED の Tunable 属性はデフォルトでそのバージョンにされます。

#### AT\_CREATION

既存のリソースが、新しいバージョンのリソースタイプにアップグレードされるのを防ぐために使用します。クラスタ管理者はリソースを削除し、再作成する必要があります。

## RTR ファイルでの RT\_version の変更

RTR ファイルの内容が変更されても、そのたびに RTR ファイルの RT\_version プロパティを変更するだけで済みます。このバージョンのリソースタイプが最新バージョンであることを明確に示す、このプロパティの値を選択します。

RTR ファイルの RT\_version 文字列には次の文字を含めないでください。次の文字を含めると、リソースタイプの登録が失敗します。

- スペース
- タブ
- スラッシュ (/)
- 逆スラッシュ (\)
- アスタリスク (\*)
- 疑問符 (?)
- コンマ (,)
- セミコロン (;)
- 左角括弧 (l)
- 右角括弧 (l)

RT\_version プロパティは、Sun Cluster 3.0 まではオプションですが、Sun Cluster 3.1 以降のリリースでは必須です。

## 以前のバージョンの Sun Cluster のリソースタイプ名

次に示すように、Sun Cluster 3.0 のリソースタイプ名には、バージョン接尾辞がありません。

*vendor-id.resource-type*

Sun Cluster 3.0 で登録したリソースタイプの名前については、Sun Cluster 3.1 および Sun Cluster 3.2 でもこの構文が保たれます。RTR ファイルを、`#$upgrade` が省略された Sun Cluster 3.1 または Sun Cluster 3.2 で登録した場合でも、リソースタイプ名はこの構文に従います。

クラスタ管理者は、Sun Cluster 3.0 では、`#$upgrade` ディレクティブや `#$upgrade_from` ディレクティブを使った RTR ファイルの登録は可能ですが、Sun Cluster 3.0 では、既存のリソースの新しいリソースタイプへのアップグレードはサポートされません。

## クラスタ管理者がアップグレードする際の処理

リソースタイプをアップグレードする時点でクラスタ管理者が行わなければならない処理、およびシステムにより行われる処理を次に示します。

- 既存のリソースプロパティ属性が新しいリソースタイプのバージョンの妥当性検査の条件を満たしていない場合、クラスタ管理者は有効な値を指定する必要があります。

クラスタ管理者は、次の条件のもとで有効な値を提供する必要があります。

- リソースタイプの新しいバージョンが、以前のバージョンでは宣言されていなかったプロパティを使用し、デフォルト値がない場合。
- 既存のリソースが、新しいバージョンでは値が宣言されていないか無効であるプロパティを使用している場合。リソースタイプの新しいバージョンでは宣言されていない宣言済みプロパティは、リソースから削除されます。
- サポートされていないバージョンのリソースタイプからアップグレードを試みると失敗します。
- アップグレード後、リソースは、新しいバージョンのリソースタイプから、すべてのプロパティのリソースプロパティ属性を継承します。
- RTR ファイルでリソースタイプのデフォルト値を変更すると、既存のリソースにより新しいデフォルト値が継承されます。プロパティが `AT_CREATION` または `WHEN_DISABLED` のみで `tunable` に宣言されている場合であっても、新しいデフォルト値は継承されます。クラスタ管理者が作成する同じタイプのプロパティも、このデフォルト値を継承します。ただし、クラスタ管理者がプロパティに新しいデフォルト値を指定する場合、RTR ファイルで指定されているデフォルト値よりも、新しいデフォルト値が優先されます。

---

注 - Sun Cluster 3.0 で作成されたリソースは、それ以降のバージョンの Sun Cluster にアップグレードされたときに、リソースタイプから新しいデフォルトリソースプロパティ属性を継承しません。この制限は、Sun Cluster 3.0 クラスタからアップグレードされた Sun Cluster 3.1 クラスタのみに適用されます。クラスタ管理者は、プロパティに値を指定し、デフォルトよりも優先させることによって、この制限に対処できます。

---

## リソースタイプモニターコードの実装

クラスタ管理者は Sun Cluster 3.0 のアップグレード対応のリソースタイプを登録できます。ただし、Sun Cluster ではバージョン接尾辞の付かないリソースタイプ名が記録されます。このリソースタイプのモニターコードを Sun Cluster 3.0 および Sun Cluster 3.1 で正しく実行するためには、そのモニターコードで次の命名規則の両方を処理できるようにする必要があります。

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

リソースタイプ名の形式は、[360 ページ](#)の「リソースタイプ名の形式」で説明されています。

クラスタ管理者は、2つの異なる名前の下で、同じバージョンのリソースタイプを2回登録することはできません。モニターコードが正しい名前を判断できるようにするには、モニターコードで次のコマンドを呼び出します。

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

続いて、出力値と `vers` を比較します。`vers` の特定の値に対して、これらのコマンドのいずれか1つのみが成功します。

## インストール要件とパッケージの決定

リソースタイプパッケージのインストール要件とパッケージを決定するには、次の2つの要件を考慮します。

- 新しいリソースタイプが登録されている場合、ディスク上のRTRファイルにアクセスできなければなりません。
- 新しいタイプのリソースを作成した場合、新しいタイプのすべての宣言済みメソッドのパス名および監視プログラムがディスク上に存在し、実行可能でなければなりません。リソースが使用されている間は、以前のメソッドおよび監視プログラムを定位置に確保しておく必要があります。

使用すべき適切なパッケージを決定するには、次の点を考慮する必要があります。

- RTRファイルが変更されたか
- プロパティのデフォルト値または `tunable` 属性が変更されたか
- プロパティの `min` または `max` 値が変更されたか
- アップグレードによってプロパティが追加されたか、または削除されたか
- モニターコードが変更されたか
- メソッドコードが変更されたか
- 新しいメソッド、モニターコード、またはその両方が以前のバージョンと互換性があるか

これらの点を確認しておくこと、新しいリソースタイプに使用する適切なパッケージの決定に役立ちます。

## RTR ファイルを変更する前に

リソースタイプを変更する場合、必ずしも新しいメソッドやモニターコードを作成する必要はありません。たとえば、リソースプロパティのデフォルト値や Tunable 属性のみを変更する場合があります。この場合、メソッドコードを変更していないため、読み取り可能な RTR ファイルへの新しい有効なパス名のみが必要になります。

古いリソースタイプを再登録する必要がない場合、新しい RTR ファイルは以前のバージョンを上書きできます。再登録する必要がある場合、新しいパスに新しい RTR ファイルを配置します。

アップグレードによりプロパティのデフォルト値または Tunable 属性が変更された場合、リソースタイプの新しいバージョンに対して Validate メソッドを使用し、既存のプロパティ属性が新しいリソースタイプに対して有効であることを確認します。有効でない場合、クラスタ管理者は既存のリソースのプロパティを正しい値に変更できます。アップグレードによりプロパティの min、max、または type 属性が変更された場合、クラスタ管理者がリソースタイプをアップグレードするときに、`clresourcetype(1CL)` コマンドによってこれらの制約の有効性が自動的に確認されます。

アップグレードにより新しいプロパティが追加された場合や古いプロパティが削除された場合、通常、コールバックメソッドまたはモニターコードを変更する必要があります。

## モニターコードの変更

リソースタイプのモニターコードのみを変更した場合、パッケージのインストールではモニターのパイナリを上書きできます。

## メソッドコードの変更

リソースタイプでメソッドコードのみを変更した場合、新しいメソッドコードが古いメソッドコードと互換性があるかどうかを判断する必要があります。この判断により、新しいメソッドコードを新しいパス名で格納する必要があるかどうか、または古いメソッドを上書きできるかどうかが決まります。

古いバージョンの Start、Prenet\_stop、Init メソッドにより初期化または起動されたリソースに対して、新しい Stop、Postnet\_stop、Fini メソッド (宣言されている場合) を適用できる場合は、新しいメソッドで古いメソッドを上書きできます。

プロパティに新しいデフォルト値を適用することで、Stop、Postnet\_stop、Fini などのメソッドが失敗する場合、リソースタイプのアップグレード時に、クラスタ管理者はそれに従ってリソースの状態を制限する必要があります。

Type\_version プロパティの Tunable 属性を制限することにより、クラスタ管理者が、アップグレード時のリソースの状態を制限できるようにすることができます。

パッケージの1つの方法としては、引き続きサポートされている以前のバージョンのリソースタイプをすべてパッケージに含めるという方法もあります。この方法では、メソッドへの古いパスを上書きまたは削除することなく、新しいパッケージのバージョンで古いバージョンのパッケージを置き換えることができます。サポートする以前のバージョンの数は、リソースタイプ開発者が決定する必要があります。

## 使用するパッケージスキーマの決定

次の表に、新しいリソースタイプに使用すべきパッケージスキーマの概要を示します。

表4-1 使用するパッケージスキーマの決定

変更のタイプ	Tunable属性の値	パッケージスキーマ
RTR ファイルのみでプロパティを変更します。	ANYTIME	新しい RTR ファイルのみを提供します。
メソッドを更新します。	ANYTIME	古いメソッドとは異なるパスに、更新されたメソッドを配置します。
新しい監視プログラムをインストールします。	WHEN_UNMONITORED	モニターの直前のバージョンのみを上書きします。
メソッドを更新します。 新しい Update および Stop メソッドと古い Start メソッドの間には互換性はありません。	WHEN_OFFLINE	古いメソッドとは異なるパスに、更新されたメソッドを配置します。
メソッドを更新し、RTR ファイルに新しいプロパティを追加します。新しいメソッドには新しいプロパティが必要です。  目的は、ノードまたはゾーン上でリソースグループがオフライン状態からオンライン状態に移行した場合に、リソースの所属リソースグループをオンラインのまま保持しながらリソースがオンラインになるのを防ぐことです。	WHEN_DISABLED	以前のバージョンのメソッドを上書きします。
メソッドを更新し、RTR ファイルに新しいプロパティを追加します。新しいメソッドは新しいプロパティを必要としません。	ANYTIME	以前のバージョンのメソッドを上書きします。

表 4-1 使用するパッケージスキーマの決定 (続き)

変更のタイプ	Tunable 属性の値	パッケージスキーマ
メソッドを更新します。新しい Fini メソッドと古い Init メソッドには互換性がありません。	WHEN_UNMANAGED	古いメソッドとは異なるパスに、更新されたメソッドを配置します。
メソッドを更新します。RTR ファイルは変更されていません。	該当しない。RTR ファイルは変更されていません。	以前のバージョンのメソッドを上書きしません。RTR ファイルには変更を加えていないため、リソースを登録またはアップグレードする必要はありません。

## 変更されたリソースタイプに提供すべき文書

『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』の「リソースタイプの更新」では、クラスタ管理者に対するリソースタイプのアップグレード方法が説明されています。変更されるリソースタイプをクラスタ管理者がアップグレードできるようにするには、上記の手順に、この節で説明する追加情報を補足します。

通常、新しいリソースタイプを作成する場合、次の内容を含む文書を提供する必要があります。

- 追加、変更、または削除するプロパティを説明する
- プロパティを新しい要件に準拠させる方法を説明する
- リソースに対する Tunable 属性の制約を記載する
- 新しいデフォルトプロパティ属性を述べる
- 必要に応じて、既存のリソースプロパティを適切な値に設定できることをクラスタ管理者に通知する

## アップグレードのインストール前に実行すべき事柄に関する情報

次のように、ノード上でのアップグレードパッケージのインストール前に実行すべき事柄を、クラスタ管理者に説明します。

- アップグレードパッケージが既存のメソッドを上書きする場合、非クラスタモードでノードを再起動するようクラスタ管理者に指示します。
- アップグレードパッケージはモニターコードのみを更新し、メソッドコードを変更しない場合は、ノードをクラスタモードで実行し続けるようクラスタ管理者に通知します。また、すべてのリソースタイプの監視をオフにするようクラスタ管理者に通知します。

- アップグレードパッケージはRTRファイルのみを更新し、モニターコードを変更しない場合は、ノードをクラスタモードで実行し続けるようクラスタ管理者に通知します。また、すべてのリソースタイプの監視をオンのままにするようクラスタ管理者に通知します。

## リソースをアップグレードする時点に関する情報

リソースを新しいバージョンのリソースタイプにアップグレードできる時点をクリックスタ管理者に説明します。

クラスタ管理者がリソースタイプをアップグレードできる条件は、次に示すように、RTRファイル内のリソースの各バージョンの`#$upgrade_from`ディレクティブのTunable属性に依存します。

- いつでもよい (ANYTIME)
- リソースが監視されていない場合のみ (WHEN\_UNMONITORED)
- リソースがオフラインである場合のみ (WHEN\_OFFLINE)
- リソースが無効である場合のみ (WHEN\_DISABLED)
- リソースグループが管理されていない場合のみ (WHEN\_UNMANAGED)

例4-2 クラスタ管理者がアップグレードできる時点を`#$upgrade_from`が定義する方法

次の例では、`#$upgrade_from`ディレクティブのTunable属性が、クラスタ管理者がリソースを新しいバージョンのリソースタイプにアップグレードできる条件にどのように影響するかを示します。

```
#$upgrade_from "1.1" WHEN_OFFLINE
#$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
```

バージョン	クラスタ管理者がリソースをアップグレードできる時点
1.1、1.2、1.3	リソースがオフラインのときのみ
2.0	リソースが監視されていないときのみ
2.1	すべての時刻
そのほかのすべてのバージョン	リソースグループが管理されていないときのみ

## リソースプロパティに対する変更に関する情報

クラスタ管理者がアップグレードを行う時点で、クラスタ管理者による既存のリソースのプロパティの変更を要求するリソースタイプに対して行われたすべての変更を説明します。

可能な変更には次のものが含まれます。

- 変更された既存のリソースタイププロパティのデフォルト設定
- 導入されたリソースタイプの新しい拡張プロパティ
- 取り消されたリソースタイプの既存のプロパティ
- リソースタイプに対して宣言された標準プロパティのセットに対する変更
- 変更されたリソースプロパティ (`min`、`max`、`arraymin`、`arraymax`、`default`、`tunability` など) の属性
- 宣言されたメソッドのセットに対する変更
- 変更されたメソッドまたは障害モニターの実装

## サンプルデータサービス

---

この章では、`in.named` アプリケーションを Sun Cluster データサービスとして稼働する HA-DNS について説明します。`in.named` デモンは Solaris におけるドメインネームサービス (DNS) の実装です。サンプルのデータサービスでは、Resource Management API を使用して、アプリケーションの高可用性を実現する方法を示します。

RMAPI は、シェルスクリプトと C プログラムの両方のインタフェースをサポートします。この章のサンプルアプリケーションはシェルスクリプトインタフェースで作成されています。

この章の内容は次のとおりです。

- 91 ページの「サンプルデータサービスの概要」
- 92 ページの「リソースタイプ登録ファイルの定義」
- 98 ページの「すべてのメソッドに共通な機能の提供」
- 103 ページの「データサービスの制御」
- 109 ページの「障害モニターの定義」
- 119 ページの「プロパティ更新の処理」

### サンプルデータサービスの概要

サンプルのデータサービスはクラスタのイベント (管理アクション、アプリケーションの異常終了、ノードまたはゾーンの異常終了など) に応じて、DNS アプリケーションの起動、停止、再起動や、クラスタのゾーンまたはノード間での DNS アプリケーションの切り替えを行います。

アプリケーションの再起動は、プロセス監視機能 (PMF) によって管理されます。アプリケーションの障害が再試行最大期間または再試行最大回数を超えると、障害モニターは、アプリケーションリソースを含むリソースグループを別のノードまたはゾーンにフェイルオーバーします。

サンプルのデータサービスは、`nslookup` コマンドを使用してアプリケーションが正常であることを確認する `PROBE` メソッドという形で障害監視機能を提供します。DNS サービスのハングを検出すると、`PROBE` は DNS アプリケーションをローカルで再起動することによって、この状況を修正しようとします。DNS アプリケーションをローカルで再起動することで状況が改善されず、サービスの問題が繰り返し検出される場合、`PROBE` は、サービスをクラスタ内の別のノードまたはゾーンにフェイルオーバーしようとします。

サンプルのデータサービスには、具体的に、次のような要素が含まれています。

- リソースタイプ登録ファイル - データサービスの静的なプロパティを定義します。
- `Start` コールバックメソッド - HA-DNS データサービスを含むリソースグループがオンラインになるときに RGM によって実行され、`in.named` デーモンを起動します。
- `Stop` コールバックメソッド - HA-DNS を含むリソースグループがオフラインになるときに RGM によって実行され、`in.named` デーモンを停止します。
- 障害モニター - DNS サーバーが動作しているかどうかを確認することによって、サービスの信頼性を検査します。障害モニターはユーザー定義の `PROBE` メソッドによって実装され、`Monitor_start` と `Monitor_stop` コールバックメソッドによって起動および停止されます。
- `Validate` コールバックメソッド - RGM によって実行され、サービスの構成ディレクトリがアクセス可能であるかどうかを検査します。
- `Update` コールバックメソッド - クラスタ管理者がリソースプロパティの値を変更したときに RGM によって呼び出され、障害モニターを再起動します。

## リソースタイプ登録ファイルの定義

この例で使用するサンプルのリソースタイプ登録 (RTR) ファイルは、DNS リソースタイプの静的な構成を定義します。このタイプのリソースは、RTR ファイルで定義されているプロパティを継承します。

RTR ファイル内の情報は、クラスタ管理者が HA-DNS データサービスを登録したときに Resource Group Manager (RGM) によって読み取られます。慣例により、RTR ファイルは `/opt/cluster/lib/rgm/rtrreg/` ディレクトリに置きます。パッケージインストーラは、Agent Builder が作成した RTR ファイルもこのディレクトリに置きます。

## RTR ファイルの概要

RTR ファイルの形式は明確に定義されています。リソースタイププログラム、システム定義リソースプロパティ、拡張プロパティという順番で並んでいます。詳

細は、`rt_reg(4)` のマニュアルページ、および [34 ページ](#) の「リソースとリソースタイププロパティの設定」を参照してください。

以降の節では、サンプル RTR ファイルの特定のプロパティについて説明します。これらの節には、ファイルのさまざまな部分のリストがあります。サンプル RTR ファイルの内容の完全なリストについては、[303 ページ](#) の「リソースタイプ登録ファイルのリスト」を参照してください。

## サンプル RTR ファイルのリソースタイププロパティ

次のリストに示すように、サンプルの RTR ファイルはコメントから始まり、そのあとに、HA-DNS 構成を定義するリソースタイププロパティが続きます。

---

注-リソースグループ、リソース、およびリソースタイプのプロパティ名は大文字と小文字が区別されません。プロパティ名を指定する際には、大文字と小文字を任意に組み合わせることができます。

---

```
#
# Copyright (c) 1998-2006 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;
```

---

ヒント-RTR ファイルの最初のエントリには、Resource\_type プロパティを宣言する必要があります。最初のエントリで宣言されていない場合は、リソースタイプの登録に失敗します。

---

次に、これらのプロパティについての情報を説明します。

- リソースタイプ名は、Resource\_type プロパティだけで指定できます (例: sample)。または接頭辞 *vendor-id*+ピリオド(.)+リソースタイププロパティ (例: SUNW.sample) の形式を使用することでも指定できます。  
*vendor-id* を指定する場合、リソースタイプを定義する企業の略号を使用します。リソースタイプ名はクラスタ内で一意である必要があります。
- RT\_version プロパティは、ベンダーによって指定されたサンプルのデータサービスのバージョンを識別します。
- API\_version プロパティは Sun Cluster のバージョンを識別します。たとえば、API\_version = 2 は、データサービスが Sun Cluster 3.0 以降の任意のバージョンの Sun Cluster で動作できることを示します。API\_version = 7 は、データサービスを 3.2 以降の任意のバージョンの Sun Cluster にインストールできることを示します。ただし、API\_version = 7 は、3.2 よりも前にリリースされたどのバージョンの Sun Cluster にもデータサービスをインストールできないことも示します。このプロパティについては、[253 ページの「資源タイプのプロパティ」](#)の API\_version の項目で詳しく説明しています。
- Failover = TRUE は、データサービスが、複数のノードまたはゾーン上で、同時にオンライン可能なリソースグループでは動作できないことを示します。
- RT\_basedir は相対パス (コールバックメソッドのパスなど) を補完するためのディレクトリパスで、/opt/SUNWsample/bin を指します。
- Start、Stop、Validate は、RGM によって実行される個々のコールバックメソッドプログラムへのパスを提供します。これらのパスは、RT\_basedir で指定されたディレクトリからの相対パスです。
- Pkglist は、SUNWsample をサンプルのデータサービスのインストールを含むパッケージとして識別します。

この RTR ファイルに指定されていないリソースタイププロパティ (Single\_instance、Init\_nodes、Installed\_nodes など) は、デフォルト値に設定されます。リソースタイププロパティの完全なリストとそのデフォルト値については、[253 ページの「資源タイプのプロパティ」](#)を参照してください。

クラスタ管理者は、RTR ファイルのリソースタイププロパティの値を変更できません。

## サンプル RTR ファイルのリソースプロパティ

慣習上、RTR ファイルでは、次のリソースプロパティをリソースタイププロパティのあとに宣言します。リソースプロパティには、Sun Cluster ソフトウェアが提供するシステム定義プロパティと、データサービス開発者が定義する拡張プロパティが含まれます。どちらのタイプの場合でも、Sun Cluster ソフトウェアが提供するプロパティ属性の数(最小、最大、デフォルト値など)を指定できます。

### RTR ファイルのシステム定義プロパティ

次のリストは、サンプル RTR ファイルのシステム定義プロパティを示しています。

```
# リソースタイプ宣言のあとに、中括弧に囲まれたリソースプロパティ宣言の
# リストが続く。プロパティ名宣言は、各エントリの左中括弧の直後にある
# 最初の属性である必要がある。
```

```
# <method>_timeout プロパティの値は、RGM がメソッドの呼び出しが
# 失敗したと結論するまでの時間 (秒) を設定する。
```

```
# すべてのメソッドタイムアウトの MIN 値は 60 秒に設定されている。
# これは、管理者が短すぎる時間を設定するのを防ぐためである。短すぎる
# 時間を設定すると、スイッチオーバーやフェイルオーバーの性能が上がらず、
# さらには予期せぬ RGM アクションが発生する可能性がある(誤った
# フェイルオーバー、ノードの再起動、リソースグループの
# ERROR_STOP_FAILED 状態への移行など、管理者の介入を必要とする
# RGM アクション)。
# メソッドタイムアウトに短すぎる時間を設定すると、データサービス全体の
# 可用性が低下する。
```

```
{
  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;
}
# 当該ノード上でアプリケーションを正常に起動できないと結論するまでに
# 指定された時間内 (Retry_Interval) に行う再試行回数
{
    PROPERTY = Retry_count;
    MIN=0;
    MAX=10;
    DEFAULT=2;
    TUNABLE = ANYTIME;
}
# Retry_Interval には 60 の倍数を指定する。これは、この値は秒から分に変換され、
# 端数が切り上げられるためである。
# たとえば、50 (秒) は 1 分に変換される。このプロパティ値は再試行回数
# (Retry_Count) のタイミングを指定する。
{
    PROPERTY = Retry_interval;
    MIN=60;
    MAX=3600;
    DEFAULT=300;
    TUNABLE = ANYTIME;
}
{
    PROPERTY = Network_resources_used;
    TUNABLE = AT_CREATION;
    DEFAULT = "";
}
```

Sun Cluster ソフトウェアはシステム定義プロパティを提供しますが、リソースプロパティ属性を使用すると、異なるデフォルト値を設定できます。リソースプロパティに適用するために利用できる属性の完全なリストについては、[301 ページの「リソースプロパティの属性」](#)を参照してください。

サンプル RTR ファイル内のシステム定義リソースプロパティについては、次の点に注意してください。

- Sun Cluster は、すべてのタイムアウトに最小値 (1 秒) とデフォルト値 (3600 秒 = 1 時間) を提供します。サンプル RTR ファイルは、最小タイムアウトを 60 秒に変更し、デフォルト値を 300 秒に変更しています。クラスタ管理者は、このデフォルト値を使用することも、タイムアウト値を 60 秒以上の別の値に変更することもできます。Sun Cluster は最大値を設定していません。
- プロパティ `Thorough_probe_interval`、`Retry_count`、`Retry_interval` の TUNABLE 属性は ANYTIME に設定されています。これらの設定は、データサービスが動作中でも、クラスタ管理者がこれらのプロパティの値を変更できることを意味します。上記のプロパティは、サンプルのデータサービスによって実装される障害モニターによって使用されます。サンプルのデータサービスは、管理アクションによってさまざまなリソースが変更されたときに障害モニターを停止および再起動するように、Update を実装します。[124 ページの「Update メソッドの仕組み」](#)を参照してください。
- リソースプロパティは次のように分類されます。
  - 必須。クラスタ管理者はリソースを作成するときに値を指定する必要があります。
  - 任意。クラスタ管理者が値を指定しない場合、システムがデフォルト値を提供します。
  - 条件付き。RGM は、RTR ファイル内にプロパティが宣言されている場合にかぎりプロパティを作成します。

サンプルのデータサービスの障害モニターは、`Thorough_probe_interval`、`Retry_count`、`Retry_interval`、`Network_resources_used` という条件付きプロパティを使用しているため、開発者はこれらのプロパティを RTR ファイルで宣言する必要があります。プロパティの分類の詳細については、[r\\_properties\(5\)](#) のマニュアルページ、または [264 ページの「リソースのプロパティ」](#)を参照してください。

## RTR ファイルの拡張プロパティ

次に、RTR ファイルの最後の例として、拡張プロパティを示します。

```
# 拡張プロパティ
```

```
# クラスタ管理者は、このプロパティの値として、アプリケーションによって
# 使用される構成ファイルが格納されているディレクトリのパスを指定する。
```

```
# このアプリケーション (DNS) は、PXFS 上の DNS 構成ファイルのパス(通常
# named.conf) のパスを指定する。
{
    PROPERTY = Confdir;
    EXTENSION;
    STRING;
    TUNABLE = AT_CREATION;
    DESCRIPTION = "The Configuration Directory Path";
}

# 検証の失敗が宣言されるまでのタイムアウト値 (秒)。
{
    PROPERTY = Probe_timeout;
    EXTENSION;
    INT;
    DEFAULT = 120;
    TUNABLE = ANYTIME;
    DESCRIPTION = "Time out value for the probe (seconds)";
}
```

サンプルの RTR ファイルは2つの拡張プロパティー、`Confdir`と`Probe_timeout`を定義します。`Confdir`プロパティーは、DNS 構成ディレクトリへのパスを指定します。このディレクトリには、DNS が正常に動作するために必要な `in.named` ファイルが格納されています。サンプルのデータサービスの `Start` と `Validate` メソッドはこのプロパティーを使用し、DNS を起動する前に、構成ディレクトリと `in.named` ファイルがアクセス可能であるかどうかを確認します。

データサービスが構成されるとき、`Validate` メソッドは、新しいディレクトリがアクセス可能であるかどうかを確認します。

サンプルのデータサービスの `PROBE` メソッドは、Sun Cluster コールバックメソッドではなく、ユーザー定義メソッドです。したがって、Sun Cluster はこの `Probe_timeout` プロパティーを提供しません。開発者は拡張プロパティーを RTR ファイルに定義し、クラスタ管理者が `Probe_timeout` の値を構成できるようにする必要があります。

## すべてのメソッドに共通な機能の提供

この節では、サンプルのデータサービスのすべてのコールバックメソッドで使用される次のような機能について説明します。

- 99 ページの「コマンドインタプリタの指定およびパスのエクスポート」
- 99 ページの「`PMF_TAG` と `SYSLOG_TAG` 変数の宣言」
- 100 ページの「関数の引数の構文解析」
- 102 ページの「エラーメッセージの生成」
- 102 ページの「プロパティー情報の取得」

## コマンドインタプリタの指定およびパスのエクスポート

シェルスクリプトの最初の行は、コマンドインタプリタを指定します。サンプルのデータサービスの各メソッドスクリプトは、次に示すように、コマンドインタプリタを指定します。

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

サンプルアプリケーション内のすべてのメソッドスクリプトは、Sun Cluster のバイナリとライブラリへのパスをエクスポートします。ユーザーの PATH 設定には依存しません。

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

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

## PMF\_TAG と SYSLOG\_TAG 変数の宣言

Validate を除くすべてのメソッドスクリプトは、pmfadm コマンドを使用して、データサービスまたはモニターのいずれかを起動または停止するか、あるいはリソース名を渡します。各スクリプトは変数 PMF\_TAG を定義し、pmfadm コマンドに渡すことによって、データサービスまたはモニターを識別できます。

同様に、各メソッドスクリプトは、logger コマンドを使用してメッセージをシステムログに記録します。各スクリプトは変数 SYSLOG\_TAG を定義し、-t オプションで logger に渡すことによって、メッセージが記録されるリソースのリソースタイプ、リソース名、リソースグループを識別できます。

すべてのメソッドは、次に示す例と同じ方法で SYSLOG\_TAG を定義します。

dns\_probe、dns\_svc\_start、dns\_svc\_stop、dns\_monitor\_check の各メソッドは、次のように PMF\_TAG を定義します。なお、pmfadm と logger は dns\_svc\_stop メソッドのものを使用しています。

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

```
PMF_TAG=$RESOURCE_NAME.named
```

```
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
```

```
# データサービスに SIGTERM シグナルを送り、タイムアウト値の 80%
```

```
# が経過するまで待機する。
pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
if [ $? -ne 0 ]; then
    logger -p ${SYSLOG_FACILITY}.info \
        -t [$SYSLOG_TAG] \
        "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
        SIGKILL"
```

dns\_monitor\_start、dns\_monitor\_stop、dns\_update メソッドは次のように PMF\_TAG を定義します。なお、pmfadm は dns\_monitor\_stop メソッドのものを使用しています。

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

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
...
# in.named が実行中であるかどうかを確認し、実行中であれば強制終了する。
if pmfadm -q $PMF_TAG.monitor; then
    pmfadm -s $PMF_TAG.monitor KILL
```

## 関数の引数の構文解析

RGM は、次に示すように、Validate を除くすべてのコールバックメソッドを実行します。

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

*method\_name* は、コールバックメソッドを実装するプログラムのパス名です。データサービスは、各メソッドのパス名を RTR ファイルに指定します。このようなパス名は、RTR ファイルの RT\_basedir プロパティに指定されたディレクトリからのパスになります。たとえば、サンプルのデータサービスの RTR ファイルでは、ベースディレクトリとメソッド名は次のように指定されます。

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

コールバックメソッドの引数はすべて、次のようにフラグ付きの値として渡されます。-R 引数はリソースインスタンスの名前を示します。-T 引数はリソースのタイプを示します。-G 引数はリソースが構成されているグループを示します。コールバックメソッドの詳細については、rt\_callbacks(1HA) のマニュアルページを参照してください。

注-Validate メソッドを呼び出すときは、追加の引数(リソースのプロパティ値と呼び出しが行われるリソースグループ)を使用します。詳細は、[119 ページの「プロパティ更新の処理」](#)を参照してください。

各コールバックメソッドには、渡された引数を構文解析する関数が必要です。すべてのコールバックメソッドには同じ引数が渡されるので、データサービスは、アプリケーション内のすべてのコールバックメソッドで使用される単一の構文解析関数を提供します。

次のサンプルに、サンプルアプリケーションのコールバックメソッドに使用される `parse_args()` 関数を示します。

```
#####
# プログラム引数の解析。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソース名。
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されたリソース
                # グループ名。
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプ名。
                RESOURCETYPE_NAME=$OPTARG
                ;;
            *)
                logger -p ${SYSLOG_FACILITY}.err \
                    -t [${RESOURCETYPE_NAME},${RESOURCEGROUP_NAME},${RESOURCE_NAME}] \
                    "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
    done
}
```

---

注- サンプルのアプリケーションの PROBE メソッドはユーザー定義メソッドですが、Sun Cluster コールバックメソッドと同じ引数で呼び出されます。したがって、このメソッドには、ほかのコールバックメソッドにより使用されるものと同じ構文解析関数が含まれています。

---

構文解析関数は、次に示すように、MAIN の中で呼び出されます。

```
parse_args "$@"
```

## エラーメッセージの生成

エンドユーザーに対してエラーメッセージを出力するには、コールバックメソッドは `syslog()` 関数を使用する必要があります。サンプルのデータサービスのすべてのコールバックメソッドは、次に示すように、`scha_cluster_get` コマンドを使用し、クラスタログ用に使用されている `syslog()` 関数番号を取得します。

```
SYSLOG_FACILITY='scha_cluster_get -0 SYSLOG_FACILITY'
```

この値はシェル変数 `SYSLOG_FACILITY` に格納されます。 `logger` コマンドの機能として使用すると、エラーメッセージをクラスタログに記録できます。たとえば、サンプルのデータサービスの `Start` メソッドは、次に示すように、`syslog()` 関数を取得し、データサービスが起動したことを示すメッセージを記録します。

```
SYSLOG_FACILITY='scha_cluster_get -0 SYSLOG_FACILITY'
```

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

詳細については、`scha_cluster_get(1HA)` のマニュアルページを参照してください。

## プロパティ情報の取得

ほとんどのコールバックメソッドは、データサービスのリソースとリソースタイプのプロパティについての情報を取得する必要があります。このために、API は `scha_resource_get()` 関数を提供しています。

システム定義プロパティと拡張プロパティの両方が使用できます。システム定義プロパティは事前に定義されています。拡張プロパティは、データサービス開発者が RTR ファイルに定義します。

`scha_resource_get()` を使用してシステム定義プロパティの値を取得するときには、`-O` オプションでプロパティの名前を指定します。このコマンドは、プロパティの値だけを戻します。たとえば、サンプルのデータサービスの `Monitor_start` メソッドは検証プログラムを特定し、起動できるようにしておく必要があります。検証プログラムはデータサービスのベースディレクトリ (`RT_basedir` プロパティが指すディレクトリ) 内に存在します。したがって、`Monitor_start` メソッドは、次に示すように、`RT_basedir` の値を取得し、その値を `RT_BASEDIR` 変数に格納します。

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

拡張プロパティの場合、データサービス開発者は、このプロパティが拡張プロパティであることを示す `-O` オプションを使用する必要があります。また、最後の引数としてプロパティの名前を指定する必要があります。拡張プロパティの場合、このコマンドは、プロパティのタイプと値の両方を戻します。たとえば、サンプルのデータサービスの検証プログラムは、次に示すように、`Probe_timeout` 拡張プロパティのタイプと値を取得し、次に `awk` コマンドを使用して値だけを `PROBE_TIMEOUT` シェル変数に格納します。

```
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}''
```

## データサービスの制御

データサービスは、クラスタ内でアプリケーションデーモンを起動するために `Start` メソッドまたは `Prenet_start` メソッドを提供し、クラスタ内でアプリケーションデーモンを停止するために `Stop` メソッドまたは `Postnet_stop` メソッドを提供する必要があります。サンプルのデータサービスは、`Start` メソッドと `Stop` メソッドを実装します。代わりに `Prenet_start` メソッドと `Postnet_stop` メソッドを使用する場合については、[47 ページの「Start および Stop メソッドを使用するかどうかの決定」](#) を参照してください。

## Start メソッドの仕組み

データサービスリソースのあるリソースグループがクラスタノードまたはゾーン上でオンラインになったとき、またはリソースグループがすでにオンラインになっていてリソースが有効になったとき、RGM はそのノードまたはゾーン上で `Start` メソッドを実行します。サンプルのアプリケーションでは、`Start` メソッドはそのノード上の大域ゾーン内で `in.named` DNS デーモンを起動します。

この節では、サンプルのアプリケーションの `Start` メソッドの重要な部分だけを説明します。 `parse_args()` 関数など、すべてのコールバックメソッドに共通な機能につ

いては説明しません。また、`syslog()` 関数の使用法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

Start メソッドの完全なリストについては、307 ページの「Start メソッドのコードリスト」を参照してください。

## Start メソッドの動作

DNS を起動する前に、サンプルのデータサービスの Start メソッドは、構成ディレクトリと構成ファイル (`named.conf`) がアクセス可能で利用可能であるかどうかを確認します。DNS が正常に動作するためには、`named.conf` の情報が重要です。

このコールバックメソッドは、PMF (`pmfadm`) を使って DNS デーモン (`in.named`) を起動します。DNS がクラッシュしたり、起動に失敗したりすると、PMF は、指定の期間に所定の回数だけ DNS デーモンの起動を試行します。再試行の回数と期間は、データサービスの RTR ファイル内のプロパティで指定されます。

## 構成の確認

DNS が動作するためには、構成ディレクトリ内の `named.conf` ファイルからの情報が必要です。したがって、Start メソッドは、DNS を起動しようとする前にいくつかの妥当性検査を実行し、ディレクトリやファイルがアクセス可能であるかどうかを確認します。

`Confdir` 拡張プロパティは、構成ディレクトリへのパスを提供します。プロパティ自身は RTR ファイルに定義されています。しかし、実際の位置は、クラスタ管理者がデータサービスを構成するときに指定します。

サンプルのデータサービスでは、Start メソッドは `scha_resource_get()` 関数を使用して構成ディレクトリの位置を取得します。

---

注 - `Confdir` は拡張プロパティであるため、`scha_resource_get()` はタイプと値の両方を戻します。したがって、`awk` コマンドで値だけを取得し、シェル変数 `CONFIG_DIR` にその値を格納します。

---

```
# クラスタ管理者がリソースの追加時に設定した Confdir の
# 値を検索。
config_info='scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir'
```

```
# scha_resource_get は、拡張プロパティの値とともにタイプを返す。
# 拡張プロパティの値だけを取得。
CONFIG_DIR='echo $config_info | awk '{print $2}''
```

Start メソッドは CONFIG\_DIR の値を使用し、ディレクトリがアクセス可能であるかどうかを確認します。アクセス可能ではない場合、Start メソッドはエラーメッセージを記録し、エラー状態で終了します。106 ページの「Start の終了状態」を参照してください。

```
# CONFIG_DIR がアクセス可能かどうかを確認。
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
```

アプリケーションデーモンを起動する前に、このメソッドは最終検査を実行し、named.conf ファイルが存在するかどうかを確認します。ファイルが存在しない場合、Start メソッドはエラーメッセージを記録し、エラー状態で終了します。

```
# データファイルに相対パス名が含まれている場合は CONFIG_DIR
# ディレクトリに移動。
cd $CONFIG_DIR

# named.conf ファイルが CONFIG_DIR ディレクトリに存在することを確認。
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
```

## アプリケーションの起動

このメソッドは、プロセス監視機能 (pmfadm) を使用してアプリケーションを起動します。pmfadm コマンドを使用すると、指定した期間内にアプリケーションの再起動を試みる回数を設定できます。RTR ファイルには、2つのプロパティがあり、Retry\_count は、アプリケーションを再起動する回数を指定し、Retry\_interval は、アプリケーションを再起動する期間を指定します。

Start メソッドは、scha\_resource\_get() 関数を使用して Retry\_count と Retry\_interval の値を取得し、これらの値をシェル変数に格納します。次に Start メソッドは、-n オプションと -t オプションを使用し、これらの値を pmfadm に渡します。

```
# RTR ファイルから再試行回数の値を取得する。
RETRY_CNT='scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME'
# RTR ファイルから次の再試行までの時間 (秒数) を取得。この値は pmfadm
# に渡されるため分数に変換される。変換時に端数が切り上げられる点に注意。
```

```
# たとえば 50 秒は 1 分に変換される。
((RETRY_INTRVAL='scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME' / 60))

# PMF 制御下で in.named デーモンを起動する。RETRY_INTERVAL の期間、
# $RETRY_COUNT の回数だけクラッシュおよび再起動できる。
# それ以上の回数クラッシュした場合、PMF はそれ以上再試行しない。
# <$PMF_TAG> タグで登録済みのプロセスがある場合、PMF はプロセスが
# すでに実行中であるという警告メッセージを送信する。
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# HA-DNS が起動していることを示すメッセージを記録。
if [ $? -eq 0 ]; then
    logger -p ${SYSLOG_FACILITY}.err \
        -t [$SYSLOG_TAG] \
        "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

## Start の終了状態

Start メソッドは、実際のアプリケーションが本当に動作して実行可能になるまで、成功状態で終了してはなりません。特に、ほかのデータサービスが依存している場合は注意する必要があります。これを実現するための1つの方法は、Start メソッドが終了する前に、アプリケーションが動作しているかどうかを確認することです。複雑なアプリケーション(データベースなど)の場合、RTR ファイルの Start\_timeout プロパティに十分高い値を設定することによって、アプリケーションが初期化され、クラッシュ回復を実行できる時間を提供します。

---

注- サンプルのデータサービスのアプリケーションリソース(DNS)は直ちに起動するため、サンプルのデータサービスは、成功状態で終了する前に、ポーリングでアプリケーションが動作していることを確認していません。

---

このメソッドがDNSの起動に失敗し、失敗状態で終了すると、RGMは Failover\_mode プロパティを検査し、どのように対処するかを決定します。サンプルのデータサービスは明示的に Failover\_mode プロパティを設定していないため、このプロパティはデフォルト値 NONE が設定されています(ただし、クラスタ管理者がデフォルト値を変更して異なる値を指定していないと仮定します)。したがって、RGMは、データサービスの状態を設定するだけで、ほかのアクションは行いません。同じノードまたはゾーン上での再起動や、別のノードまたはゾーンへのフェイルオーバーは、クラスタ管理者が行う必要があります。

## Stop メソッドの仕組み

HA-DNS リソースのあるリソースグループがクラスタノードまたはゾーン上でオフラインになったとき、またはリソースグループがオンラインでリソースが無効になったとき、RGMはそのノードまたはゾーン上で Stop メソッドを実行します。このメソッドは、そのノードまたはゾーン上で `in.named` (DNS) デーモンを停止します。

この節では、サンプルのアプリケーションの Stop メソッドの重要な部分だけを説明します。 `parse_args()` 関数など、すべてのコールバックメソッドに共通な機能については説明しません。また、 `syslog()` 関数の使用法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

Stop メソッドの完全なリストについては、310 ページの「Stop メソッドのコードリスト」を参照してください。

## Stop メソッドの動作

データサービスを停止するときは、考慮すべきことが2点あります。1点は、停止処理を正しい順序で行うことです。停止処理を正しい順序で行う最良の方法は、 `pmfadm` 経由で `SIGTERM` シグナルを送信することです。

もう1点は、データサービスが本当に停止していることを保証することによって、データベースが `Stop_failed` 状態にならないようにすることです。データサービスをこの状態にする最良の方法は、 `pmfadm` 経由で `SIGKILL` シグナルを送信することです。

サンプルのデータサービスの `STOP` メソッドは、このような点を考慮しています。まず、 `SIGTERM` シグナルを送信します。このシグナルがデータサービスの停止に失敗した場合は、 `SIGKILL` シグナルを送信します。

DNS を停止しようとする前に、この Stop メソッドは、プロセスが実際に動作しているかどうかを確認します。プロセスが動作している場合には、Stop は `PMF` (`pmfadm`) を使ってプロセスを停止します。

この Stop メソッドは何回か呼びだしてもその動作が変わらないことが保証されます。RGM は、Start メソッドの呼び出しでまずデータサービスを起動せずに、Stop メソッドを2回呼び出すことはありません。しかし、RGM は、リソースが起動されていないか、あるいは、リソースが自発的に停止している場合でも、Stop メソッドをリソース上で呼び出すことができます。つまり、DNS がすでに動作していない場合でも、この Stop メソッドは成功状態で終了します。

## アプリケーションの停止

Stop メソッドは、データサービスを停止するために2段階の方法を提供します。 `pmfadm` 経由で `SIGTERM` シグナルを使用する規則正しい方法と、 `SIGKILL` シグナルを使用する強制的な方法です。Stop メソッドは、Stop メソッドが戻るまでの時間を示す

Stop\_timeout 値を取得します。Stop メソッドはこの時間の 80% を規則正しい方法に割り当て、15% を強制的な方法に割り当てます (5% は予約済み)。次の例を参照してください。

```
STOP_TIMEOUT='scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME'
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))
((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))
```

Stop メソッドは pmfadm -q を使用し、DNS デーモンが動作しているかどうかを確認します。DNS デーモンが動作している場合、Stop はまず pmfadm -s を使用して TERM シグナルを送信し、DNS プロセスを終了します。このシグナルを送信してからタイムアウト値の 80% が経過してもプロセスが終了しない場合、Stop は SIGKILL シグナルを送信します。このシグナルを送信してからタイムアウト値の 15% 以内にプロセスが終了しない場合、Stop メソッドはエラーメッセージを記録し、エラー状態で終了します。

pmfadm がプロセスを終了した場合、STOP メソッドはプロセスが停止したことを示すメッセージを記録し、成功状態で終了します。

DNS プロセスが動作していない場合、STOP メソッドは DNS プロセスが動作していないことを示すメッセージを記録しますが、成功状態で終了します。次のコード例に、Stop メソッドがどのように pmfadm を使用して DNS プロセスを停止するかを示します。

```
# in.named が実行中であるかどうかを確認し、実行中であれば強制終了する。
if pmfadm -q $PMF_TAG; then
# データサービスに SIGTERM シグナルを送り、タイムアウト値の 80%
# が経過するまで待機する。
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"

# SIGTERM シグナルでデータサービスが停止しないので、今度は SIGKILL を
# 使って、合計タイムアウト値の残りの 15% が経過するまで待機する。
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
```

```

# この時点でデータサービスは実行されていない。メッセージを記録して
# 成功状態で終了する。
logger -p ${SYSLOG_FACILITY}.err \
      -t [${SYSLOG_TAG}] \
      "HA-DNS is not started"

# HA-DNS が実行中でなくても、データサービスリソースを STOP_FAILED
# 状態にするのを避けるため成功状態で終了する。
exit 0
fi

# DNS の停止に成功。メッセージを記録して成功状態で終了する。
logger -p ${SYSLOG_FACILITY}.err \
      -t [${RESOURCE_TYPE_NAME},${RESOURCE_GROUP_NAME},${RESOURCE_NAME}] \
      "HA-DNS successfully stopped"
exit 0

```

## Stop の終了状態

Stop メソッドは、実際のアプリケーションが本当に停止するまで、成功状態で終了してはなりません。特に、ほかのデータサービスが依存している場合は注意する必要があります。そうしなければ、データが破壊される可能性があります。

複雑なアプリケーション(データベースなど)の場合、RTR ファイルの `Stop_timeout` プロパティに十分高い値を設定することによって、アプリケーションが停止中にクリーンアップできる時間を提供します。

このメソッドが DNS の停止に失敗し、失敗状態で終了すると、RGM は `Failover_mode` プロパティを検査し、どのように対処するかを決定します。サンプルのデータサービスは明示的に `Failover_mode` プロパティを設定していないため、このプロパティはデフォルト値 `NONE` が設定されています(ただし、クラスタ管理者がデフォルト値を変更して異なる値を指定していないと仮定します)。したがって、RGM は、データサービスの状態を `Stop_failed` に設定するだけで、ほかのアクションは行いません。アプリケーションを強制的に停止し、`Stop_failed` 状態をクリアするには、クラスタ管理者の操作が必要です。

## 障害モニターの定義

サンプルのアプリケーションは、DNS リソース (`in.named`) の信頼性を監視する基本的な障害モニターを実装します。

障害モニターは、次の要素から構成されます。

- `dns_probe - nslookup` を使用し、サンプルのデータサービスの制御下にある DNS リソースが動作しているかどうかを確認するユーザー定義プログラム。DNS が動作していない場合、このメソッドは DNS をローカルで再起動しようとします。あるいは、再起動の再試行回数によっては、RGM がデータサービスを別のノードまたはゾーンに再配置することを要求します。
- `dns_monitor_start - dns_probe` を起動するコールバックメソッド。監視が有効である場合、RGM は、サンプルのデータサービスがオンラインになったあと、自動的に `dns_monitor_start` を呼び出します。
- `dns_monitor_stop - dns_probe` を停止するコールバックメソッド。RGM は、サンプルのデータサービスがオフラインになる前に、自動的に `dns_monitor_stop` を呼び出します。
- `dns_monitor_check - PROBE` プログラムがデータサービスを新しいノードまたはゾーンにフェイルオーバーするとき、`Validate` メソッドを呼び出し、構成ディレクトリが利用可能であるかどうかを確認するコールバックメソッド。

## 検証プログラムの仕組み

`dns_probe` プログラムは、サンプルのデータサービスの管理下にある DNS リソースが動作しているかどうかを確認する、連続して動作するプロセスを実行します。`dns_probe` は、サンプルのデータサービスがオンラインになったあと、RGM によって自動的に実行される `dns_monitor_start` メソッドによって起動されます。データサービスは、サンプルのデータサービスがオフラインになる前、RGM によって実行される `dns_monitor_stop` メソッドによって停止されます。

この節では、サンプルのアプリケーションの `PROBE` メソッドの重要な部分だけを説明します。`parse_args()` 関数など、すべてのコールバックメソッドに共通な機能については説明しません。また、`syslog()` 関数の使用法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

`PROBE` メソッドの完全なリストについては、313 ページの「`PROBE` プログラムのコードリスト」を参照してください。

## 検証プログラムの動作

検証プログラムは無限ループで動作します。検証プログラムは、`nslookup` を使用し、適切な DNS リソースが動作しているかどうかを確認します。DNS が動作している場合、検証プログラムは一定の期間 (`Thorough_probe_interval` システム定義プロパティに設定されている期間) だけ休眠し、再び検証を行います。DNS が動作していない場合、検証プログラムは DNS をローカルで再起動しようとするか、再起動の再試行回数によっては、RGM がデータサービスを別のノードまたはゾーンに再配置することを要求します。

## プロパティ値の取得

このプログラムには、次のプロパティの値が必要です。

- `Thorough_probe_interval` - 検証プログラムが休眠する期間を設定します。
- `Probe_timeout` - `nslookup` コマンドが検証を行う期間(タイムアウト値)を設定します。
- `Network_resources_used` - DNS が動作するサーバーを設定します。
- `Retry_count` と `Retry_interval` - 再起動を行う回数と期間を設定します。
- `RT_basedir` - PROBE プログラムと `gettime` ユーティリティが格納されているディレクトリを設定します。

`scha_resource_get()` 関数は、次に示すように、上記プロパティの値を取得し、シェル変数に格納します。

```
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 - システム定義プロパティ (`Thorough_probe_interval` など) の場合、`scha_resource_get()` 関数は値だけを返します。拡張プロパティ (`Probe_timeout` など) の場合、`scha_resource_get()` 関数はタイプと値を返します。値だけを取得するには `awk` コマンドを使用します。

---

## サービスの信頼性の検査

検証プログラム自身は、`nslookup` コマンドの `while` による無限ループです。 `while` ループの前に、`nslookup` の応答を保管する一時ファイルを設定します。 `probefail` 変数と `retries` 変数は 0 に初期化されます。

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

while ループは、次の作業を行います。

- 検証プログラム用の休眠期間を設定します。
- hatimerun を使用して nslookup を起動し、Probe\_timeout の値を渡し、ターゲットホストを指定します。
- nslookup の戻りコード (成功または失敗) に基づいて、probfail 変数を設定します。
- probfail が 1 (失敗) に設定された場合、nslookup への応答がサンプルのデータサービスから来ており、ほかの DNS サーバーから来ていないことを確認します。

次に、while ループコードを示します。

```
while :
do
# 検証が実行される時間は THOROUGH_PROBE_INTERVAL プロパティ
# に指定されている。したがって、THOROUGH_PROBE_INTERVAL の間
# 検証が Sleep するように設定する。
sleep $PROBE_INTERVAL

# DNS がサービスを提供している IP アドレス上で nslookup コマンドを実行する。
hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
> $DNSPROBEFILE 2>&1

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

# nslookup への応答が /etc/resolv.conf ファイルに指定されている
# そのほかのネームサーバーではなく HA-DNS サーバーから返されて
# いることを確認する。
if [ $probfail -eq 0 ]; then
# nslookup 照会に応答したサーバーの名前を取得する。
SERVER=' awk ' $1=="Server:" { print $2 }' \
$DNSPROBEFILE | awk -F. ' { print $1 } ' '
if [ -z "$SERVER" ]; then
    probfail=1
else
    if [ $SERVER != $DNS_HOST ]; then
        probfail=1
    fi
```

```
fi
fi
```

## 再起動とフェイルオーバーの評価

probefail 変数が 0 (成功) 以外である場合、nslookup コマンドがタイムアウトしたか、あるいは、サンプルのサービスの DNS 以外のサーバーから応答が来ていることを示します。どちらの場合でも、DNS サーバーは期待どおりに機能していないので、障害モニターは `decide_restart_or_failover()` 関数を呼び出し、データサービスをローカルで起動するか、RGM がデータサービスを別のノードまたはゾーンに再配置することを要求するかを決定します。probefail 変数が 0 の場合、検証が成功したことを示すメッセージが生成されます。

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

`decide_restart_or_failover()` 関数は、再試行最大期間 (`Retry_interval`) と再試行最大回数 (`Retry_count`) を使用し、DNS をローカルで再起動するか、RGM がデータサービスを別のノードまたはゾーンに再配置することを要求するかを決定します。この関数は、次のような条件付きコードを実装します。コードリストについては、[313 ページの「PROBE プログラムのコードリスト」](#)にある `decide_restart_or_failover()` を参照してください。

- 最初の障害である場合、データサービスをローカルで再起動します。エラーメッセージを記録し、`retries` 変数の再試行カウンタをインクリメントします。
- 最初の障害ではなく、再試行時間が再試行最大期間を過ぎている場合、データサービスをローカルで再起動します。エラーメッセージを記録し、再試行カウンタをリセットし、再試行時間をリセットします。
- 再試行時間が再試行最大期間を過ぎておらず、再試行カウンタが再試行最大回数を超えている場合、別のノードまたはゾーンにフェイルオーバーします。フェイルオーバーが失敗すると、エラーメッセージを記録し、検証プログラムを状態 1 (失敗) で終了します。
- 再試行時間が再試行最大期間を過ぎておらず、再試行カウンタが再試行最大回数を超えていない場合、データサービスをローカルで再起動します。エラーメッセージを記録し、`retries` 変数の再試行カウンタをインクリメントします。

期限 (再試行最大期間) 内に再起動の回数 (再試行カウンタ) が制限 (再試行最大回数) に到達した場合、この関数は、RGM がデータサービスを別のノードまたはゾーンに再配置することを要求します。再起動の回数が制限に到達していない場合、あるいは、再試行最大期間を過ぎていて、再試行カウンタをリセットする場合、この関数は DNS を同じノードまたはゾーン上で再起動しようとします。

この関数については、次の点に注意してください。

- `gettime` ユーティリティを使用すると、再起動間の時間を追跡できます。これは C プログラムで、`(RT_basedir)` ディレクトリ内にあります。
- `Retry_count` と `Retry_interval` のシステム定義リソースプロパティは、再起動を行う回数と期間を決定します。RTR ファイルでは、これらのプロパティのデフォルト値は、再試行が 2 回、期間が 5 分 (300 秒) ですが、クラスタ管理者はこれらの値を変更できます。
- `restart_service()` 関数は、同じノードまたはゾーン上でデータサービスの再起動を試行する場合に呼び出されます。この関数の詳細については、次の節である [114 ページの「データサービスの再起動」](#) を参照してください。
- `scha_control()` API 関数を `SCHA_GIVEOVER` 引数付きで実行すると、サンプルデータサービスのあるリソースグループがオフラインになり、別ノードまたはゾーン上でオンラインに戻ります。

## データサービスの再起動

`restart_service()` 関数は、`decide_restart_or_failover()` によって呼び出され、同じノードまたはゾーン上でデータサービスの再起動を試行します。

この関数は次の作業を行います。

- データサービスがまだ PMF 下に登録されているかどうかを判別します。  
サービスがまだ登録されている場合、この関数は次の作業を行います。
  - データサービスの Stop メソッド名と `Stop_timeout` 値を取得します。
  - `hatimerun` を使用してデータサービスの Stop メソッドを起動し、`Stop_timeout` 値を渡します。
  - データサービスが正常に停止した場合は、データサービスの Start メソッド名と `Start_timeout` 値を取得します。
  - `hatimerun` を使用してデータサービスの Start メソッドを起動し、`Start_timeout` 値を渡します。
- データサービスが PMF 下に登録されていない場合は、データサービスが PMF 下で許可されている再試行最大回数を超過していることを示しています。  
`scha_control` コマンドが `GIVEOVER` 引数付きで実行され、それによってデータサービスが別のノードまたはゾーンにフェイルオーバーします。

```
function restart_service
{
```

```
    # データサービスを再起動するには、まずデータサービス自体が PMF に
    # 登録されているかどうかを確認する。
    pmfadm -q $PMF_TAG
    if [[ $? -eq 0 ]]; then
```

```

# データサービスの TAG が PMF に登録されている場合、データ
# サービスを停止し、再起動する。

# 当該リソースの Stop メソッド名と
# STOP_TIMEOUT 値を取得する。
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

# 当該リソースの START メソッド名と START_TIMEOUT 値
# を取得する。
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
# データサービスの TAG が PMF に登録されていない場合、
# データサービスが PMF で許可されている最大再試行回数
# を超過している。したがって、データサービスを再起動しては
# ならない。代わりに、同じクラスタ内の別のノードへの
# フェイルオーバーを試みる。
scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
-R $RESOURCE_NAME
fi

return 0
}

```

## 検証プログラムの終了状態

ローカルでの再起動が失敗したり、別のノードまたはゾーンへのフェイルオーバーが失敗したりすると、サンプルのデータサービスの PROBE プログラムは失敗で終了します。このプログラムは「Failover attempt failed」(フェイルオーバーは失敗しました)というメッセージを記録します。

## Monitor\_start メソッドの仕組み

RGM は、サンプルデータサービスがオンラインになったあとに、Monitor\_start メソッドを呼び出して dns\_probe メソッドを起動します。

この節では、サンプルアプリケーションの Monitor\_start メソッドの重要な部分だけを説明します。parse\_args() 関数など、すべてのコールバックメソッドに共通な機能については説明しません。また、syslog() 関数の使用法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

Monitor\_start メソッドの完全なリストについては、319 ページの「Monitor\_start メソッドのコードリスト」を参照してください。

## Monitor\_start メソッドの動作

このメソッドは PMF (pmfadm) を使って検証プログラムを起動します。

## 検証プログラムの起動

Monitor\_start メソッドは、RT\_basedir プロパティの値を取得し、PROBE プログラムの完全パス名を構築します。このメソッドは、pmfadm の無限再試行オプション (-n -1、-t -1) を使用して検証プログラムを起動します。つまり、検証プログラムの起動に失敗しても、PMF は検証プログラムを無限に起動しようとしています。

```
# リソースの RT_BASEDIR プロパティを取得し、検証プログラムが存在する
# 場所を確認する。
RT_BASEDIR=$(scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME)

# PMF の制御下でデータサービスの検証を開始する。無限再試行オプションを使って
# 検証プログラムを起動する。リソースの名前、タイプ、グループを検証
# プログラムに渡す。
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
    -T $RESOURCETYPE_NAME
```

## Monitor\_stop メソッドの仕組み

RGMは、サンプルデータサービスがオフラインになるときに、Monitor\_stop メソッドを呼び出して dns\_probe の実行を停止します。

この節では、サンプルアプリケーションの Monitor\_stop メソッドの重要な部分だけを説明します。parse\_args() 関数など、すべてのコールバックメソッドに共通な機能については説明しません。また、syslog() 関数の使用方法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

Monitor\_stop メソッドの完全なリストについては、321 ページの「Monitor\_stop メソッドのコードリスト」を参照してください。

## Monitor\_stop メソッドの動作

このメソッドは、PMF (pmfadm) を使用して検証プログラムが動作しているかどうかを判断し、動作している場合は検証プログラムを停止します。

## 検証プログラムの停止

Monitor\_stop メソッドは、pmfadm -q を使用して検証プログラムが動作しているかどうかを判断し、動作している場合は pmfadm -s を使用して検証プログラムを停止します。検証プログラムがすでに停止している場合でも、このメソッドは成功状態です。これによって、メソッドが呼び出し回数に依存しないことが保証されます。



注意 - 必ず KILL シグナルと pmfadm を使用して検証プログラムを停止してください。TERM などのマスク可能なシグナルは使用しないでください。そうしないと、Monitor\_stop メソッドが無限にハングし、結果としてタイムアウトする可能性があります。これは、データサービスを再起動またはフェイルオーバーする必要がある場合に PROBE メソッドは scha\_control() を呼び出すためです。データサービスをオフラインにするプロセスの一部として scha\_control() が Monitor\_stop を呼び出す場合、Monitor\_stop がマスク可能なシグナルを使用すると、Monitor\_stop は scha\_control() の完了を待機してハングし、scha\_control() は Monitor\_stop の完了を待機してハングします。

# 検証プログラムが動作しているかどうかを判断し、動作している場合は停止する。

```
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
        # 検証プログラムの停止に成功。メッセージを記録する。
        logger -p ${SYSLOG_FACILITY}.err \
            -t [${SYSLOG_TAG}] \
            "${ARGV0} Monitor for resource " $RESOURCE_NAME \
            " successfully stopped"
    fi
fi
exit 0

```

## Monitor\_stopの終了状態

PROBE メソッドを停止できない場合、Monitor\_stop メソッドはエラーメッセージを記録します。RGM は、主ノードまたはゾーン上でサンプルのデータサービスを MONITOR\_FAILED 状態にするため、そのノードに障害が発生することがあります。

Monitor\_stop メソッドは、検証プログラムが停止するまで終了してはなりません。

## Monitor\_checkメソッドの仕組み

PROBE メソッドが、データサービスのあるリソースグループを新しいノードまたはゾーンにフェイルオーバーしようとするときに、RGM は必ず Monitor\_check メソッドを呼び出します。

この節では、サンプルアプリケーションの Monitor\_check メソッドの重要な部分だけを説明します。parse\_args() 関数など、すべてのコールバックメソッドに共通な機能については説明しません。また、syslog() 関数の使用法についても説明しません。共通の機能については、98 ページの「すべてのメソッドに共通な機能の提供」を参照してください。

Monitor\_check メソッドの完全なリストについては、323 ページの「Monitor\_check メソッドのコードリスト」を参照してください。

Monitor\_check メソッドは、並行して実行中のそのほかのメソッドと競合しない方法で実装する必要があります。

Monitor\_check メソッドは Validate メソッドを呼び出し、新しいノードまたはゾーン上で DNS 構成ディレクトリが利用可能かどうかを確認します。Confdir 拡張プロパティが DNS 構成ディレクトリを指します。したがって、Monitor\_check は Validate メソッドのパスと名前、および Confdir の値を取得します。Monitor\_check は、次のように、この値を Validate に渡します。

```

# リソースタイプの RT_BASEDIR プロパティから Validate メソッドの
# 完全パスを取得する。
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \

```

```

-G $RESOURCEGROUP_NAME

# 当該リソースの Validate メソッド名を取得する。
VALIDATE_METHOD=`scha_resource_get -O Validate \
  -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME`

# データサービスを起動するための Confdir プロパティの値を取得する。
# 入力されたリソース名とリソースグループを使用し、リソースを追加
# するときに設定した Confdir の値を取得する。
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
  -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get は、Confdir 拡張プロパティの値とともにタイプも戻す。
# awk を使用し、Confdir 拡張プロパティの値だけを取得する。
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Validate メソッドを呼び出し、データサービスを新しいノードに
# フェイルオーバーできるかどうかを確認する。
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
  -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

ノードまたはゾーンがデータサービスのホストとして最適であるかどうかをサン
プルアプリケーションが確認する方法については、119 ページの「Validate メソッドの
仕組み」を参照してください。

```

## プロパティ更新の処理

サンプルのデータサービスは、クラスタ管理者によるプロパティの更新を処理するために、Validate メソッドと Update メソッドを実装します。

### Validate メソッドの仕組み

リソースが作成されたとき、および、リソースまたは(リソースを含む)リソースグループのプロパティが管理アクションによって更新される時、RGM は Validate メソッドを呼び出します。RGM は、作成または更新が行われる前に、Validate メソッドを呼び出します。任意のノードまたはゾーン上でメソッドから失敗の終了コードが戻ると、作成または更新は取り消されます。

RGM が Validate メソッドを呼び出すのは、クラスタ管理者がリソースまたはリソースグループのプロパティを変更したときだけです。RGM がプロパティを設定したときや、モニターがリソースプロパティ Status および Status\_msg を設定したときではありません。

---

注-PROBE メソッドがデータサービスを新しいノードまたはゾーンにフェイルオーバーしようとする際には常に、Monitor\_check メソッドは Validate メソッドを明示的に呼び出します。

---

## Validate メソッドの動作

RGM は、ほかのメソッドに渡す引数以外にも、引数を追加して Validate メソッドを呼び出します。この追加引数には、更新されるプロパティと値が含まれます。したがって、サンプルのデータサービスの Validate メソッドは、追加の引数を処理する別の parse\_args() 関数を実装する必要があります。

サンプルのデータサービスの Validate メソッドは、単一のプロパティである Confdir 拡張プロパティを確認します。このプロパティは、DNS が正常に動作するために重要な DNS 構成ディレクトリを指します。

---

注-DNS が動作している間、構成ディレクトリは変更できないため、Confdir プロパティは RTR ファイルで TUNABLE = AT\_CREATION と宣言します。したがって、Validate メソッドが呼び出されるのは、更新の結果として Confdir プロパティを確認するためではなく、データサービスリソースが作成されているときだけです。

---

Confdir が、RGM が Validate に渡すプロパティの 1 つである場合、parse\_args() 関数はその値を取得し、保存します。Validate メソッドは、Confdir の新しい値が指すディレクトリがアクセス可能であるかどうか、および、named.conf ファイルがそのディレクトリ内に存在し、データを持っているかどうかを確認します。

parse\_args() 関数が、RGM から渡されたコマンド行引数から Confdir の値を取得できない場合でも、Validate は Confdir プロパティの妥当性を検査しようとします。まず、Validate メソッドは scha\_resource\_get() 関数を使用し、静的な構成から Confdir の値を取得します。次に、Validate は同じ検査を実行し、構成ディレクトリがアクセス可能であるかどうか、および、空でない named.conf ファイルがそのディレクトリ内に存在するかどうかを確認します。

Validate メソッドが失敗で終了した場合、Confdir だけでなく、すべてのプロパティの更新または作成が失敗します。

## Validate メソッドの構文解析関数

RGM は、ほかのコールバックメソッドとは異なる引数セットを Validate メソッドに渡すため、Validate メソッドには、ほかのメソッドとは異なる引数を構文解析する別の関数が必要です。Validate メソッドやほかのコールバックメソッドに渡される引数の詳細については、rt\_callbacks(1HA) のマニュアルページを参照してください。次のコードサンプルに、Validate メソッドの parse\_args() 関数を示します。

```
#####
# Validate 引数の構文解析。
#
function parse_args # [args...]
{
    typeset opt
    while getopts 'cur:x:g:R:T:G:' opt
    do
        case "$opt" in
            R)
                # DNS リソース名。
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されたリソース
                # グループ名。
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプ名。
                RESOURCETYPE_NAME=$OPTARG
                ;;
            r)
                # メソッドはシステム定義プロパティ
                # にアクセスしていない。したがって、このフラグは動作なし。
                ;;
            g)
                # メソッドはリソースグループプロパティに
                # アクセスしていない。したがって、このフラグは動作なし。
                ;;
            c)
                # Validate メソッドがリソースの作成中に
                # 呼び出されていることを示す。したがって、このフラグは動作なし。
                ;;
            u)
                # リソースがすでに存在しているときは、
                # プロパティの更新を示す。Confdir
                # プロパティを更新する場合、Confdir
                # がコマンド行引数に現れる。現れない場合、メソッドは
                # scha_resource_get を使用して Confdir を探す必要がある。
                UPDATE_PROPERTY=1
                ;;
            x)
                # 拡張プロパティのリスト。プロパティ
                # と値のペア。区切り文字は「=」
                PROPERTY='echo $OPTARG | awk -F= '{print $1}''
                VAL='echo $OPTARG | awk -F= '{print $2}''
        esac
    done
}
#####
```

```

# Confdir 拡張プロパティがコマンド行
# 上に存在する場合、その値を記録する。
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
}

```

ほかのメソッドの `parse_args()` 関数と同様に、この関数は、リソース名を取得するためのフラグ(R)、リソースグループ名を取得するためのフラグ(G)、RGM から渡されるリソースタイプを取得するためのフラグ(T)を提供します。

r フラグ(システム定義プロパティを示す)、g フラグ(リソースグループプロパティを示す)、c フラグ(リソースの作成中に妥当性の検査が行われていることを示す)は無視されます。これらのフラグが無視されるのは、このメソッドはリソースが更新されるときに拡張プロパティの妥当性を検査するために呼び出されるためです。

u フラグは、`UPDATE_PROPERTY` シェル変数の値を 1 (TRUE) に設定します。x フラグは、更新されているプロパティの名前と値を取得します。更新されているプロパティの中に `Confdir` が存在する場合、その値が `CONFDIR` シェル変数に格納され、`CONFDIR_FOUND` 変数が 1 (TRUE) に設定されます。

## Confdir の妥当性検査

`Validate` メソッドはまず、その `MAIN` 関数において、`CONFDIR` 変数を空の文字列に設定し、`UPDATE_PROPERTY` と `CONFDIR_FOUND` を 0 に設定します。

```

CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0

```

次に、`Validate` メソッドは `parse_args()` 関数を呼び出し、RGM から渡された引数を構文解析します。

```

parse_args "$@"

```

`Validate` は、`Validate` がプロパティの更新の結果として呼び出されているのかどうかを検査します。また `Validate` は、`Confdir` 拡張プロパティがコマンド行上に存

在するかどうかも検査します。次に、Validate メソッドは、Confdir プロパティが値を持っているかどうかを確認します。値を持っていない場合、Validate メソッドはエラーメッセージを記録し、失敗状態で終了します。

```
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

# Confdir プロパティが値を持っているかどうかを確認する。持っていない場合、状態 1 (失敗) で終了する。
if [[ -z $CONFDIR ]]; then
    logger -p ${SYSLOG_FACILITY}.err \
        "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
    exit 1
fi
```

---

注 - 具体的には、このコードは、Validate が更新 (\$UPDATE\_PROPERTY == 1) の結果として呼び出されているかどうかを検査し、プロパティがコマンド行上で「見つからなかった」かどうか (CONFDIR\_FOUND == 0) を検査します。この場合、コードは `scha_resource_get()` 関数を使用して Confdir の既存の値を取得します。コマンド行で Confdir が見つかった場合 (CONFDIR\_FOUND == 1)、CONFDIR の値は `scha_resource_get()` 関数からではなく `parse_args()` 関数から来ています。

---

Validate メソッドは CONFDIR の値を使用し、ディレクトリがアクセス可能であるかどうかを確認します。ディレクトリがアクセス可能ではない場合、Validate メソッドはエラーメッセージを記録し、エラー状態で終了します。

```
# $CONFDIR がアクセス可能であるかどうかを検査する。
if [ ! -d $CONFDIR ]; then
    logger -p ${SYSLOG_FACILITY}.err \
        -t [${SYSLOG_TAG}] \
        "${ARGV0} Directory $CONFDIR missing or not mounted"
    exit 1
fi
```

Confdir プロパティの更新の妥当性を検査する前に、Validate メソッドは最終検査を実行し、`named.conf` ファイルが存在するかどうかを確認します。ファイルが存在しない場合、Validate メソッドはエラーメッセージを記録し、エラー状態で終了します。

```
# named.conf ファイルが Confdir ディレクトリ内に存在するかどうかを検査する。
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

```

最終検査を通過した場合、Validate メソッドは、成功を示すメッセージを記録し、成功状態で終了します。

```

# Validate メソッドが成功したことを示すメッセージを記録する。
logger -p ${SYSLOG_FACILITY}.err \
    -t [${SYSLOG_TAG} \
    "${ARGV0} Validate method for resource "$RESOURCE_NAME \
    " completed successfully"

exit 0

```

## Validate の終了状態

Validate メソッドが成功 (0) で終了すると、新しい値を持つ Confdir が作成されます。Validate メソッドが失敗 (1) で終了すると、Confdir を含むすべてのプロパティが作成されず、理由を示すメッセージが生成されます。

## Update メソッドの仕組み

リソースのプロパティが変更されたとき、RGM は Update メソッドを実行し、動作中のリソースにその旨を通知します。RGM は、クラスタ管理者がリソースまたはそのグループのプロパティの設定に成功したあとに、Update を実行します。このメソッドは、リソースがオンラインであるノードまたはゾーン上で呼び出されます。

## Update メソッドの動作

Update メソッドはプロパティを更新しません。プロパティの更新は RGM が行います。Update メソッドは、更新が発生したことを動作中のプロセスに通知します。サンプルのデータサービスでは、プロパティの更新によって影響を受けるプロセスは障害モニターだけですが、障害モニタープロセスは、Update メソッドが停止および再起動するプロセスです。

Update メソッドは、障害モニターが動作していることを確認してから、pmfadm コマンドを使用して障害モニターを強制終了する必要があります。Update メソッドは、障害モニターを実装する検証プログラムの位置を取得し、pmfadm コマンドを使用して障害モニターを再起動します。

## Update による障害モニターの停止

Update メソッドは、`pmfadm -q` を使用し、障害モニターが動作していることを確認します。動作している場合、`pmfadm -s TERM` で障害モニターを強制終了します。障害モニターが正常に終了した場合、その影響を示すメッセージがクラスタ管理者に送信されます。障害モニターを停止できない場合、Update メソッドは、エラーメッセージをクラスタ管理者に送信し、失敗状態で終了します。

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

# すでに動作している障害モニターを強制終了する。
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
    # DNS の停止に成功。メッセージを記録する。
    logger -p ${SYSLOG_FACILITY}.err \
      -t [${RESOURCE_TYPE_NAME},${RESOURCEGROUP_NAME},${RESOURCE_NAME}] \
        "Monitor for HA-DNS successfully stopped"
  fi
fi
```

## 障害モニターの再起動

障害モニターを再起動するために、Update メソッドは検証プログラムを実装するスクリプトの位置を見つける必要があります。検証プログラムはデータサービスのベースディレクトリ (`RT_basedir` プロパティが指すディレクトリ) 内に存在します。Update は、次に示すように、`RT_basedir` の値を取得し、`RT_BASEDIR` 変数に格納します。

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

次に、Update は、`RT_BASEDIR` の値を `pmfadm` で使用し、`dns_probe` プログラムを再起動します。検証プログラムを再起動できた場合、Update メソッドはその影響を示すメッセージをクラスタ管理者に送信し、成功状態で終了します。`pmfadm` が検証プログラムを再起動できない場合、Update メソッドはエラーメッセージを記録し、失敗状態で終了します。

## Update の終了状態

Update メソッドが失敗すると、リソースが“update failed”(更新失敗)の状態になります。この状態はRGMのリソース管理に影響しません。しかし、`syslog()` 関数を通じて、管理ツールへの更新アクションが失敗したことを示します。



# Data Service Development Library

---

この章では、Data Services Development Library (DSDL) を形成するアプリケーションプログラミングインタフェースの概要を説明します。DSDL は `libdsdev.so` ライブラリとして実装されており、Sun Cluster パッケージに含まれています。

この章の内容は次のとおりです。

- 127 ページの「DSDL の概要」
- 128 ページの「構成プロパティの管理」
- 129 ページの「データサービスの起動と停止」
- 129 ページの「障害モニターの実装」
- 130 ページの「ネットワークアドレス情報へのアクセス」
- 130 ページの「実装したリソースタイプのデバッグ」
- 131 ページの「高可用性ローカルファイルシステムの有効化」

## DSDL の概要

DSDL API は、RMAPI の最上位の階層を形成します。そのため、DSDL API は RMAPI の代わりになるものではなく、RMAPI の機能をカプセル化および拡張するためのものです。DSDL は、特定の Sun Cluster 統合問題に対する事前定義されたソリューションを提供することによって、データサービスの開発を簡素化します。その結果、アプリケーションに本来求められている高可用性とスケーラビリティの問題に、より多くの開発時間を割くことが可能になります。また、アプリケーションの起動、シャットダウン、および監視機能を Sun Cluster に統合する際に、多くの時間を費やすこともありません。

## 構成プロパティの管理

すべてのコールバックメソッドは構成プロパティにアクセスする必要があります。

DSDL は、次の手段により、プロパティへのアクセスをサポートします。

- 環境の初期化
- プロパティ値を簡単に取得できる関数セットの提供

`scds_initialize()` 関数 (各コールバックメソッドの開始時に呼び出す必要がある) は、次の処理を行います。

- RGM がコールバックメソッドに渡すコマンド行引数 (`argc` と `argv[]`) を検査および処理します。そのため、コマンド行解析関数を作成する必要はありません。
- ほかの DSDL 関数ができるように内部データ構造を設定します。たとえば、DSDL で提供されている関数によって RGM から取得されたプロパティ値はこのデータ構造に格納されます。同様に、コマンド行から入力された値 (RGM から取得された値よりも優先される) もこのデータ構造に格納されます。
- 関数はロギング環境を初期化して、障害モニターの検証設定の妥当性を検査します。

---

注 - `Validate` メソッドの場合、`scds_initialize()` はコマンド行で渡されたプロパティ値を解析します。そのため、`Validate` 用の解析関数を作成する必要はありません。

---

DSDL は、リソース、リソースタイプ、リソースグループのプロパティ、および、よく使用される拡張プロパティを取得するための関数セットを提供します。

これらの関数は、次のような規則を使用して、プロパティへのアクセスを標準化しています。

- 各関数は、`scds_initialize()` から戻されるハンドル引数だけを取ります。
- 各関数が特定のプロパティに対応します。関数の戻り値のタイプは取得するプロパティ値のタイプに一致します。
- 値は `scds_initialize()` によってあらかじめ算出されているため、関数はエラーを戻しません。新しい値がコマンド行で渡された場合を除き、関数は RGM から値を取得します。

## データサービスの起動と停止

Start メソッドは、クラスタノードまたはゾーン上でデータサービスを起動するために必要なアクションを実行します。通常、このようなアクションには、リソースプロパティの取得、アプリケーション固有の実行可能ファイルおよび構成ファイルの格納先の特定、および適切なコマンド行引数を用いたアプリケーションの起動が含まれます。

`scds_initialize()` 関数はリソース構成を取得します。Start メソッドはプロパティ用の DSDL 関数を使用して、アプリケーションを起動するのに必要な構成ディレクトリや構成ファイルを識別するための特定のプロパティ (`Confdir_list` など) の値を取得します。

Start メソッドは、`scds_pmf_start()` を呼び出して、プロセス監視機能 (PMF) の制御下でアプリケーションを起動します。PMF を使用すると、プロセスに適用する監視レベルを指定したり、異常終了したプロセスを再起動したりできます。DSDL で実装する Start メソッドの例については、148 ページの「[xfnts\\_start メソッド](#)」を参照してください。

Stop メソッドは呼び出し回数に依存しないように実装されていなければなりません。つまり Stop メソッドは、アプリケーションが動作していないときにノードまたはゾーン上で呼び出された場合でも、正常終了する必要があります。Stop メソッドが失敗した場合、停止するリソースが `STOP_FAILED` 状態に設定され、クラスタのハードウェア再起動を招いてしまう可能性があります。

リソースが `STOP_FAILED` 状態になるのを防止するために、Stop メソッドはあらゆる手段を構じてリソースを停止する必要があります。`scds_pmf_stop()` 関数は、段階的にリソースを停止しようとします。この関数はまず、`SIGTERM` シグナルを使用してリソースを停止しようとします。これに失敗した場合は、`SIGKILL` シグナルを使用します。詳細は、`scds_pmf_stop(3HA)` のマニュアルページを参照してください。

## 障害モニターの実装

DSDL は、事前に定義されたモデルを提供することによって、障害モニターを実装する際の煩雑さをほとんど取り除きます。リソースがノードまたはゾーン上で起動すると、`Monitor_start` メソッドは PMF の制御下で障害モニターを起動します。リソースがノードまたはゾーン上で動作している間、障害モニターは無限ループを実行します。

次に、DSDL 障害モニターのロジックの概要を示します。

- `scds_fm_sleep()` 関数は `Thorough_probe_interval` プロパティを使用して、検証を行う期間を決定します。この期間中に PMF がアプリケーションプロセスの失敗を検出した場合、リソースは再起動されます。
- 検証機能自身は、障害の重要度を示す値を戻し、この値の範囲は、0 (障害なし) から 100 (致命的な障害) までです。
- 検証機能が戻した値は、`scds_action()` 関数に送信されます。`scds_action()` 関数は、`Retry_interval` プロパティの期間中に、障害の履歴を累積します。
- `scds_action()` 関数は、次に示すように、障害が発生した場合の処置を決定します。
  - 累積した障害が 100 より少ない場合は、何もしません。
  - 累積した障害が 100 に到達した場合 (完全な障害)、データサービスを再起動します。`Retry_interval` を超えた場合、障害の履歴をリセットします。
  - `Retry_interval` で指定された期間中に、再起動の回数が `Retry_count` プロパティを上回った場合、データサービスをフェイルオーバーします。

## ネットワークアドレス情報へのアクセス

DSDL は、リソースおよびリソースグループのネットワークアドレス情報を戻す関数を提供します。たとえば、`scds_get_netaddr_list()` は、リソースが使用するネットワークアドレスリソースを取得して、障害モニターがアプリケーションを検証できるようにします。

また、DSDL は TCP ベースの監視を行う関数セットも提供します。通常、このような関数はサービスへの単純なソケット接続を確立し、サービスのデータを読み書きしたあとで、サービスから切断します。検証の結果を DSDL の `scds_fm_action()` 関数に送信し、次に実行すべき処理を決定できます。

TCP ベースの障害監視の例については、163 ページの「`xfnts_validate` メソッド」を参照してください。

## 実装したリソースタイプのデバッグ

DSDL は、データサービスをデバッグするときに役立つ組み込み機能を提供します。

DSDL の `scds_syslog_debug()` ユーティリティは、実装したリソースタイプにデバッグ文を追加するための基本的なフレームワークを提供します。デバッグレベル (1 から 9 までの数字) は、各クラスターノードまたはゾーン上のリソースタイプの実装ごとに動的に設定できます。ファイル `/var/cluster/rgm/rt/rtname/loglevel` は、1 から 9 までの整数だけが含まれているファイルであり、すべてのリソースタイプ

コールバックメソッドはこのファイルを読み取ります。DSDL の `scds_initialize()` 関数はこのファイルを読み取って、内部デバッグレベルを指定されたレベルに設定します。デフォルトのデバッグレベルは0であり、この場合、データサービスはデバッグメッセージを記録しません。

`scds_syslog_debug()` 関数は、`LOG_DEBUG` の優先順位において、`scha_cluster_getlogfacility()` 関数から戻された機能を使用します。このようなデバッグメッセージは `/etc/syslog.conf` ファイルで構成できます。

`scds_syslog()` 関数を使用すると、いくつかのデバッグメッセージをリソースタイプの通常の動作(おそらくは `LOG_INFO` 優先順位)における情報メッセージとして使用することができます。第8章のサンプル DSDL アプリケーションでは、`scds_syslog_debug()` および `scds_syslog()` 関数が使用されています。

## 高可用性ローカルファイルシステムの有効化

HASStoragePlus リソースタイプを使用すると、ローカルファイルシステムを SunCluster 環境内で高可用性にすることができます。

---

注 - ローカルファイルシステムには、UNIX File System (UFS)、Quick File System (QFS)、Veritas File System (VxFS)、Solaris ZFS (Zettabyte File System) などがあります。

---

このためには、ローカルファイルシステムのパーティションをグローバルディスクグループ内に配置しなければなりません。また、アフィニティスイッチオーバーを有効にし、Sun Cluster 環境をフェイルオーバー用に構成する必要があります。この設定によって、クラスタ管理者は、多重ホストディスクに直接接続された任意のホストから、多重ホストディスク上の任意のファイルシステムにアクセスできるようになります。I/O に負荷が集中する、一部のデータサービスに対しては、高可用性ローカルファイルシステムを使用します。HASStoragePlus リソースタイプの構成については、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』の「高可用性ローカルファイルシステムの有効化」を参照してください。



## リソースタイプの設計

---

この章では、リソースタイプの設計や実装で Data Service Development Library (DSDL) を通常どのように使用するかについて説明します。また、リソース構成を検証したり、リソースの開始、停止、および監視を行なったりするためのリソースタイプの設計についても説明します。さらに、リソースタイプのコールバックメソッドを DSDL を使って実装する方法を説明します。

詳細は、`rt_callbacks(1HA)` のマニュアルページを参照してください。

これらの作業を行うには、リソースのプロパティ設定値にアクセスできなければなりません。DSDL ユーティリティー `scds_initialize()` を使用すると、統一された方法でリソースプロパティにアクセスできます。この機能は、各コールバックメソッドの始めの部分で呼び出す必要があります。このユーティリティー関数は、クラスタフレームワークからリソースのすべてのプロパティを取り出し、これによって、そのリソースは、`scds_getname()` 関数群から利用できるようになります。

この章の内容は次のとおりです。

- 134 ページの「リソースタイプ登録ファイル」
- 134 ページの「Validate メソッド」
- 136 ページの「Start メソッド」
- 137 ページの「Stop メソッド」
- 139 ページの「Monitor\_start メソッド」
- 139 ページの「Monitor\_stop メソッド」
- 140 ページの「Monitor\_check メソッド」
- 140 ページの「Update メソッド」
- 141 ページの「Init、Fini、Boot の各メソッドの説明」
- 142 ページの「障害モニターデーモンの設計」

# リソースタイプ登録ファイル

RTR (Resource Type Registration、リソースタイプ登録) ファイルは、Sun Cluster ソフトウェアに対して、リソースタイプの詳細な情報を指定します。

詳細情報には次の情報が含まれます。

- 実装に必要なプロパティ
- これらのプロパティのデータタイプやデフォルト値
- リソースタイプの実装用のコールバックメソッドのファイルシステムパス
- システム定義プロパティのさまざまな設定

ほとんどのリソースタイプ実装では、DSDL に添付されるサンプル RTR ファイルで十分なはずですが、必要な作業は、リソースタイプ名、リソースタイプのコールバックメソッドのパス名などの基本的な要素の編集だけです。リソースタイプを実装する際に新しいプロパティが必要な場合は、そのプロパティをリソースタイプ実装の RTR ファイルで拡張プロパティとして宣言します。新しいプロパティのアクセスには DSDL `scds_get_ext_property()` ユーティリティを使用します。

## Validate メソッド

リソースタイプ実装の `Validate` コールバックメソッドの目的は、リソースに対する新しいプロパティ設定により指定されるリソースの新しい設定値が、そのリソースタイプにとって有効であるかどうかを検査することにあります。

リソースタイプ実装の `Validate` メソッドは、次のどちらかの条件のときに RGM によって呼び出されます。

- このリソースタイプの新規リソースが作成されつつある。
- このリソースまたはリソースグループのプロパティが更新されつつある。

この2つの操作は、リソースの `Validate` メソッドに渡されるコマンド行オプション `-c` (作成) と `-u` (更新) の存在によって区別されます。

`Validate` メソッドはノード群の各ノードまたは各ゾーンに対して呼び出されます。ノード群またはゾーンは、リソースタイププロパティ `Init_nodes` の値で定義されます。`Init_nodes` が `RG_PRIMARYES` に設定されている場合、`Validate` は、そのリソースを含むリソースグループを収容できる (その主ノードになりうる) 各ノードまたはゾーンに対して呼び出されます。`Init_nodes` が `RT_INSTALLED_NODES` に設定されている場合、`Validate` は、リソースタイプソフトウェアがインストールされている各ノードまたはゾーン (通常は、クラスタのすべてのノードまたはゾーン) に対して呼び出されます。

`Init_nodes` のデフォルト値は `RG_PRIMARYES` です (`rt_reg(4)` のマニュアルページを参照)。`Validate` メソッドが呼び出される時点では、RGM はまだリソースを作成していません (作成コールバックの場合)。あるいは、更新するプロパティの更新値をまだ適用していません (更新コールバックの場合)。

注-HAStoragePlus リソースタイプによって管理されるローカルファイルシステムを使用している場合は、`scds_hasp_check()` 関数を使ってそのリソースタイプの状態を検査します。当該リソース用に定義されている `Resource_dependencies` または `Resource_dependencies_weak` のシステム属性を使用することによって、当該リソースが依存しているすべての SUNW.HAStoragePlus リソース状態 (オンラインであるか、オンラインでないか) についての情報が得られます。`scds_hasp_check()` 関数から返される状態コードの完全なリストについては、`scds_hasp_check(3HA)` のマニュアルページを参照してください。

DSDL 関数 `scds_initialize()` は、リソースの作成や更新を次のように処理します。

- リソースの作成では、`scds_initialize()` はコマンド行で渡された新しいリソースプロパティを解析します。これによって、リソースプロパティの新しい値を、そのリソースがすでにシステムで作成されているかのように使用できます。
- リソースやリソースグループの更新では、クラスタ管理者によって更新されているプロパティの新しい値は、コマンド行から読み込まれます。残りのプロパティ (値が更新されないもの) は、RMAPI を使って Sun Cluster から読み込みます。DSDL を使用する場合は、このような作業を考慮する必要はありません。開発者は、リソースのすべてのプロパティが使用可能であるものとして、リソースの検証を行うことができます。

リソースのプロパティの検証を実装する関数は `svc_validate()` と呼ばれます。この関数は、`scds_get_name()` 関数群を使って、検証しようとするプロパティを検査します。リソースの設定が有効ならこの関数から戻りコード 0 が返されるとすると、リソースタイプの Validate メソッドは、次のコード部分のようになります。

```
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);
}
```

さらに検証関数は、リソースの検証が失敗した理由を記録する必要もあります。ただし、詳細は省略することによって、次に示すように、より単純な例である `svc_validate()` 関数を実装できます (第 8 章には検証関数の実際的な取り扱いが記載されています)。

```
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); /* 無効なリソースプロパティ設定 */
    }
    return (0); /* 有効な設定 */
}
```

このように、リソースタイプの開発者は、`svc_validate()` 関数を実装することだけに集中できます。

## Start メソッド

リソースタイプ実装の Start コールバックメソッドは、特定のクラスタノードまたはゾーンのリソースを開始するときに RGM によって呼び出されます。リソースグループ名とリソース名、およびリソースタイプ名はコマンド行から渡されます。Start メソッドは、クラスタノードまたはゾーンでデータサービスリソースを開始するために必要なアクションを行います。通常、このようなアクションには、リソースプロパティの取得、アプリケーション固有の実行可能ファイルと構成ファイルの一方または両方の格納先の特定、および適切なコマンド行引数を用いたアプリケーションの起動が含まれます。

DSDL では、リソース構成ファイルが `scds_initialize()` ユーティリティによってすでに取得されています。アプリケーションの起動アクションは、`svc_start()` 関数に指定できます。さらに、アプリケーションが実際に起動されたかどうかを確認するために、`svc_wait()` 関数を呼び出すことができます。Start メソッドのコード (詳細は省略) は、次のようになります。

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

    if (scds_initialize(&handle, argc, argv) != SCHA_ERR_NOERR) {
        return (1); /* 初期化エラー */
    }
    if (svc_validate(handle) != 0) {
        return (1); /* 無効な設定 */
    }
    if (svc_start(handle) != 0) {
        return (1); /* 起動に失敗 */
    }
}
```

```
    return (svc_wait(handle));  
}
```

この起動メソッドの実装では、`svc_validate()` を呼び出してリソース構成を検証します。検証が失敗する場合は、リソース構成とアプリケーション構成が一致していないか、現在このクラスタノードまたはゾーンのシステムに関して何らかの問題があることを示しています。たとえば、リソースに必要なクラスタファイルシステムが、現在このクラスタノードまたはゾーンで使用できない可能性などが考えられます。その場合には、このクラスタノードまたはゾーンでこのリソースを起動しても意味がないので、RGM を使って別のノードまたはゾーンのリソースを起動すべきです。

ただし、上記の文では `svc_validate()` が十分に限定的であり、アプリケーションにより絶対に必要なリソースがあるかどうかをそのクラスタノードまたはゾーンだけで検査することに注意してください。そうでないと、このリソースはすべてのクラスタノードまたはゾーンで起動に失敗し、`START_FAILED` の状態になる可能性があります。この状態の詳細は、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』を参照してください。

`svc_start()` 関数は、このノードまたはゾーンでリソースの起動に成功した場合は戻りコード 0 を、問題を検出した場合は 0 以外の戻りコードをそれぞれ返す必要があります。この関数から 0 以外の値が返されると、RGM は、このリソースを別のクラスタノードまたはゾーンで起動しようと試みます。

DSDL を最大限に活用するには、`svc_start()` 関数で `scds_pmf_start()` ユーティリティを呼び出して、アプリケーションを PMF (プロセス管理機能) のもとで起動できます。このユーティリティは、PMF の障害コールバックアクション機能を使って、プロセス障害を検出します。詳細については、`pmfadm(1M)` マニュアルページの `-a` アクション引数の説明を参照してください。

## Stop メソッド

リソースタイプ実装の Stop コールバックメソッドは、特定のクラスタノードまたはゾーンでアプリケーションを停止するときに RGM によって呼び出されます。

Stop メソッドのコールバックが有効であるためには、次の条件が必要です。

- Stop メソッドは結果に依存しない命令 (idempotent) でなければなりません。つまり、Stop メソッドは、そのノードまたはゾーンで Start メソッドが正常に終了していなくても、RGM から呼び出されることがあります。したがって、Stop メソッドは、そのクラスタノードまたはゾーンでアプリケーションが動作していない場合でも (したがって、特別な処理が必要ない場合でも)、正常に (終了コード 0 で) 終了しなければなりません。
- リソースタイプの Stop メソッドが、あるクラスタノードまたはゾーンで失敗に終わると (0 以外で終了)、停止中のリソースは STOP\_FAILED の状態になります。リソースの Failover\_mode 設定によっては、この条件により、クラスタノードが RGM によってハードウェア的に再起動されることがあります。

したがって、Stop メソッドの設計時には、このメソッドがアプリケーションを明示的に停止する手段が必要です。アプリケーションが停止しない場合は、SIGKILL などを使って、アプリケーションを強制的かつ即時に停止する必要があります。

さらに、このメソッドによるアプリケーションの停止は一定の時間内に行われなければなりません。Stop\_timeout プロパティで設定した時間が経過すると、停止が失敗したものとみなされ、リソースは STOP\_FAILED の状態になるからです。

ほとんどのアプリケーションには、DSDL ユーティリティー `scds_pmf_stop()` で十分なはずです。これは、アプリケーションを SIGTERM で「静かに」停止しようとするためです。続いてこの関数は、プロセスに対して SIGKILL を適用します。この関数は、まず、アプリケーションが PMF の `scds_pmf_start()` で起動されたものとみなします。このユーティリティーの詳細については、[219 ページの「PMF 関数」](#)を参照してください。

アプリケーションを停止するそのアプリケーション固有の関数を `svc_stop()` とするのなら、Stop メソッドは、次のように実装します。

```
if (scds_initialize(&handle, argc, argv) != SCHA_ERR_NOERR)
{
    return (1); /* 初期化エラー */
}
return (svc_stop(handle));
```

前述の `svc_stop()` 関数の実装に `scds_pmf_stop()` 関数が含まれているかどうかは、ここでは関係ありません。`scds_pmf_stop()` 関数を含めるかの決定は、アプリケーションが PMF のもとで Start メソッドによって起動されているかどうかに依存します。

Stop メソッドの実装では、`svc_validate()` メソッドは使用されません。システムに問題があったとしても、Stop メソッドは、このノードまたはゾーンでこのアプリケーションを Stop すべきだからです。

## Monitor\_start メソッド

RGMは、リソースの障害モニターを起動する場合に `Monitor_start` メソッドを呼び出します。障害モニターは、このリソースによって管理されているアプリケーションの状態を監視します。リソースタイプの実装では、通常、障害モニターはバックグラウンドで動作する独立したデーモンとして実装されます。このデーモンの起動には、適切な引数をもつ `Monitor_start` コールバックメソッドが使用されます。

モニターデーモン自体は障害が発生しやすいため(たとえばモニターは、アプリケーションを監視されない状態にしたまま、停止することがある)、モニターデーモンは、PMFを使って起動すべきです。DSDLユーティリティー `scds_pmf_start()` には、障害モニターを起動する機能が組み込まれています。このユーティリティーは、モニターデーモンプログラムのリソースタイプコールバックメソッド実装の場所を表す `RT_basedir` からの相対パス名を使用します。このユーティリティーは、DSDLによって管理される `Monitor_retry_interval` 拡張プロパティーと `Monitor_retry_count` 拡張プロパティーを使って、デーモンが際限なく再起動されるのを防止します。

このユーティリティーでは、モニターデーモンのコマンド行構文には、すべてのコールバックメソッドに対して定義されたコマンド行構文と同じものが使用されます (`-R resource -G resource-group -T resource-type`) が、モニターデーモンがRGMから直接呼び出されることは決してありません。このユーティリティーでは、モニターデーモン実装自体が `scds_initialize()` ユーティリティーで独自の環境を設定できます。したがって、主な作業は、モニターデーモン自体を設計することです。

## Monitor\_stop メソッド

RGMは、`Monitor_start` メソッドで起動された障害モニターデーモンを停止するために、`Monitor_stop` メソッドを呼び出します。このコールバックメソッドの失敗は、`Stop` メソッドの失敗とまったく同じように処理されます。したがって、`Monitor_stop` メソッドは、`Stop` メソッドと同じように強固なものでなければなりません。

障害モニターデーモンを `scds_pmf_start()` ユーティリティーを使って起動したら、`scds_pmf_stop()` ユーティリティーで停止する必要があります。

## Monitor\_check メソッド

RGM は、指定されたリソースについて、クラスタノードまたはゾーンがリソースをマスターする能力を持っているかどうかを確認するために、そのノードまたはゾーンのリソースに対して `Monitor_check` コールバックメソッドを実行します。つまり、RGM は、そのリソースによって管理されるアプリケーションがそのノードまたはゾーンで正常に動作するかどうかを判別するためにこのメソッドを実行します。

通常、この状況では、アプリケーションに必要なすべてのシステムリソースが本当にクラスタノードまたはゾーンで使用可能かどうかを確認されます。[134 ページ](#)の「`Validate` メソッド」で説明されているように、開発者が実装する `svc_validate()` 関数は、少なくともこの確認が行われなければなりません。

リソースタイプ実装によって管理されている特定のアプリケーションによっては、`Monitor_check` メソッドでそのほかの作業を行うことがあります。`Monitor_check` メソッドは、並行して実行中のそのほかのメソッドと競合しない方法で実装する必要があります。DSDL を使用する場合には、リソースプロパティに対するアプリケーション固有の検証を実装する `svc_validate()` 関数を `Monitor_check` メソッドで呼び出す必要があります。

## Update メソッド

RGM は、リソースタイプ実装の `Update` メソッドを呼び出して、クラスタ管理者が行なったすべての変更をアクティブリソースの構成に適用します。`Update` メソッドは、そのリソースがオンラインになっているすべてのノードまたはゾーンに対して呼び出されます。

リソースの構成に対して行われた変更は、リソースタイプ実装にとって必ず有効なものです。RGM は、リソースタイプの `Update` メソッドを呼び出す前に `Validate` メソッドを呼び出すからです。`Validate` メソッドは、リソースやリソースグループのプロパティが変更される前に呼び出されます。したがって、`Validate` メソッドは新しい変更を拒否できます。変更が適用されると、`Update` メソッドが呼び出され、新しい設定値がアクティブ (オンライン) リソースに通知されます。

リソースタイプの開発者は、どのプロパティを動的に変更できるようにするかを慎重に決定し、RTR ファイルでこれらのプロパティに `TUNABLE = ANYTIME` を設定する必要があります。通常、障害モニターデーモンによって使用されるリソースタイプ実装のプロパティは、すべて動的に更新できるように指定できます。ただし、`Update` メソッドの実装は、少なくともモニターデーモンを再起動できなければなりません。

使用できるプロパティの候補には次のものがあります。

- `Thorough_probe_interval`
- `Retry_count`

- `Retry_interval`
- `Monitor_retry_count`
- `Monitor_retry_interval`
- `Probe_timeout`

これらのプロパティーは、障害モニターデーモンがサービスの状態をどのようにチェックするかや、デーモンがチェックをどのような頻度で行うか、エラーをデーモンがどのような履歴間隔を使用して追跡管理するか、あるいは、PMF がどのような再起動しきい値を設定するかに影響を及ぼします。DSDLには、これらのプロパティーの更新を行うための `scds_pmf_restart()` ユーティリティーが備わっています。

リソースプロパティーを動的に更新できなければならないが、プロパティーの変更によって動作中のアプリケーションに影響が及ぶ可能性がある場合は、適切なアクションを行なう必要があります。プロパティーに対する更新が動作中のアプリケーションインスタンスに正しく適用されるようにしなければなりません。現在のところ、DSDLを使用してこのようにリソースプロパティーを動的に更新することはできません。変更されたプロパティーをコマンド行で `Update` に渡すことはできません (`Validate` では可能)。

## Init、Fini、Boot の各メソッドの説明

これらのメソッドは、リソース管理 API 仕様の定義による「一度だけのアクション」を行うためのものです。DSDL のサンプル実装には、これらのメソッドの使い方は示されていません。しかし、これらのメソッドを使用する必要がある場合には、DSDL のすべての機能をこれらのメソッドでも使用できます。通常、「一度だけのアクション」を使用するリソースタイプ実装では、`Init` メソッドと `Boot` メソッドはまったく同じように機能します。`Fini` メソッドは、一般に、`Init` メソッドや `Boot` メソッドのアクションを「取り消す」ためのアクションを実行します。

## 障害モニターデーモンの設計

DSDL を使用したリソースタイプ実装には、通常、次の役割を実行する障害モニターデーモンがあります。

- 管理されているアプリケーションの状態を定期的に監視します。モニターデーモンのこの役割は特定のアプリケーションに大きく依存し、リソースタイプによって大幅に異なることがあります。DSDL には、TCP に基づく簡単なサービスの状態を検査するいくつかのユーティリティー関数が組み込まれています。HTTP、NNTP、IMAP、POP3 など、ASCII ベースのプロトコルを使用するアプリケーションは、これらのユーティリティーを使って実装できます。
- アプリケーションによって検出された問題を、リソースプロパティ `Retry_interval` や `Retry_count` を使って追跡します。さらに、アプリケーションが完全に異常停止した場合、障害モニターは、PMF アクションスクリプトを使ってサービスを再起動すべきかどうかや、アプリケーションの障害が急速に蓄積されるためにフェイルオーバーを実行する必要があるかどうかを判断する必要があります。DSDL のユーティリティー `scds_fm_action()` と `scds_fm_sleep()` は、この機構の実装を助けることを目的としています。
- 一般に、アプリケーションを再起動するか、リソースを含むリソースグループのフェイルオーバーを試みるなど、適切なアクションを実行します。DSDL ユーティリティー `scds_fm_action()` には、このアルゴリズムが実装されています。このユーティリティーは、この目的のために、過去の `Retry_interval` 秒数の間に起った検証障害の現在の累積を計算します。
- リソースの状態を更新します。これによって、Sun Cluster の管理コマンドやクラスタ管理 GUI からアプリケーションの状態を知ることができます。

DSDL ユーティリティーの設計では、障害モニターデーモンの主要ループは、この節の最後にある擬似コードで表すことができます。

DSDL を使用して障害モニターを実装する際には、次の点に注意してください。

- アプリケーションプロセスの異常停止は、`scds_fm_sleep()` によって迅速に検出されます。これは、PMF によるアプリケーションプロセス停止の通知が非同期に行われるためです。そのため、障害の検出時間が大幅に短くなり、サービスの可用性が高くなります。別の方法としては、障害モニターが頻繁にスリープから復帰してサービスの状態を検査し、アプリケーションプロセスの停止を検出する方法があります。
- RGM が `scha_control` API によるサービスのフェイルオーバーを拒否すると、`scds_fm_action()` は、現在の障害履歴を「リセット」(消去)します。この関数が現在の障害履歴をリセットするのは、障害履歴が `Retry_count` の値をすでに超えているからです。モニターデーモンは、次のサイクルでスリープから復帰したあとに、デーモンの状態検査を正常に完了できないと、`scha_control()` 関数を再び呼び出そうとします。しかし、前回のサイクルで呼び出しが拒否され状況が依然として残っていれば、この呼び出しは今回も拒否されるはずで、履歴がリセットされていれば、障害モニターは、少なくとも、次のサイクルでアプリケーションの再起動などによってその状況を内部的に訂正しようとしています。
- 再起動が失敗に終わった場合、`scds_fm_action()` は、アプリケーション障害履歴をリセットしません。これは、状況が訂正されなければ、`scha_control()` が間もなく呼び出される可能性が高いからです。
- ユーティリティー `scds_fm_action()` は、障害履歴に従って、リソースステータスを `SCHA_RSSTATUS_OK`、`SCHA_RSSTATUS_DEGRADED`、`SCHA_RSSTATUS_FAULTED` のどれかに更新します。その結果、このステータスをクラスタシステム管理から使用できるようになります。

ほとんどの場合、アプリケーション固有の状態検査アクションは、スタンドアロンの別個のユーティリティー(たとえば、`svc_probe()`)に実装できます。これは、次の汎用的なメインループに統合できます。

```
for (;;) {
    /* 正常な検証と検証の間の thorough_probe_interval
     * だけスリープする。
     */
    (void) scds_fm_sleep(scds_handle,
        scds_get_rs_thorough_probe_interval(scds_handle));
    /* 使用するすべての ipaddress を検証する。次の各要素を繰り返し検証する。
     * 1. 使用するすべてのネットリソース
     * 2. 特定のリソースのすべての ipaddresses
     * 検証する ipaddress ごとに
     * 障害履歴を計算する。
     */
    probe_result = 0;
    /* すべてのリソースを繰り返し調べて、
     * svc_probe() の呼び出しに使用する各 IP アドレスを取得する。
     */
}
```

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
/* 状態を検証する必要があるホスト名とポート
* を取得する。
*/
hostname = netaddr->netaddrs[ip].hostname;
port = netaddr->netaddrs[ip].port_proto.port;
/*
* HA-XFS は、1 つのポートしかサポートしないため
* ポート配列の最初のエントリから
* ポート値を取得する。
*/
ht1 = gethrtime();
/* Latch probe start time */
probe_result = svc_probe(scds_handle, hostname, port, timeout);
/*
* サービス検証履歴を更新し、
* 必要に応じてアクションを実行する。
* 検証終了時刻を保存する。
*/
ht2 = gethrtime();
/* ミリ秒に変換する。 */
dt = (ulong_t)((ht2 - ht1) / 1e6);
/*
* 障害履歴を計算し、
* 必要に応じてアクションを実行する。
*/
(void) scds_fm_action(scds_handle,
probe_result, (long)dt);
} /* 各ネットワークリソース */
} /* 検証を続ける */
```

## サンプル DSDL リソースタイプの実装

---

この章では、DSDL で実装したサンプルのリソースタイプ `SUNW.xfnts` について説明します。データサービスは C 言語で作成されています。使用するアプリケーションは TCP/IP ベースのサービスである X Font Server です。付録 C では、`SUNW.xfnts` リソースタイプにおける各メソッドの完全なコードを示します。

この章の内容は次のとおりです。

- 145 ページの「X Font Server について」
- 147 ページの「`SUNW.xfnts` の RTR ファイル」
- 147 ページの「関数とコールバックメソッドの命名規則」
- 148 ページの「`scds_initialize()` 関数」
- 148 ページの「`xfnts_start` メソッド」
- 153 ページの「`xfnts_stop` メソッド」
- 154 ページの「`xfnts_monitor_start` メソッド」
- 155 ページの「`xfnts_monitor_stop` メソッド」
- 156 ページの「`xfnts_monitor_check` メソッド」
- 157 ページの「`SUNW.xfnts` 障害モニター」
- 163 ページの「`xfnts_validate` メソッド」
- 166 ページの「`xfnts_update` メソッド」

### X Font Server について

X Font Server は、フォントファイルをクライアントに提供する TCP/IP ベースのサービスです。クライアントはサーバーに接続してフォントセットを要求します。サーバーはフォントファイルをディスクから読み取って、クライアントにサービスを提供します。X Font Server デーモンは、`/usr/openwin/bin/xfns` にあるサーバーバイナリから構成されます。このデーモンは通常、`inetd` から起動されます。ただし、このサンプルでは、`/etc/inetd.conf` ファイル内の適切なエントリが (たとえば、`fsadmin -d` コマンドを使用することで) 無効にされているものと想定しています。したがって、デーモンは Sun Cluster ソフトウェアだけの制御下にあります。

## X Font Server の構成ファイル

デフォルトでは、X Font Server は自身の構成情報をファイル `/usr/openwin/lib/X11/fontserver.cfg` から読み取ります。このファイルのカタログエントリには、デーモンがサービスを提供できるフォントディレクトリのリストが入っています。クラスタ管理者は、クラスタファイルシステム上のフォントディレクトリを指定できます。このような配置により、システム上でフォントデータベースのコピーを1つだけ保持すれば済むので、Sun Cluster 上の X Font Server の使用を最適化できます。クラスタ管理者が位置を変更する場合は、`fontserver.cfg` を編集して、フォントディレクトリの新しいパスを反映させる必要があります。

構成を簡単にするために、クラスタ管理者は構成ファイル自身もクラスタファイルシステム上に配置できます。`xf`s デーモンはデフォルトの格納先(このファイルの組み込み場所)を変更するコマンド行引数を提供します。SUNW.xfnts リソースタイプは、次のコマンドを使用して、Sun Cluster ソフトウェアの制御下でデーモンを起動します。

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

SUNW.xfnts リソースタイプの実装では、`Confdir_list` プロパティを使用して、`fontserver.cfg` 構成ファイルの格納場所を管理できます。

## TCP ポート番号

`xf`s サーバーデーモンが待機している TCP ポート番号は、通常は「fs」ポートであり、`/etc/services` ファイルの中で 7100 と定義されているのが普通です。ただし、`xf`s コマンドでクラスタ管理者が含める `-port` オプションにより、システム管理者はデフォルトの設定を変更できます。

SUNW.xfnts リソースタイプの `Port_list` プロパティを使用すると、デフォルト値を設定したり、クラスタ管理者が `xf`s コマンドと `-port` オプションを指定できるようになります。RTR ファイルにおいて、このプロパティのデフォルト値を `7100/tcp` と定義します。SUNW.xfnts の Start メソッドで、`Port_list` を `xf`s コマンド行の `-port` オプションに渡します。その結果、このリソースタイプのユーザーはポート番号を指定する必要がなくなります(ポートのデフォルト値は `7100/tcp`)。クラスタ管理者は、リソースタイプを構成するときには、`Port_list` プロパティに異なる値を指定できます。

## SUNW.xfnts の RTR ファイル

ここでは、SUNW.xfnts RTR ファイル内のいくつかの重要なプロパティについて説明します。各プロパティの目的については説明しません。プロパティの詳細については、34 ページの「リソースとリソースタイププロパティの設定」を参照してください。

次に示すように、Confdir\_list 拡張プロパティは構成ディレクトリ (または、ディレクトリのリスト) を指定します。

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

Confdir\_list プロパティには、デフォルト値は設定されていません。クラスタ管理者はリソースを作成するときに、ディレクトリ名を指定する必要があります。Tunable 属性が AT\_CREATION に制限されているため、作成時以降、この値を変更することはできません。

次に示すように、Port\_list プロパティは、サーバーデーモンが待機するポートを指定します。

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

このプロパティはデフォルト値を宣言しているため、クラスタ管理者はリソースを作成するときに、新しい値を指定するか、デフォルト値を使用するかを選択できます。Tunable 属性が AT\_CREATION に制限されているため、後でこの値を変更できるユーザーはいません。

## 関数とコールバックメソッドの命名規則

次の命名規則を覚えておけば、サンプルコードのさまざまな部分を特定できます。

- RMAPI 関数の名前は、scha\_ で始まります。
- DSDL 関数の名前は、scds\_ で始まります。
- コールバックメソッドの名前は、xfnts\_ で始まります。
- ユーザー定義関数の名前は、svc\_ で始まります。

## scds\_initialize() 関数

DSDL では、各コールバックメソッドの最初で `scds_initialize()` 関数を呼び出す必要があります。

この関数は次の作業を行います。

- フレームワークがデータサービスメソッドに渡すコマンド行引数 (`argc` と `argv`) を検査および処理します。メソッドは、追加のコマンド行引数を処理する必要はありません。
- ほかの DSDL 関数ができるように内部データ構造を設定します。
- ロギング環境を初期化します。
- 障害モニターの検証設定の妥当性を検査します。

`scds_close()` 関数を使用すると、`scds_initialize()` が割り当てたリソースを再利用できます。

## xfnts\_start メソッド

データサービスリソースを含むリソースグループがクラスタノードまたはゾーン上でオンラインになったとき、あるいは、リソースが有効になったとき、RGM はそのクラスタノードまたはゾーン上で `Start` メソッドを実行します。サンプルの SUNW.xfnts リソースタイプでは、`xfnts_start` メソッドが当該ノードまたはゾーン上で `xfns` デーモンを起動します。

`xfnts_start` メソッドは `scds_pmf_start()` を呼び出して、PMF の制御下でデーモンを起動します。PMF は、自動障害通知、再起動機能、および障害モニターとの統合を提供します。

---

注-`xfnts_start` は、`scds_initialize()` を最初に呼び出し、これによって、必要な「ハウスキーピング」関数が実行されます。詳細は、[148 ページ](#) の「[scds\\_initialize\(\) 関数](#)」と、`scds_initialize(3HA)` のマニュアルページを参照してください。

---

## X Font Server の起動前のサービスの検証

次に示すように、`xfnts_start` メソッドは X Font Server を起動する前に `svc_validate()` を呼び出して、`xfns` デーモンをサポートするための適切な構成が存在していることを確認します。

```
rc = svc_validate(scds_handle);
if (rc != 0) {
    scds_syslog(LOG_ERR,
```

```

        "Failed to validate configuration.");
    return (rc);
}

```

詳細については、163 ページの「[xfnts\\_validate メソッド](#)」を参照してください。

## svc\_start() によるサービスの起動

xfnts\_start メソッドは、xfnts.c ファイルで定義されている svc\_start() メソッドを呼び出して、xfs デーモンを起動します。ここでは、svc\_start() について説明します。

以下に、xfs デーモンを起動するためのコマンドを示します。

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

Confdir\_list 拡張プロパティーには *config-directory* を指定します。一方、Port\_list システムプロパティーには *port-number* を指定します。クラスタ管理者はデータサービスを構成するときに、これらのプロパティーの特定の値を指定します。

xfnts\_start メソッドはこれらのプロパティーを文字列配列として宣言します。xfnts\_start メソッドは、scds\_get\_ext\_confdir\_list() および scds\_get\_port\_list() 関数を使用して、クラスタ管理者が設定した値を取得します。これらの関数の詳細については、scds\_property\_functions(3HA) のマニュアルページを参照してください。

```

scha_str_array_t *confdirs;
scds_port_list_t  *portlist;
scha_err_t  err;

/* confdir_list プロパティーから構成ディレクトリを取得する。*/
confdirs = scds_get_ext_confdir_list(scds_handle);

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

/* Port_list プロパティーから XFS が使用するポートを取得する。*/
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);
}

```

confdirs 変数は配列の最初の要素(0)を指していることに注意してください。

xfnts\_start メソッドは sprintf() を使用して xfs のコマンド行を形成します。

```
/* xfs デーモンを起動するコマンドを構築する。 */  
(void) sprintf(cmd,  
    "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",  
    xfnts_conf, portlist->ports[0].port);
```

出力が `/dev/null` にリダイレクトされ、デーモンが生成するメッセージが抑制されることに注意してください。

次に示すように、`xfnts_start` メソッドは `xfs` コマンド行を `scds_pmf_start()` に渡して、PMF の制御下でデータサービスを起動します。

```
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_pmf_start()` を呼び出すときは、次のことに注意してください。

- `SCDS_PMF_TYPE_SVC` 引数は、データサービスアプリケーションとして起動するプログラムを指定します。このメソッドは、障害モニターなどのほかのタイプのアプリケーションも起動できます。
- `SCDS_PMF_SINGLE_INSTANCE` 引数は、これが単一インスタンスのリソースであることを指定します。
- `cmd` 引数は、以前に生成されているコマンド行です。
- 最後の引数である `-1` は、子プロセスの監視レベルを指定します。値 `-1` は、PMF がすべての子プロセスを親プロセスと同様に監視することを指定します。

`svc_pmf_start()` は `portlist` 構造体に割り当てられているメモリーを解放してから戻ります。

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

## svc\_start() からの復帰

`svc_start()` から正常に復帰した場合でも、基になるアプリケーションの起動に失敗することがあります。そのため、`svc_start()` はアプリケーションを検証して、アプリケーションが動作していることを確認してから、正常終了のメッセージを戻す必

要があります。検証では、アプリケーションがただちに利用できない理由として、アプリケーションの起動にはある程度時間がかかるということを考慮する必要があります。svc\_start() メソッドはxfnts.c ファイルで定義されている svc\_wait() を呼び出して、アプリケーションが動作していることを確認します。

```
/* サービスが完全に起動するまで待つ。 */
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() 関数は scds\_get\_netaddr\_list() を呼び出して、アプリケーションを検証するのに必要なネットワークアドレスリソースを取得します。

```
/* 検証に使用するネットワークリソースを取得する。 */
if (scds_get_netaddr_list(scds_handle, &netaddr)) {
    scds_syslog(LOG_ERR,
        "No network address resources found in resource group.");
    return (1);
}

/* ネットワークリソースが存在しない場合は、エラーを戻す。 */
if (netaddr == NULL || netaddr->num_netaddrs == 0) {
    scds_syslog(LOG_ERR,
        "No network address resource in resource group.");
    return (1);
}
```

svc\_wait() 関数は Start\_timeout および Stop\_timeout 値を取得します。

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

サーバーの起動に時間がかかることを考慮して、svc\_wait() は scds\_svc\_wait() を呼び出して、Start\_timeout 値の3%であるタイムアウト値を渡します。svc\_wait() 関数は svc\_probe() 関数を呼び出して、アプリケーションが起動していることを確認します。svc\_probe() メソッドは指定されたポート上でサーバーとの単純ソケット接続

を確立します。ポートへの接続が失敗した場合、`svc_probe()` は値 100 を戻して、致命的な障害であることを示します。ポートとの接続は確立したが、切断に失敗した場合、`svc_probe()` は値 50 を戻します。

`svc_probe()` が完全にまたは部分的に失敗した場合、`svc_wait()` は `scds_svc_wait()` をタイムアウト値 5 で呼び出します。`scds_svc_wait()` メソッドは、検証の周期を 5 秒ごとに制限します。また、このメソッドはサービスを起動しようとした回数も数えます。この回数が、リソースの `Retry_interval` プロパティで指定された期限内にリソースの `Retry_count` プロパティの値を超えた場合、`scds_svc_wait()` 関数は失敗します。この場合、`svc_start()` 関数も失敗します。

```
#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 {
    /*
     * ネットワークリソースの IP アドレスと portname 上で
     * データサービスを検証する。
     */
    rc = svc_probe(scds_handle,
        netaddr->netaddrs[0].hostname,
        netaddr->netaddrs[0].port_proto.port, probe_timeout);
    if (rc == SCHA_ERR_NOERR) {
        /* 成功。リソースを解放して終了。 */
        scds_free_netaddr_list(netaddr);
        return (0);
    }

    /* サービスが何度も失敗する場合は、scds_svc_wait() を呼び出す。
    if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
        != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR, "Service failed to start.");
        return (1);
    }

    /* RGM のタイムアウトを待ってプログラムを終了する。 */
} while (1);
```

---

注-xfnts\_startメソッドは終了する前にscds\_close()を呼び出して、scds\_initialize()が割り当てたリソースを再利用します。詳細は、[148 ページ](#)の「[scds\\_initialize\(\)関数](#)」と、scds\_close(3HA)のマニュアルページを参照してください。

---

## xfnts\_stopメソッド

xfnts\_startメソッドはscds\_pmf\_start()を使用してPMFのもとでサービスを起動するため、xfnts\_stopはscds\_pmf\_stop()を使用してサービスを停止します。

---

注-xfnts\_stopは、scds\_initialize()を最初に呼び出し、これによって、必要な「ハウスキーピング」関数が実行されます。詳細は、[148 ページ](#)の「[scds\\_initialize\(\)関数](#)」と、scds\_initialize(3HA)のマニュアルページを参照してください。

---

次に示すように、xfnts\_stopメソッドは、xfnts.cファイルで定義されているsvc\_stop()メソッドを呼び出します。

```
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); /* 正常に停止。 */
```

svc\_stop()からscds\_pmf\_stop()関数を呼び出すときは、次のことに注意してください。

- SCDS\_PMF\_TYPE\_SVC 引数は、データサービスアプリケーションとして停止するプログラムを指定します。このメソッドは、障害モニターなどのほかのタイプのアプリケーションも停止できます。
- SCDS\_PMF\_SINGLE\_INSTANCE 引数は、シグナルを指定します。
- SIGTERM 引数は、リソースインスタンスを停止するのに使用するシグナルを指定します。このシグナルでインスタンスを停止できなかった場合、scds\_pmf\_stop()はSIGKILLを送信してインスタンスを停止しようとします。このシグナルでもイ

インスタンスを停止できなかった場合、タイムアウトエラーで戻ります。詳細については、scds\_pmf\_stop(3HA) のマニュアルページを参照してください。

- タイムアウト値は、リソースの Stop\_timeout プロパティの値を示します。

---

注-xfnts\_stop メソッドは終了する前に scds\_close() を呼び出して、scds\_initialize() が割り当てたリソースを再利用します。詳細は、[148 ページ](#) の「scds\_initialize() 関数」と、scds\_close(3HA) のマニュアルページを参照してください。

---

## xfnts\_monitor\_start メソッド

RGM は、ノードまたはゾーンでリソースが起動されたあとに、そのノードまたはゾーン上で Monitor\_start メソッドを呼び出して障害モニターを起動します。xfnts\_monitor\_start メソッドは scds\_pmf\_start() を使用して PMF の制御下でモニターデーモンを起動します。

---

注-xfnts\_monitor\_start は、scds\_initialize() を最初に呼び出し、これによって、必要な「ハウスキーピング」関数が実行されます。詳細は、[148 ページ](#) の「scds\_initialize() 関数」と、scds\_initialize(3HA) のマニュアルページを参照してください。

---

次に示すように、xfnts\_monitor\_start メソッドは、xfnts.c ファイルに定義されている mon\_start メソッドを呼び出します。

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

/* scds_pmf_start を呼び出し、検証の名前を渡す。 */
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); /* モニターを正常に起動。 */
}
```

`svc_mon_start()` から `scds_pmf_start()` 関数を呼び出すときは、次のことに注意してください。

- `SCDS_PMF_TYPE_MON` 引数は、障害モニターとして起動するプログラムを指定します。このメソッドは、データサービスなどのほかのタイプのアプリケーションも起動できます。
- `SCDS_PMF_SINGLE_INSTANCE` 引数は、これが単一インスタンスのリソースであることを指定します。
- `xfnts_probe` 引数は、起動するモニターデーモンを指定します。このモニターデーモンは、ほかのコールバックプログラムと同じディレクトリに存在するものと想定されています。
- 最後の引数である `0` は、子プロセスの監視レベルを指定します。この場合、この値は PMF がモニターデーモンだけを監視することを示します。

---

注-`xfnts_monitor_start` メソッドは終了する前に `scds_close()` を呼び出して、`scds_initialize()` が割り当てたリソースを再利用します。詳細は、[148 ページ](#) の「[scds\\_initialize\(\) 関数](#)」と、`scds_close(3HA)` のマニュアルページを参照してください。

---

## xfnts\_monitor\_stop メソッド

`xfnts_monitor_start` メソッドは `scds_pmf_start()` を使用して PMF のもとでモニターデーモンを起動するため、`xfnts_monitor_stop` は `scds_pmf_stop()` を使用してモニターデーモンを停止します。

---

注-`xfnts_monitor_stop` は、`scds_initialize()` を最初に呼び出し、これによって、必要な「ハウスキーピング」関数が実行されます。詳細は、[148 ページ](#) の「[scds\\_initialize\(\) 関数](#)」と、`scds_initialize(3HA)` のマニュアルページを参照してください。

---

次に示すように、`xfnts_monitor_stop` メソッドは、`xfnts.c` ファイルで定義されている `mon_stop()` メソッドを呼び出します。

```
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); /* モニターを正常に停止。 */
}
```

svc\_mon\_stop() から scds\_pmf\_stop() 関数を呼び出すときは、次のことに注意してください。

- SCDS\_PMF\_TYPE\_MON 引数は、障害モニターとして停止するプログラムを指定します。このメソッドは、データサービスなどのほかのタイプのアプリケーションも停止できます。
- SCDS\_PMF\_SINGLE\_INSTANCE 引数は、これが単一インスタンスのリソースであることを指定します。
- SIGKILL 引数は、リソースインスタンスを停止するのに使用するシグナルを指定します。このシグナルでインスタンスを停止できなかった場合、scds\_pmf\_stop() はタイムアウトエラーで戻ります。詳細については、scds\_pmf\_stop(3HA) のマニュアルページを参照してください。
- タイムアウト値は、リソースの Monitor\_stop\_timeout プロパティの値を示します。

---

注-xfnts\_monitor\_stop メソッドは終了する前に scds\_close() を呼び出して、scds\_initialize() が割り当てたリソースを再利用します。詳細は、[148 ページ](#)の「[scds\\_initialize\(\) 関数](#)」と、scds\_close(3HA) のマニュアルページを参照してください。

---

## xfnts\_monitor\_check メソッド

障害モニターが、リソースが属するリソースグループを別のノードまたはゾーンにフェイルオーバーしようとするたびに、RGM は Monitor\_check メソッドを呼び出します。xfnts\_monitor\_check メソッドは svc\_validate() メソッドを呼び出して xfs デーモンをサポートするための適切な構成が存在していることを確認します。詳細については、[163 ページ](#)の「[xfnts\\_validate メソッド](#)」を参照してください。次に、xfnts\_monitor\_check のコードを示します。

```
/* RGM から渡された引数を処理し、syslog を初期化する。 */
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);

    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);

    /* モニター検査の一環として実行した検証メソッドの結果を戻す。 */
    return (rc);
}

```

## SUNW.xfnts 障害モニター

リソースがノードまたはゾーン上で起動されたあと、RGMはPROBEメソッドを直接呼び出すのではなく、代わりにMonitor\_startメソッドを呼び出してモニターを起動します。xfnts\_monitor\_startメソッドはPMFの制御下で障害モニターを起動します。xfnts\_monitor\_stopメソッドは障害モニターを停止します。

SUNW.xfnts 障害モニターは、次の処理を実行します。

- 単純なTCPベースのサービス(xfsなど)を検査するために特別に設計されたユーティリティを使用して、定期的にxfsサーバーデーモンの状態を監視します。
- (Retry\_countとRetry\_intervalプロパティを使用して)ある期間内にアプリケーションが遭遇した問題を追跡し、アプリケーションが完全に失敗した場合には、データサービスを再起動するか、フェイルオーバーするかどうかを決定します。scds\_fm\_action()とscds\_fm\_sleep()関数は、この追跡および決定機構の組み込みサポートを提供します。
- scds\_fm\_action()を使用して、フェイルオーバーまたは再起動の決定を実装します。
- リソースの状態を更新して、管理ツールやGUIで利用できるようにします。

## xfnts\_probe のメインループ

xfnts\_probe メソッドは、ループを実装します。

ループを実装する前に、xfnts\_probe は次の処理を行います。

- 次に示すように、xfnts リソース用のネットワークアドレスリソースを取得します。

```
/* 当該リソース用に利用できる IP アドレスを取得する。 */
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);
}

/* ネットワークリソースが存在しない場合、エラーを戻す。 */
if (netaddr == NULL || netaddr->num_netaddrs == 0) {
    scds_syslog(LOG_ERR,
        "No network address resource in resource group.");
    return (1);
}
```

- scds\_fm\_sleep() を呼び出し、タイムアウト値として Thorough\_probe\_interval の値を渡します。次に示すように、検証を実行する間、検証機能は Thorough\_probe\_interval で指定された期間、休眠状態になります。

```
timeout = scds_get_ext_probe_timeout(scds_handle);

for (;;) {
    /*
     * 連続する検証の間、thorough_probe_interval で指定された期間、
     * 休眠状態になる。
     */
    (void) scds_fm_sleep(scds_handle,
        scds_get_rs_thorough_probe_interval(scds_handle));
```

xfnts\_probe メソッドは次のようなループを実装します。

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
    /*
     * 状態を監視するホスト名と
     * ポートを取得する。
     */
    hostname = netaddr->netaddrs[ip].hostname;
    port = netaddr->netaddrs[ip].port_proto.port;
    /*
     * HA-XFS がサポートするポートは 1 つだけなので、
```

```

    * ポート値はポートの配列の最初の
    * エントリから取得する。
    */
    ht1 = gethrtime(); /* 検証開始時間を取得する。 */
    scds_syslog(LOG_INFO, "Probing the service on port: %d.", port);

    probe_result =
    svc_probe(scds_handle, hostname, port, timeout);

    /*
    * サービス検証履歴を更新し、
    * 必要に応じて、アクションを行う。
    * 検証終了時間を取得する。
    */
    ht2 = gethrtime();

    /* ミリ秒に変換する。 */
    dt = (ulong_t)((ht2 - ht1) / 1e6);

    /*
    * 障害の履歴を計算し、必要に応じて
    * アクションを実行する。
    */
    (void) scds_fm_action(scds_handle,
        probe_result, (long)dt);
} /* ネットワークリソースごとに */
} /* 検証を永続的に繰り返す。 */

```

svc\_probe() 関数は検証ロジックを実装します。svc\_probe() からの戻り値は scds\_fm\_action() に渡されます。そして scds\_fm\_action() は、アプリケーションを再起動するか、リソースグループをフェイルオーバーするか、あるいは何もしないかを決定します。

## svc\_probe() 関数

svc\_probe() 関数は、scds\_fm\_tcp\_connect() を呼び出して、指定のポートへの単純なソケット接続を作成します。接続に失敗した場合、svc\_probe() は 100 の値を戻して、致命的な障害であることを示します。接続には成功したが、切断に失敗した場合、svc\_probe() は 50 の値を戻して、部分的な障害であることを示します。接続と切断の両方に成功した場合、svc\_probe() は 0 の値を戻して、成功したことを示します。

次に、svc\_probe() のコードを示します。

```

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);
}

```

svc\_probe() は終了時に、成功 (0)、部分的な障害 (50)、または致命的な障害 (100) を示す値を戻します。xfnts\_probe メソッドはこの値を scds\_fm\_action() に渡します。

## 障害モニターのアクションの決定

xfnts\_probe メソッドは scds\_fm\_action() を呼び出して、実行すべきアクションを決定します。

`scds_fm_action()` のロジックは次のとおりです。

- `Retry_interval` プロパティの値の期間中に、障害の履歴を累積します。
- 累積した障害が 100 に到達した場合 (完全な障害)、データサービスを再起動します。`Retry_interval` を超えた場合、障害の履歴をリセットします。
- `Retry_interval` で指定された期間中に、再起動の回数が `Retry_count` プロパティを上回った場合、データサービスをフェイルオーバーします。

たとえば、検証機能が xfs サーバーに正常に接続したが、切断に失敗したものと想定します。これは、サーバーは動作しているが、ハングしていたり、一時的に過負荷状態になっている可能性を示しています。切断に失敗すると、`scds_fm_action()` に部分的な障害 (50) が送信されます。この値は、データサービスを再起動するしきい値を下回っていますが、値は障害の履歴に記録されます。

次の検証でもサーバーが切断に失敗した場合、`scds_fm_action()` が保持している障害の履歴に値 50 が再度追加されます。累積した障害の履歴が 100 になるので、`scds_fm_action()` はデータサービスを再起動します。

## xfnts\_validate メソッド

リソースが作成されたとき、および、リソースまたは(リソースを含む)グループのプロパティがクラスタ管理者によって更新されたとき、RGM は `Validate` メソッドを呼び出します。RGM は、作成または更新が行われる前に `Validate` メソッドを呼び出します。任意のノードまたはゾーン上でメソッドから失敗の終了コードが戻ってくると、作成または更新は取り消されます。

RGM が `Validate` を呼び出すのは、クラスタ管理者がリソースまたはリソースグループのプロパティを変更したときや、モニターが `Status` と `Status_msg` リソースプロパティを設定したときだけです。RGM がプロパティを設定する場合、RGM は `Validate` を呼び出しません。

---

注-`PROBE` メソッドがデータサービスを新しいノードまたはゾーンにフェイルオーバーしようとする際には常に、`Monitor_check` メソッドは `Validate` メソッドを明示的に呼び出します。

---

RGM は、ほかのメソッドに渡す引数以外にも、引数を追加して `Validate` メソッドを呼び出します。この追加引数には、更新されるプロパティと値が含まれます。`xfnts_validate` の開始時に実行される `scds_initialize()` の呼び出しにより、RGM が `xfnts_validate` に渡したすべての引数が解析され、その情報が `scds_handle` 引数に格納されます。この情報は、`xfnts_validate` が呼び出すサブルーチンによって使用されます。

xfnts\_validate メソッドは svc\_validate を呼び出して、次のことを検証します。

- Confdir\_list プロパティーがリソース用に設定されており、単一のディレクトリが定義されているかどうか。

```
scha_str_array_t *confdirs;
confdirs = scds_get_ext_confdir_list(scds_handle);

/* confdir_list 拡張プロパティーが存在しない場合、エラーを戻す。 */
if (confdirs == NULL || confdirs->array_cnt != 1) {
    scds_syslog(LOG_ERR,
        "Property Confdir_list is not set properly.");
    return (1); /* 検証に失敗 */
}
```

- Confdir\_list で指定されたディレクトリに fontserver.cfg ファイルが存在しているかどうか。

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

if (stat(xfnts_conf, &statbuf) != 0) {
    /*
     * errno.h プロトタイプには void 引数がないので、
     * lint エラーが抑制される。
     */
    scds_syslog(LOG_ERR,
        "Failed to access file <%=s> : <%=s>",
        xfnts_conf, strerror(errno)); /*lint !e746 */
    return (1);
}
```

- サーバーデーモンバイナリがクラスタノードまたはゾーン上でアクセスできるかどうか。

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

- Port\_list プロパティーが単一のポートを指定しているかどうか。

```
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); /* 検証に失敗 */
}
```

```

#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); /* 検証に失敗 */
    }
#endif

```

- データサービスが属するリソースグループにも、少なくとも1つのネットワークアドレスリソースが属しているかどうか。

```

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); /* 検証に失敗 */
}

/* ネットワークアドレスリソースが存在しない場合エラーを戻す。 */
if (snrlp == NULL || snrlp->num_netresources == 0) {
    scds_syslog(LOG_ERR,
        "No network address resource in resource group.");
    rc = 1;
    goto finished;
}

```

次に示すように、`svc_validate()` は戻る前に、割り当てられているすべてのリソースを解放します。

```

finished:
    scds_free_net_list(snrlp);
    scds_free_port_list(portlist);

    return (rc); /* 検証結果を戻す。 */

```

---

注-xfnts\_validate メソッドは終了する前に `scds_close()` を呼び出して、`scds_initialize()` が割り当てたリソースを再利用します。詳細は、[148 ページ](#) の「[scds\\_initialize\(\) 関数](#)」と、`scds_close(3HA)` のマニュアルページを参照してください。

---

## xfnts\_update メソッド

RGM は Update メソッドを呼び出して、実行中のリソースのプロパティが変更されたことをそのリソースに通知します。xfnts データサービスにおいて変更可能なプロパティは、障害モニターに関連したのだけです。したがって、プロパティが更新されたとき、常に xfnts\_update メソッドは scds\_pmf\_restart\_fm() を呼び出して、障害モニターを再起動します。

```
/* 障害モニターがすでに動作していることを検査し、動作している場合、
 * 障害モニターを停止および再起動する。scds_pmf_restart_fm() への
 * 2 番目のパラメータは、再起動する必要がある障害モニターの
 * インスタンスを一意に識別する。
 */
```

```
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.");
    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);
    return (1);
}

scds_syslog(LOG_INFO,
    "Completed successfully.");
```

---

注 - scds\_pmf\_restart\_fm() への 2 番目の引数は、複数のインスタンスが存在する場合に、再起動する障害モニターのインスタンスを一意に識別します。例にある値 0 は障害モニターのインスタンスが 1 つしか存在しないことを示します。

---

# Sun Cluster Agent Builder

---

この章では、Sun Cluster Agent Builder と、Agent Builder 用の Cluster Agent モジュールについて説明します。これらのツールは、Resource Group Manager (RGM) の制御下で動作するリソースタイプ (データサービス) の作成を自動化するものです。「リソースタイプ」とは、アプリケーションが RGM の制御下にあるクラスタ環境で動作できるようにするアプリケーションのラッパーのことです。

この章の内容は次のとおりです。

- 167 ページの「Agent Builder の概要」
- 168 ページの「Agent Builder の使用にあたって」
- 169 ページの「Agent Builder の使用」
- 186 ページの「Agent Builder で作成されるディレクトリ構造」
- 187 ページの「Agent Builder の出力」
- 191 ページの「Agent Builder の Cluster Agent モジュール」

## Agent Builder の概要

Agent Builder は、アプリケーションや作成するリソースタイプの種類に関する情報を入力するためのグラフィカルユーザーインターフェース (GUI) を提供します。Agent Builder は、ネットワーク対応のアプリケーションとネットワーク対応でないアプリケーション (非ネットワーク対応アプリケーション) をサポートします。ネットワーク対応アプリケーションは、ネットワークを使用してクライアントとの通信を行います。非ネットワーク対応アプリケーションは、スタンドアロンのアプリケーションです。

---

注 - GUI バージョンの Agent Builder にアクセスできない場合は、コマンド行インターフェースを使用して Agent Builder にアクセスできます。185 ページの「コマンド行バージョンの Agent Builder を使用する方法」を参照してください。

---

Agent Builder は、指定された情報にもとづき、次のソフトウェアを生成します。

- リソースタイプのメソッドコールバックに対応したフェイルオーバータイプまたはスケラブルリソースタイプ向けの C、Korn シェル (ksh)、または汎用データサービス (GDS) ソースファイル群。これらのファイルは、ネットワーク対応アプリケーション (クライアントサーバーモデル) と非ネットワーク対応 (クライアントレス) アプリケーションの両方に対応します。
- C シェルまたは Korn シェルのソースコードを生成する場合は、カスタマイズされたリソースタイプ登録 (Resource Type Registration: RTR) ファイル。
- リソースタイプのインスタンス (リソース) を起動、停止、および削除するためのカスタマイズされたユーティリティスクリプト。また、これらの各ファイルの使用方法を説明するカスタマイズされたマニュアルページ。
- C のソースコードを生成する場合はバイナリを含む Solaris パッケージとユーティリティスクリプト。C または Korn シェルのソースコードを生成する場合は RTR ファイルを含む Solaris パッケージとユーティリティスクリプト。

Agent Builder を使って、プロセス監視機能 (PMF) によって個別に監視および再起動される複数の独立したプロセスツリーを持つアプリケーション用のリソースタイプを生成できます。

## Agent Builder の使用にあたって

Agent Builder を使用する場合は、独立した複数のプロセスツリーを持つリソースタイプの作成方法をあらかじめ認識しておく必要があります。

Agent Builder は、複数の独立したプロセスツリーを持つアプリケーション用のリソースタイプを作成できます。これらのプロセスツリーは、RMF によって監視と起動が個別に行われるという意味でそれぞれ独立していると言えます。PMF は、独自のタグを使用して各プロセスツリーを起動します。

---

注 - Agent Builder を使って、複数の独立したプロセスツリーをもつリソースタイプを作成できますが、そのためには、生成されるソースコードとして C か GDS を指定する必要があります。Agent Builder を使って、このようなリソースタイプを Korn シェル用に作成することはできません。Korn シェル用にこれらのリソースタイプを作成するには、それらのコードを手動で作成する必要があります。

---

複数の独立したプロセスツリーを持つベースアプリケーションの場合、1つのコマンド行だけでアプリケーションを起動することはできません。代わりに、アプリケーションの各プロセスツリーを起動するコマンドへの完全パスを行ごとに記述したテキストファイルを作成します。このファイルには空白行を含めることはできません。そして、このファイルへのパスを Agent Builder 構成画面の「起動コマンド」テキストフィールドに指定します。

このファイルに実行権を設定しないことで、Agent Builder はこのファイルを識別できません。このファイルは、複数のコマンドが入ったシンプルな実行可能スクリプトから複数のプロセスツリーを起動するためのものです。このテキストファイルに実行権を設定しても、リソースはクラスタ上で問題なく動作するように見えます。しかし、すべてのコマンドが1つのPMFタグ下で起動されるため、PMFはプロセスツリーの監視と再起動を個別に行うことができません。

## Agent Builder の使用

この節では、Agent Builder の使用方法について説明します。また、Agent Builder を使用する前に実施すべき作業についても説明しています。リソースタイプコードを生成したあとで Agent Builder を活用する方法についても説明します。

この章では、次の内容について説明します。

- 169 ページの「アプリケーションの分析」
- 170 ページの「Agent Builder のインストールと構成」
- 170 ページの「Agent Builder 画面」
- 171 ページの「Agent Builder の起動」
- 172 ページの「Agent Builder のナビゲーション」
- 175 ページの「作成画面の使用」
- 178 ページの「構成画面の使用」
- 180 ページの「Agent Builder の Korn シェルベース `$hostnames` 変数の使用」
- 181 ページの「プロパティー変数の使用」
- 183 ページの「Agent Builder で作成したコードの再利用」
- 185 ページの「コマンド行バージョンの Agent Builder を使用する方法」

## アプリケーションの分析

Agent Builder を使用する前に、高可用アプリケーションまたはスケーラブルアプリケーションにしようとしているアプリケーションが必要な条件を満たしているかを確認します。この分析はアプリケーションの実行時特性だけに基づくものなので、Agent Builder はこの分析を行うことができません。詳細は、29 ページの「アプリケーションの適合性の分析」を参照してください。

Agent Builder を使用しても、アプリケーションに適した完全なリソースタイプを必ず作成できるとはかぎりません。しかし、一般に Agent Builder は少なくとも部分的なソリューションにはなりません。たとえば、比較的機能の高いアプリケーションでは、Agent Builder がデフォルトでは生成しないコード(プロパティーの妥当性検査を追加するコードや Agent Builder がエクスポートしないパラメータを調節するコードなど)を別途生成する必要が生じる場合があります。このような場合、生成されたコードまたは RTR ファイルを修正する必要があります。Agent Builder は、まさにこのような柔軟性をもたらすように設計されています。

Agent Builder は、生成されるソースコードの特定の場所にコメントを埋め込みます。ユーザーは、この場所に独自のリソースタイプコードを追加できます。ソースコードを修正したあと、Agent Builder が生成した Makefile を使用すれば、ソースコードを再コンパイルし、リソースタイプパッケージを生成し直すことができます。

Agent Builder が生成したリソースタイプコードを使用せずに、リソースタイプコードを完全に作成し直す場合でも、Agent Builder が生成した Makefile やディレクトリ構造を使用すれば、独自のリソースタイプ用の Solaris パッケージを作成できます。

## Agent Builder のインストールと構成

Agent Builder を個別にインストールする必要はありません。Agent Builder は、Sun Cluster ソフトウェアのインストール時にデフォルトでインストールされる SUNWscdev パッケージに含まれます。詳細は、『Sun Cluster ソフトウェアのインストール (Solaris OS 版)』を参照してください。

Agent Builder を使用する前に、次の要件を確認してください。

- \$PATH 変数に Java 実行時環境が含まれているかどうか。Agent Builder は、Java Development Kit (Version 1.3.1) 以降に依存しています。Java Development Kit が \$PATH 変数に含まれていないと、Agent Builder コマンド (scdsbuilder) はエラーメッセージを返します。
- Solaris 9 OS または Solaris 10 OS の「Developer System Support」ソフトウェアグループがインストールされていること。
- cc コンパイラが \$PATH 変数に含まれているか。Agent Builder は \$PATH 変数で最初に現れる cc を使用して、リソースタイプの C バイナリコードを生成するコンパイラを識別します。cc が \$PATH に存在しない場合、Agent Builder は C コードを生成するオプションを無効にします。詳細は、175 ページの「作成画面の使用」を参照してください。

---

注 - Agent Builder では、標準の cc コンパイラ以外のコンパイラも使用できます。別のコンパイラを使用するためには、そのコンパイラ (gcc など) に対するシンボリックリンクを \$PATH 内に指定します。あるいは、Makefile におけるコンパイラ指定 (現在は CC=cc) を変更し、別のコンパイラへの完全パスを指定することもできます。たとえば、Agent Builder により生成される Makefile の中で、CC=cc を CC=pathname/gcc に変更します。この場合、Agent Builder を直接実行することはできません。代わりに、make や make pkg コマンドを使用して、データサービスコードとパッケージを生成する必要があります。

---

## Agent Builder 画面

Agent Builder は2つのステップ(画面)からなるウィザードです。

Agent Builder では、次の2つの画面を使用して、新しいリソースタイプを作成します。

1. 「作成画面」この画面では、作成するリソースタイプについての基本情報(リソースタイプの名前や生成されるファイルの作業ディレクトリなど)を指定します。作業ディレクトリは、リソースタイプテンプレートの作成や構成に使用する場所です。

次の情報も指定します。

- 作成するリソースの種類(スケーラブルまたはフェイルオーバー)
- ベースアプリケーションがネットワーク対応かどうか(つまり、そのクライアントとの通信にネットワークを使用するか)
- 生成するコードのタイプ(C、Korn シェル(ksh)、またはGDS)

GDSの詳細は、[第10章](#)を参照してください。この画面の情報はすべて指定する必要があります。指定後、「作成」を選択してその出力を生成します。この後「構成」画面を表示できます。

2. 「構成画面」この画面には、ベースアプリケーションを起動するために任意のUNIXシェルに渡すことができる完全なコマンド行を指定する必要があります。オプションとして、アプリケーションを停止するコマンドや検証するコマンドも提供できます。これらの2つのコマンドを指定しないと、生成される出力は信号を送信してアプリケーションを停止し、デフォルトの検証メカニズムを提供します。検証コマンドの説明は、[178ページ](#)の「[構成画面の使用](#)」を参照してください。「構成」画面では、起動コマンド、停止コマンド、および検証コマンドのタイムアウト値の変更も行えます。

## Agent Builder の起動

---

注 - GUIバージョンの Agent Builder にアクセスできない場合は、コマンド行インタフェースを使用して Agent Builder にアクセスできます。[185ページ](#)の「[コマンド行バージョンの Agent Builder を使用する](#)方法」を参照してください。

---

既存のリソースタイプの作業ディレクトリから Agent Builder を起動すると、このツールは「作成」画面と「構成」画面を既存のリソースタイプの値に初期化します。

Agent Builder は次のコマンドを入力して起動します。

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

「作成」画面が表示されます。

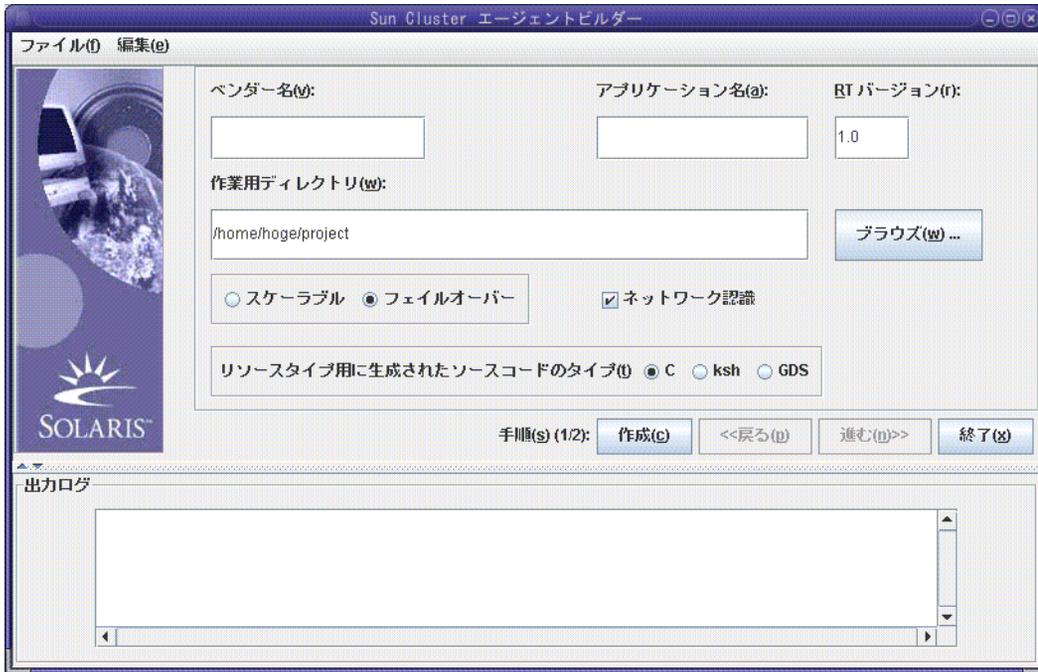


図 9-1 Agent Builder の「作成」画面

## Agent Builder のナビゲーション

「作成」画面と「構成」画面の情報は、次の操作で入力します。

- フィールドに情報を入力
- ディレクトリ構造をブラウズして、ファイルまたはディレクトリを選択
- ラジオボタンの中から1つだけ選択 (たとえば、「スケラブル」または「フェイルオーバー」)
- 「ネットワーク認識」チェックボックスのオン/オフ切り替え。ベースアプリケーションをネットワーク対応と指定する場合はこのボックスを選択状態にし、非ネットワーク対応アプリケーションと指定する場合はこのボックスを空のままにします。

各画面の下にあるボタンを使用すると、作業を完了したり、次の画面に進んだり、以前の画面に戻ったり、Agent Builder を終了したりできます。Agent Builder は、必要に応じてこれらのボタンを強調表示にしたり、グレー表示にしたりします。

たとえば、「作成」画面で必要なフィールドに入力し、希望するオプションを選択してから、画面の下にある「作成」ボタンをクリックします。この時点で、以前の画面は存在しないので、「戻る」ボタンはグレー表示されます。また、この作業が完成するまで次の手順には進めないため、「進む」ボタンもグレー表示されます。



Agent Builder は、画面の下にある「出力ログ」領域に進捗メッセージを表示します。作業が終了したとき、Agent Builder は成功メッセージまたは警告メッセージを表示します。「進む」が強調表示されます。あるいは、これが最後の画面の場合は、「キャンセル」だけが強調表示されます。

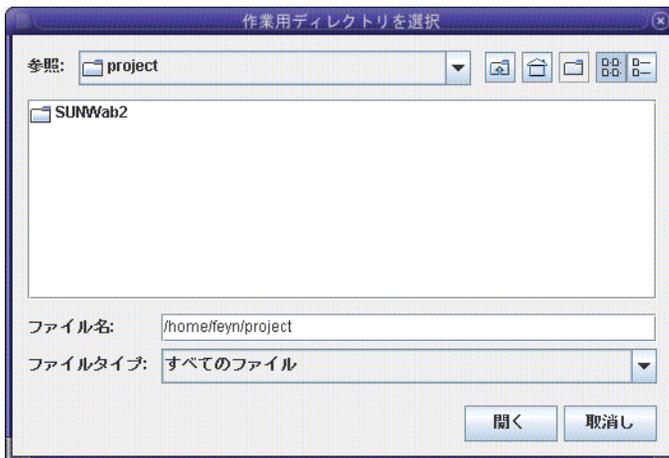
「キャンセル」ボタンをクリックすると、任意の時点で Agent Builder を終了できます。

## 「ブラウズ」コマンド

Agent Builder のフィールドには、そのフィールドに情報を入力できるものや、「ブラウズ」をクリックしてディレクトリ構造内をブラウズし、ファイルまたはディレクトリを選択できるものなどがあります。



「ブラウズ」をクリックすると、次のような画面が表示されます。



フォルダをダブルクリックすると、フォルダが開きます。カーソルをファイルに移動させると、「ファイル名」フィールドにファイルの名前が表示されます。必要なファイルを見つけ、そこにカーソルを移動したら、「選択」をクリックします。

注-ディレクトリをブラウズする場合は、必要なディレクトリにカーソルを移動し、「開く」をクリックします。ディレクトリにサブディレクトリがない場合、Agent Builder はブラウズウィンドウを閉じて、ユーザーがカーソルを移動したディレクトリの名前を適切なフィールドに表示します。サブディレクトリがある場合、「閉じる」をクリックすると、ブラウズウィンドウが閉じて、以前の画面に戻ります。Agent Builder は、ユーザーがカーソルを移動したディレクトリの名前を適切なフィールドに表示します。

「ブラウズ」画面の右上隅にあるアイコンには、次のような処理を行います。

アイコン	目的
	ディレクトリツリーの1つ上のレベルに移動します。
	ホームフォルダに戻ります。
	現在選択しているフォルダの下に新しいフォルダを作成します。
	ビューを切り替えます。将来のために予約されています。

## Agent Builder のメニュー

Agent Builder には、ドロップダウンメニューとして「ファイル」メニューと「編集」メニューがあります。

### Agent Builder の「ファイル」メニュー

「ファイル」メニューでは、次の2つのオプションを使用できます。

- 「リソースタイプをロード」既存のリソースタイプをロードします。Agent Builder が提供するブラウズ画面を使用して、既存のリソースタイプ用の作業ディレクトリを選択します。Agent Builder を起動したディレクトリにリソースタイプが存在する場合、Agent Builder は自動的にそのリソースタイプをロードします。「リソースタイプをロード」コマンドを使用すると、任意のディレクトリから Agent Builder を起動したあと、既存のリソースタイプを選択して、新しいリソースタイプを作成するためのテンプレートとして使用できます。[183 ページの「Agent Builder で作成したコードの再利用」](#)を参照してください。
- 「終了」Agent Builder を終了します。「作成」または「構成」画面で「キャンセル」をクリックして Agent Builder を終了することもできます。

## Agent Builder の「編集」メニュー

「編集」メニューでは、次の2つのオプションを使用できます。

- 「出力ログをクリア」。出力ログの情報を消去します。「作成」または「構成」を選択するたびに、Agent Builder は状態メッセージを出力ログに追加します。繰り返しソースコードを修正し、Agent Builder で出力を生成し直しているときに、出力の生成ごとに状態メッセージを記録する場合は、出力ログを使用するたびにログファイルの内容を保存および消去できます。
- 「出力ログを保存」。ログ出力をファイルに保存します。Agent Builder が提供するブラウザ画面を使用すると、ディレクトリを選択して、ファイル名を指定できます。

## 作成画面の使用

リソースタイプを作成する最初の段階では、Agent Builder を起動したときに表示される「作成」画面に必要な情報を入力します。次の図は、フィールドに情報を入力したあとの「作成」画面を示しています。

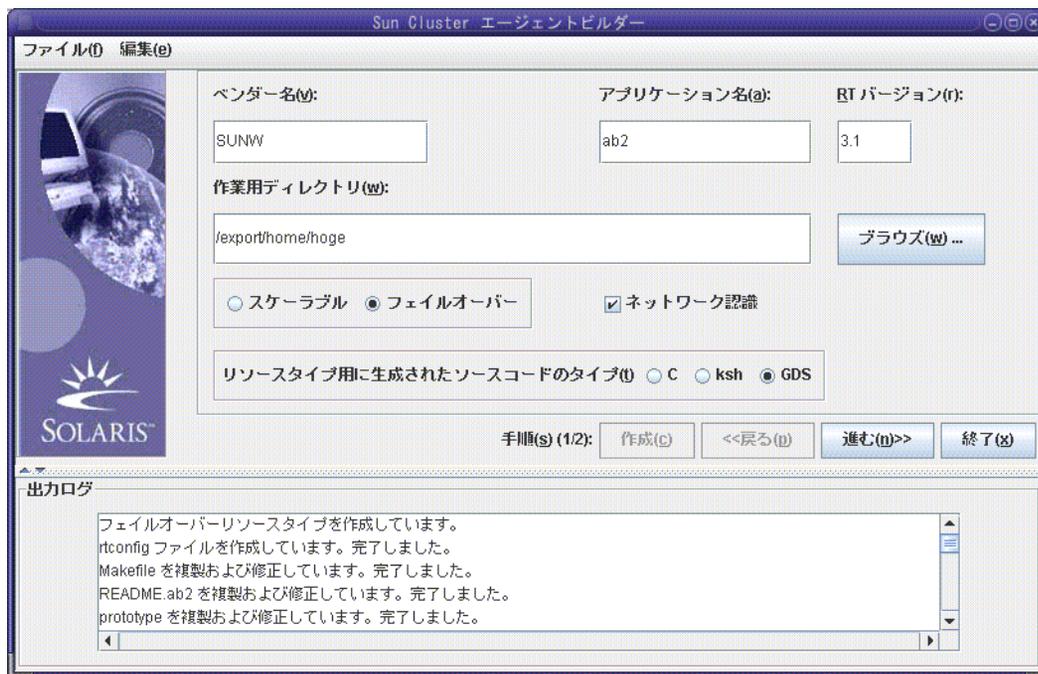


図 9-2 Agent Builder の「作成」画面 (情報の入力後)

作成画面には、次のフィールド、ラジオボタン、およびチェックボックスがあります。

- 「ベンダー名」。リソースタイプのベンダーの名前。通常、ベンダーの株式の略号を指定します。しかし、ベンダーを一意に識別する名前であれば、どのような名前でも有効です。英数字文字だけを使用します。
- 「アプリケーション名」。リソース型の名前です。英数字文字だけを使用します。

---

注-ベンダー名とアプリケーション名の両方で、リソースタイプの完全な名前が形成されます。Solaris 9 OS 以降では、ベンダー名とアプリケーション名の両方を合わせて 10 文字以上を指定できます。

---

- 「RT バージョン」。生成されるリソースタイプのバージョン。同一のベースリソースタイプのバージョン(またはアップグレード)が複数登録されている場合は、「RT バージョン」でそれらを区別します。

「RT バージョン」フィールドで次の文字を使用することはできません。

- スペース
  - タブ
  - スラッシュ (/)
  - 逆スラッシュ (\)
  - アスタリスク (\*)
  - 疑問符 (?)
  - コンマ (,)
  - セミコロン (;)
  - 左角括弧 ( [ )
  - 右角括弧 ( ] )
- 「作業ディレクトリ」 Agent Builder は、このディレクトリの中に、ターゲットリソースタイプ用のすべてのファイルを格納するディレクトリ構造を作成します。1つの作業ディレクトリには1つのリソースタイプしか作成できません。Agent Builder は、このフィールドを Agent Builder が起動されたディレクトリのパスで初期化します。ただし、別のディレクトリ名を入力したり、「ブラウズ」を使用して異なるディレクトリを指定することもできます。

Agent Builder は、作業ディレクトリの下にリソースタイプ名を持つサブディレクトリを作成します。たとえば、ベンダー名が SUNW で、アプリケーション名が ftp である場合、Agent Builder はこのサブディレクトリに SUNWftp という名前をつけます。

Agent Builder は、ターゲットリソースタイプ用のすべてのディレクトリとファイルをこのサブディレクトリの下に置きます。186 ページの「Agent Builder で作成されるディレクトリ構造」を参照してください。

- 「スケーラブル」または「フェイルオーバー」。ターゲットのリソースタイプがフェイルオーバーなのかスケーラブルなのかを指定します。
- 「ネットワーク認識」。ベースアプリケーションがネットワーク対応かどうかを指定します。つまり、アプリケーションがネットワークを使用してクライアントと通信するかどうかを指定します。ネットワーク対応であることを指定する場合は「ネットワーク認識」チェックボックスを選択し、非ネットワーク対応を指定する場合は選択しません。
- 「C, ksh」。生成されるソースコードの言語を指定します。この2つのオプションを同時に指定することはできませんが、Agent Builder では、Korn シェルで生成されたコードでリソースタイプを作成してから、同じ情報を再使用して、C で生成されたコードを作成することができます。183 ページの「Agent Builder で作成したコードの再利用」を参照してください。
- 「GDS」。このサービスが汎用データサービスであることを示します。汎用データサービスの詳しい作成および構成方法については、第 10 章を参照してください。

---

注 - cc コンパイラが \$PATH 変数に含まれていないと、Agent Builder は「C」ラジオボタンをグレー表示し、「ksh」ラジオボタンを選択可能にします。別のコンパイラを指定する方法については、170 ページの「Agent Builder のインストールと構成」の最後にある注記を参照してください。

---

必要な情報を入力したあと、「作成」をクリックします。画面の一番下にある「出力ログ」領域には、Agent Builder が行なったアクションが表示されます。「編集」メニューの「出力ログを保存」を使用すれば、出力ログ内の情報を保存できます。

これが終わると、Agent Builder は、成功メッセージか警告メッセージを表示します。

- Agent Builder がこの手順を正常に終了できなかった場合は、出力ログで詳しい情報を調べてください。
- Agent Builder が正常に完了した場合は、「進む」をクリックして「構成」画面を表示します。この画面でリソースタイプの生成を完了することができます。

---

注 - 完全なリソースタイプを生成するには2段階の作業が必要ですが、最初の段階（つまり、作成）が完了したあとに Agent Builder を終了しても、指定した情報や Agent Builder で作成した内容が失われることはありません。183 ページの「Agent Builder で作成したコードの再利用」を参照してください。

---

## 構成画面の使用

リソースタイプを作成する最初の段階では、Agent Builder を起動したときに表示される「作成」画面に必要な情報を入力します。すると、次の画面が表示されます。リソースタイプの作成が完了していなければ、構成画面にはアクセスできません。

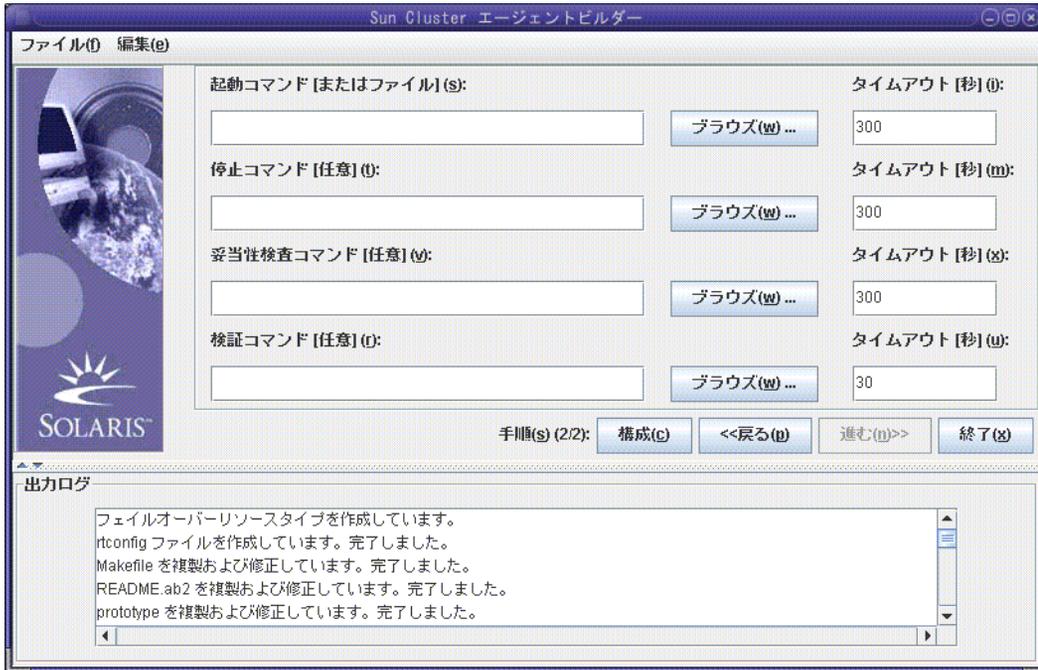


図 9-3 Agent Builder の「構成」画面

構成画面には、次のフィールドがあります。

- 「起動コマンド」。ベースアプリケーションを起動するために任意の UNIX シェルに渡すことができる完全なコマンド行。これには、起動コマンドを指定する必要があります。このフィールドにコマンドを入力するか、「ブラウズ」ボタンを使用して、アプリケーションを起動するコマンドが記述されているファイルを指定します。

完全なコマンド行には、アプリケーションを起動するのに必要なすべての要素が含まれていなければなりません。たとえば、ホスト名、ポート番号、構成ファイルへのパスなどです。あるいは、プロパティー変数を指定することもできます。この変数については、[181 ページの「プロパティー変数の使用」](#)を参照してください。Korn シェルベースのアプリケーションにコマンド行からホスト名を指定する必要がある場合は、Agent Builder が定義する `$hostnames` 変数を使用できます。詳細は、[180 ページの「Agent Builder の Korn シェルベース \\$hostnames 変数の使用」](#)を参照してください。

コマンドは二重引用符 (") で囲んではいけません。

---

注-ベースアプリケーションが複数の独立したプロセスツリーを持ち、各プロセスツリーが Process Monitor Facility (PMF) の制御下で独自のタグによって起動される場合、単一のコマンドは指定できません。代わりに、各プロセスツリーを起動するための個々のコマンドを記述したテキストファイルを作成し、そのファイルへのパスを「起動コマンド」テキストフィールドに指定する必要があります。[168 ページの「Agent Builder の使用にあたって」](#)を参照してください。この節には、このファイルが適切に機能するために必要な特性が示されています。

---

- 「停止コマンド」。ベースアプリケーションを停止するために任意の UNIX シェルに渡すことができる完全なコマンド行。このフィールドにコマンドを入力するか、「ブラウズ」ボタンを使用して、アプリケーションを停止するコマンドが記述されているファイルを指定します。あるいは、プロパティー変数を指定することもできます。この変数については、[181 ページの「プロパティー変数の使用」](#)を参照してください。Korn シェルベースのアプリケーションにコマンド行からホスト名を指定する必要がある場合は、Agent Builder が定義する `$hostnames` 変数を使用できます。詳細は、[180 ページの「Agent Builder の Korn シェルベース \\$hostnames 変数の使用」](#)を参照してください。

このコマンドは省略可能です。

停止コマンドを指定しない場合、生成されるコードは、次に示すように、Stop メソッドでシグナルを使用して、アプリケーションを停止します。

- Stop メソッドは SIGTERM を送信してアプリケーションを停止しようとします。そして、アプリケーション用のタイムアウト値の 80% だけ待機して、停止しない場合は終了します。
- SIGTERM シグナルが失敗した場合、Stop メソッドは SIGKILL を送信して、アプリケーションを停止しようとします。そして、アプリケーション用のタイムアウト値の 15% だけ待機して、停止しない場合は終了します。
- SIGKILL が失敗した場合、Stop メソッドは異常終了します。タイムアウト値の残りの 5% はオーバーヘッドとみなされます。



注意 - 停止コマンドは、アプリケーションが完全に停止するまで戻らないことに注意してください。

- 「検証コマンド」。定期的に行われ、アプリケーションの状態を検査して、0 (正常) から 100 (致命的な障害) の範囲の終了状態に戻すコマンド。このコマンドは省略可能です。このフィールドにコマンドの完全パスを入力するか、「ブラウザ」を使用して、アプリケーションを検証するコマンドが記述されているファイルを指定します。

通常は、単にベースアプリケーションのクライアントを指定します。検証コマンドを指定しない場合、生成されるコードは、リソースが使用するポートへの接続と切断を試みます。接続と切断に成功すれば、アプリケーションの状態が正常であると判断します。あるいは、プロパティ変数を指定することもできます。この変数については、181 ページの「プロパティ変数の使用」を参照してください。Korn シェルベースのアプリケーションに検証コマンド行からホスト名を指定する必要がある場合は、Agent Builder が定義する \$hostnames 変数を使用できません。詳細は、180 ページの「Agent Builder の Korn シェルベース \$hostnames 変数の使用」を参照してください。

コマンドは二重引用符 (") で囲んではいけません。

- 「タイムアウト」各コマンドのタイムアウト値 (秒数)。新しい値を指定するか、Agent Builder が提供するデフォルト値を受け入れます。起動コマンドと停止コマンドのデフォルト値は 300 秒で、検証コマンドのデフォルト値は 30 秒です。

## Agent Builder の Korn シェルベース \$hostnames 変数の使用

多くのアプリケーション (特に、ネットワーク対応アプリケーション) では、アプリケーションが通信し、顧客の要求に対してサービスを提供するホスト名をコマンド行に指定して、アプリケーションに渡す必要があります。多くの場合、ホスト名

は、構成画面において、ターゲットリソースタイプの起動、停止、および検証コマンドに指定する必要がある引数です。しかし、アプリケーションが待機するホスト名はクラスタ固有のもので、つまり、ホスト名はリソースがクラスタで実行されるときに決められ、Agent Builder がリソースタイプコードを生成する時点で決めることはできません。

この問題を解決するために、Agent Builder は `$hostnames` 変数を提供します。この変数を使用すると、起動、停止、および検証コマンドのコマンド行にホスト名を指定できます。

---

注 - `$hostnames` 変数は、Korn シェルベースのサービスでのみサポートされます。つまり、C ベースや GDS ベースのサービスではサポートされません。

---

`$hostnames` 変数を指定する方法は、実際のホスト名を指定する方法と同じです。たとえば、次のようになります。

```
% /opt/network_aware/echo_server -p port-no -l $hostnames
```

ターゲットリソースタイプのリソースがあるクラスタ上で動作するとき、そのリソースに構成されている `LogicalHostname` または `SharedAddress` ホスト名が `$hostnames` 変数の値に置き換えられます。リソースのホスト名は、`Network_resources_used` リソースのリソースプロパティで構成されます。

`Network_resources_used` プロパティに複数のホスト名を構成している場合、すべてのホスト名をコンマで区切って `$hostnames` 変数に指定します。

## プロパティ変数の使用

プロパティ変数を使用すれば、Sun Cluster のリソースタイプ、リソース、リソースグループの一部のプロパティの値を RGM フレームワークから取り出すことができます。Agent Builder は起動、検証、停止のコマンド文字列をスキャンしてプロパティ変数がないかをチェックし、プロパティ変数があればコマンドを実行する前にそれらの変数を対応する値に置き換えます。

---

注 - プロパティ変数は、Korn シェルベースのサービスではサポートされません。

---

## プロパティ変数のリスト

この節では、使用できるプロパティ変数を示します。Sun Cluster のリソースタイプ、リソース、リソースグループのプロパティについては、[付録 A](#) を参照してください。

## リソースプロパティ変数

- HOSTNAMES
- RS\_CHEAP\_PROBE\_INTERVAL
- RS\_MONITOR\_START\_TIMEOUT
- RS\_MONITOR\_STOP\_TIMEOUT
- RS\_NAME
- RS\_NUM\_RESTARTS
- RS\_RESOURCE\_DEPENDENCIES
- RS\_RESOURCE\_DEPENDENCIES\_WEAK
- RS\_RETRY\_COUNT
- RS\_RETRY\_INTERVAL
- RS\_SCALABLE
- RS\_START\_TIMEOUT
- RS\_STOP\_TIMEOUT
- RS\_THOROUGH\_PROBE\_INTERVAL
- SCHA\_STATUS

## リソースタイププロパティ変数

- RT\_API\_VERSION
- RT\_BASEDIR
- RT\_FAILOVER
- RT\_INSTALLED\_NODES
- RT\_NAME
- RT\_RT\_VERSION
- RT\_SINGLE\_INSTANCE

## リソースグループプロパティ変数

- RG\_DESIRED\_PRIMARIES
- RG\_GLOBAL\_RESOURCES\_USED
- RG\_IMPLICIT\_NETWORK\_DEPENDENCIES
- RG\_MAXIMUM\_PRIMARIES
- RG\_NAME
- RG\_NODELIST
- RG\_NUM\_RESTARTS
- RG\_PATHPREFIX
- RG\_PINGPONG\_INTERVAL
- RG\_RESOURCE\_LIST

## プロパティ変数の構文

プロパティ変数を指定する場合は、プロパティ名の前にパーセント符号(%)を指定します。次はその例です。

```
/opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

上の例の場合、Agent Builder はこれらのプロパティ変数を解釈し、たとえば、次の値を使って echo\_server スクリプトを起動します。

```
/opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

## Agent Builder によるプロパティ変数の置き換え

Agent Builder では、プロパティ変数のタイプは次のように解釈されます。

- 整数は、その実際の値 (たとえば 300) で置き換えられます。
- ブール値は、文字列 TRUE か FALSE で置き換えられます。
- 文字列は、実際の文字列 (たとえば phys-node-1) で置き換えられます。
- 文字列リストの場合は、リストが、コマンドで区切られた実際の値で置き換えられます (たとえば、phys-node-1,phys-node-2,phys-node-3)。
- 整数リストの場合は、リスト内のすべてのメンバーで置き換えられます。この場合、各整数は 1,2,3 のように区切られます。
- 列挙タイプは、その値 (文字列形式) で置き換えられます。

## Agent Builder で作成したコードの再利用

Agent Builder を使用すると、次のような方法で、完成した作業内容を再利用できます。

- Agent Builder で作成した既存のリソースタイプのクローンを作成できます。
- Agent Builder が生成したソースコードを編集して、そのコードを再コンパイルすれば、新しいパッケージを作成できます。

### ▼ 既存のリソースタイプからクローンを作成する方法

Agent Builder で作成した既存のリソースタイプのクローンを作成するには、次の手順に従います。

#### 1 既存のリソースタイプを Agent Builder にロードします。

次のいずれかの方法を使用します。

- Agent Builder で作成された既存のリソースタイプの作業ディレクトリから Agent Builder を起動します。作業ディレクトリに rtconfig ファイルが含まれているか確認します。Agent Builder がこのリソースタイプの値を「作成」や「構成」画面にロードします。
- 「ファイル」ドロップダウンメニューの「リソースタイプをロード」オプションを使用します。

- 2 作成画面で作業ディレクトリを変更します。  
「ブラウズ」を使ってディレクトリを選択する必要があります。新しいディレクトリ名を入力するだけでは不十分です。ディレクトリを選択したあと、Agent Builder は「作成」ボタンを有効に戻します。
- 3 必要に応じて既存のリソースタイプに変更を加えます。  
リソースタイプ用に生成されたコードのタイプを変更できます。  
たとえば、初めに Korn シェルバージョンのリソースタイプを作成し、あとで C バージョンのリソースタイプが必要になった場合には、次の手順で対応できます。
  - 既存の Korn シェルリソースタイプをロードする
  - 出力用の言語を C に変更する
  - 「作成」をクリックしてリソースタイプの C バージョンを構築する
- 4 リソースタイプのクローンを作成します。
  - a. 「作成」をクリックして、リソースタイプを作成します。
  - b. 「次へ」をクリックして「構成」画面を表示します。
  - c. 「構成」をクリックしてリソースタイプを構成し、次に「キャンセル」をクリックして終了します。

## 生成されたソースコードの編集

リソースタイプを作成するプロセスを簡単にするために、Agent Builder は入力できる情報量を制限しています。必然的に、生成されるリソースタイプの範囲も制限されます。したがって、より複雑な機能を追加するには、生成されたソースコードまたは RTR ファイルを修正する必要があります。付加的な機能の例としては、プロパティの妥当性検査を追加するコードや、Agent Builder がエクスポートしないパラメータを調節するコードなどが挙げられます。

ソースファイルは、*install-directory/rt-name/src* ディレクトリに置かれます。Agent Builder は、ソースコード内においてコードを追加できる場所にコメント文を埋め込みます。このようなコメントの形式は次のとおりです (C コードの場合)。

```
/* User added code -- BEGIN vvvvvvvvvvvvvvvvvv */  
/* User added code -- END   ^^^^^^^^^^^^^^^^^^^ */
```

---

注 - コメントは Korn シェルソースコードのものと同じですが、Korn シェルソースコードの場合は、コメント記号 (#) がコメントの始めを表します。

---

たとえば、`rt-name.h` は、さまざまなプログラムが使用するユーティリティ関数をすべて宣言します。宣言リストの最後はコメント文になっており、ここでは自分のコードに追加したい関数を宣言できます。

`install-directory/rt-name/src` ディレクトリには、対応するターゲットと共に Makefile も生成されます。make コマンドを使用すると、ソースコードを再コンパイルできます。make pkg コマンドを使用すると、リソースタイプパッケージを生成し直すことができます。

RTR ファイルは、`install-directory/rt-name/etc` ディレクトリに置かれます。RTR ファイルは、普通のテキストエディタで編集できます。RTR ファイルの詳細は、[34 ページの「リソースとリソースタイププロパティの設定」](#)を参照してください。プロパティについては、[付録 A](#)を参照してください。

## ▼ コマンド行バージョンの Agent Builder を使用する 方法

コマンド行バージョンの Agent Builder でも、GUI と同様の基本手順を使用します。ただし、GUI では情報を入力しましたが、コマンド行インタフェースでは `scdscreate` や `scdsconfig` コマンドに引数を渡します。詳細は、`scdscreate(1HA)` と `scdsconfig(1HA)` のマニュアルページを参照してください。

コマンド行バージョンの Agent Builder の使用方法は次のとおりです。

- 1 アプリケーションに高可用性またはスケーラビリティを持たせるため、`scdscreate` を使って **Sun Cluster** リソースタイプテンプレートを作成します。
- 2 `scdsconfig` を使って、`scdscreate` で作成したリソースタイプテンプレートを構成します。  
プロパティ変数を指定できます。プロパティ変数については、[181 ページの「プロパティ変数の使用」](#)を参照してください。
- 3 作業ディレクトリの `pkg` サブディレクトリに移動します。
- 4 `pkgadd` コマンドを実行して、`scdscreate` で作成したパッケージをインストールします。  

```
# pkgadd -d . package-name
```
- 5 (省略可能)生成されたソースコードを編集します。
- 6 起動スクリプトを実行します。

## Agent Builder で作成されるディレクトリ構造

Agent Builder は、ターゲットリソースタイプ用に生成するすべてのファイルを格納するためのディレクトリ構造を作成します。「作成」画面で作業ディレクトリを指定します。開発するリソースタイプごとに異なるインストールディレクトリを指定する必要があります。Agent Builder は、作業ディレクトリの下に、ベンダー名とリソースタイプ名を連結した名前を持つサブディレクトリを作成します。たとえば、SUNW というベンダー名を指定し、ftp というリソースタイプを作成した場合、Agent Builder は SUNWftp というディレクトリを作業ディレクトリの下に作成します。

Agent Builder は、このサブディレクトリの下に、次のようなディレクトリを作成し、各ディレクトリにファイルを配置します。

ディレクトリ名	内容
bin	C 出力の場合、ソースファイルからコンパイルしたバイナリファイルが格納されます。Korn シェル出力の場合、src ディレクトリと同じファイルが格納されます。
etc	RTR ファイルが格納されます。Agent Builder は、ベンダー名とアプリケーション名をピリオド区切り (.) で結合して RTR ファイル名を作成します。たとえば、ベンダー名が SUNW で、リソースタイプ名が ftp である場合、RTR ファイル名は SUNW.ftp となります。
man	start、stop、および remove ユーティリティースクリプト用にカスタマイズされたマニュアルページが格納されます。たとえば、startftp(1M)、stopftp(1M)、および removeftp(1M) が格納されます。  これらのマニュアルページを見る場合は、man -M オプションでこのパスを指定します。たとえば、次のように使用します。  % man -M install-directory/SUNWftp/man removeftp
pkg	作成されたデータサービスが含まれる最終的な Solaris パッケージが格納されます。
src	Agent Builder によって生成されたソースファイルが格納されます。
util	Agent Builder によって生成された start、stop、および remove ユーティリティースクリプトが格納されます。188 ページの「Sun Cluster Agent Builder で作成されるユーティリティースクリプトとマニュアルページ」を参照してください。Agent Builder は、これらのスクリプト名にアプリケーション名を追加します。たとえば、startftp、stopftp、および removeftp のようになります。

# Agent Builder の出力

この節では、Agent Builder の出力について説明します。

この章の内容は次のとおりです。

- 187 ページの「ソースファイルとバイナリファイル」
- 188 ページの「Sun Cluster Agent Builder で作成されるユーティリティースクリプトとマニュアルページ」
- 190 ページの「Agent Builder で作成されるサポートファイル」
- 190 ページの「Agent Builder で作成されるパッケージディレクトリ」
- 190 ページの「rtconfig ファイル」

## ソースファイルとバイナリファイル

Resource Group Manager (RGM) は、リソースグループを管理し、最終的にはクラスタ上のリソースも管理します。RGM は、コールバックモデル上で動作します。つまり、特定のイベント(ノードまたはゾーンの障害など)が発生したとき、RGM は、当該ノードまたはゾーン上で動作しているリソースごとにリソースタイプのメソッドを呼び出します。たとえば、RGM は Stop メソッドを呼び出して、当該ノードまたはゾーン上で動作しているリソースを停止します。次に、Stop メソッドを呼び出して、異なるノードまたはゾーン上でリソースを起動します。このモデルについては、21 ページの「RGM モデル」、24 ページの「コールバックメソッド」、および `rt_callbacks(1HA)` のマニュアルページを参照してください。

このモデルをサポートするために Agent Builder は、8 つの実行可能 C プログラムまたは Korn シェルスクリプトを `install-directory/rt-name/bin` ディレクトリに生成します。これらのプログラムまたはシェルスクリプトは、コールバックメソッドとして機能します。

---

注 - 厳密には、障害モニターを実装する `rt-name_probe` プログラムはコールバックプログラムではありません。RGM は、`rt-name_probe` を直接呼び出すのではなく、`rt-name_monitor_start` と `rt-name_monitor_stop` を呼び出します。これらのメソッドが `rt-name_probe` を呼び出すことによって、障害モニターの起動と停止が行われます。

---

Agent Builder が生成する 8 つのメソッドは次のとおりです。

- `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`

各メソッドに固有な情報については、`rt_callbacks(1HA)` のマニュアルページを参照してください。

Agent Builder は、`install-directory/rt-name/src` ディレクトリ (C 出力の場合) に、次のファイルを作成します。

- ヘッダーファイル (`rt-name.h`)
- すべてのメソッドに共通するコードが記述されているソースファイル (`rt-name.c`)
- 共通するコード用のオブジェクトファイル (`rt-name.o`)
- 各メソッド用のソースファイル (`*.c`)
- 各メソッド用のオブジェクトファイル (`*.o`)

Agent Builder は、`rt-name.o` ファイルを各メソッドの `.o` ファイルにリンクして、実行可能ファイルを `install-directory/rt-name/bin` ディレクトリに作成します。

Korn シェル出力の場合、`install-directory/rt-name/bin` ディレクトリと `install-directory/rt-name/src` ディレクトリの内容は同じです。それぞれのディレクトリには、7つのコールバックメソッドと Probe メソッドに対応する 8つの実行可能スクリプトが含まれています。

---

注 - Korn シェル出力には、2つのコンパイル済みユーティリティプログラム `gettime` と `gethostnames` が含まれています。これらのプログラムは、特定のコールバックメソッドが時間の取得や、検証を行う際に必要です。

---

ソースコードを編集して、`make` コマンドを実行すると、コードを再コンパイルできます。さらに、再コンパイル後、`make pkg` コマンドを実行すると、新しいパッケージを生成できます。ソースコードの修正をサポートするために、Agent Builder はソースコード中の適切な場所に、コードを追加するためのコメント文を埋め込みます。[184 ページの「生成されたソースコードの編集」](#)を参照してください。

## Sun Cluster Agent Builder で作成されるユーティリティスクリプトとマニュアルページ

リソースタイプを生成し、そのパッケージをクラスタにインストールしたあと、リソースタイプのインスタンス (リソース) をクラスタ上で実行する必要があります。一般に、リソースインスタンスを実行するには、管理コマンドまたは Sun Cluster Manager を使用します。しかし、便宜上 Agent Builder はこの目的のためにカスタマイズされたユーティリティスクリプトに加え、ターゲットリソースタイプのリソースの停止と削除を行うスクリプトも生成します。

これら3つのスクリプトは *install-directory/rt-name/util* ディレクトリに格納されており、次のような処理を行います。

- 起動スクリプト: リソースタイプを登録し、必要なリソースグループとリソースを作成します。また、アプリケーションがネットワーク上のクライアントと通信するためのネットワークアドレスリソース (LogicalHostname または SharedAddress) も作成します。
- 停止スクリプト: リソースを停止します。
- 削除スクリプト: 起動スクリプトによる作業を取り消します。つまり、このスクリプトは、リソース、リソースグループ、ターゲットリソースタイプを停止し、システムから削除します。

---

注- これらのスクリプトは、内部的な規則を使用してリソースとリソースグループの名前付けを行います。このため、削除スクリプトを使用できるリソースは、対応する起動スクリプトで起動されたリソースだけです。

---

Agent Builder は、スクリプト名にアプリケーション名を追加することにより、スクリプトの名前付けを行います。たとえば、アプリケーション名が ftp の場合、各スクリプトは startftp、stopftp、および removeftp になります。

Agent Builder は、各ユーティリティスクリプト用のマニュアルページを *install-directory/rt-name/man/man1m* ディレクトリに格納します。これらのマニュアルページにはスクリプトに渡す必要がある引数についての説明が記載されているので、各スクリプトを起動する前に、これらのマニュアルページをお読みください。

これらのマニュアルページを表示するには、**-M** オプションを指定して **man** コマンドを実行し、この **man** ディレクトリのパスを指定する必要があります。たとえば、ベンダーが SUNW で、アプリケーション名が ftp である場合、startftp(1M) のマニュアルページを表示するには、次のコマンドを使用します。

```
% man -M install-directory/SUNWftp/man startftp
```

クラスタ管理者は、マニュアルページユーティリティスクリプトも利用できます。Agent Builder で生成されたパッケージをクラスタ上にインストールすると、ユーティリティスクリプト用のマニュアルページは、*/opt/rt-name/man* ディレクトリに格納されます。たとえば、startftp(1M) のマニュアルページを表示するには、次のコマンドを使用します。

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

## Agent Builder で作成されるサポートファイル

Agent Builder は、サポートファイル (`pkginfo`、`postinstall`、`postremove`、`preremove` など) を `install-directory/rt-name/etc` ディレクトリに格納します。このディレクトリには、Resource Type Registration (RTR) ファイルも格納されます。RTR ファイルは、ターゲットリソースタイプが利用できるリソースとリソースタイププロパティーを宣言して、リソースをクラスタに登録するときにプロパティー値を初期化します。詳細は、34 ページの「リソースとリソースタイププロパティーの設定」を参照してください。RTR ファイルの名前は、ベンダー名とリソースタイプ名をピリオドで区切って連結したものです (たとえば、`SUNW.ftp`)。

RTR ファイルは、ソースコードを再コンパイルしなくても、標準のテキストエディタで編集および修正できます。ただし、`make pkg` コマンドを使用してパッケージを再構築する必要があります。

## Agent Builder で作成されるパッケージディレクトリ

`install-directory/rt-name/pkg` ディレクトリには、Solaris パッケージが格納されます。パッケージの名前は、ベンダー名とアプリケーション名を連結したものです (たとえば、`SUNWftp`)。 `install-directory/rt-name/src` ディレクトリ内の `Makefile` は、新しいパッケージを作成するのに役立ちます。たとえば、ソースファイルを修正し、コードを再コンパイルした場合、あるいは、パッケージユーティリティスクリプトを修正した場合、`make pkg` コマンドを使用して新しいパッケージを作成します。

パッケージをクラスタから削除する場合、複数のノードから同時に `pkgrm` コマンドを実行しようとする、コマンドが失敗する可能性があります。

この問題を解決するには、次の2つの方法があります。

- クラスタの1つのノードで `remove rt-name` スクリプトを実行してから、任意のノードで `pkgrm` コマンドを実行します。
- クラスタの1つのノードで `pkgrm` コマンドを実行して、必要なクリーンアップをすべて行います。続いて、残りのノードで (必要であれば同時に) `pkgrm` コマンドを実行します。

同時に複数のノードから `pkgrm` を実行しようとして失敗した場合は、1つのノードでこのコマンドを実行し、その後残りのノードで実行します。

## rtconfig ファイル

C または Korn シェルソースコードを作業ディレクトリ内に生成する場合、Agent Builder は構成ファイル `rtconfig` を生成します。このファイルには、「作成」画面と「構成」画面でユーザーが入力した情報が含まれます。既存のリソースタイプ用

の作業ディレクトリから Agent Builder を起動すると、Agent Builder は `rtconfig` ファイルを読み取り、既存のリソースタイプに指定された情報を「作成」画面と「構成」画面に表示します。また、「ファイル」ドロップダウンメニューから「リソースタイプのロード」を選択して既存のリソースタイプをロードしても、Agent Builder は類似した動作を示します。この機能は、既存のリソースタイプのクローンを作成したい場合に便利です。183 ページの「Agent Builder で作成したコードの再利用」を参照してください。

## Agent Builder の Cluster Agent モジュール

Agent Builder の Cluster Agent モジュールは、NetBeans™ モジュールです。このモジュールには、Sun Java Studio (以前の Sun ONE Studio) 製品全般の Sun Cluster ソフトウェア用のリソースタイプを作成できる GUI が付属しています。

---

注 - Sun Java Studio 製品の設定、インストール、使用の詳細は、Sun Java Studio マニュアルに記載されています。このマニュアルは、<http://www.sun.com/software/sundev/jde/documentation/index.html> Web サイトより参照できます。

---

### ▼ Cluster Agent モジュールをインストールし設定する方法

Cluster Agent モジュールは、Sun Cluster ソフトウェアのインストール時にインストールされます。Sun Cluster のインストールツールは、Cluster Agent モジュールファイルを `/usr/cluster/lib/scdsbuilder` の `scdsbuilder.jar` に配置します。Sun Java Studio ソフトウェアで Cluster Agent モジュールを使用するには、このファイルに対してシンボリックリンクを作成する必要があります。

---

注 - Cluster Agent モジュールを実行する予定のシステムには、Sun Cluster 製品、Sun Java Studio 製品、および Java 1.4 がすでにインストールされ、使用可能な状況でなければなりません。

---

- 1 ユーザー全員が Cluster Agent モジュールを使用できるようにするか、あるいは自分だけが使用できるようにします。
  - ユーザー全員が使用できるようにするには、スーパーユーザーになるか、RBAC 承認 `solaris.cluster.modify` を提供する役割を使用し、シンボリックリンクをグローバルモジュールディレクトリに作成します。

```
# cd /opt/s1studio/ee/modules
# ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

---

注 - Sun Java Studio ソフトウェアを `/opt/s1studio/ee` 以外のディレクトリにすでにインストールしてある場合は、このディレクトリパスを、使用したパスに読み替えてください。

---

- 自分だけが使用できるようにするには、自分の `modules` サブディレクトリにシンボリックリンクを作成します。

```
% cd ~your-home-dir/ffjuser40ee/modules
% ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

- 2 Sun Java Studio ソフトウェアを停止し、再起動します。

## ▼ Cluster Agent モジュールを起動する方法

次に、Sun Java Studio ソフトウェアから Cluster Agent モジュールを起動する手順を示します。

---

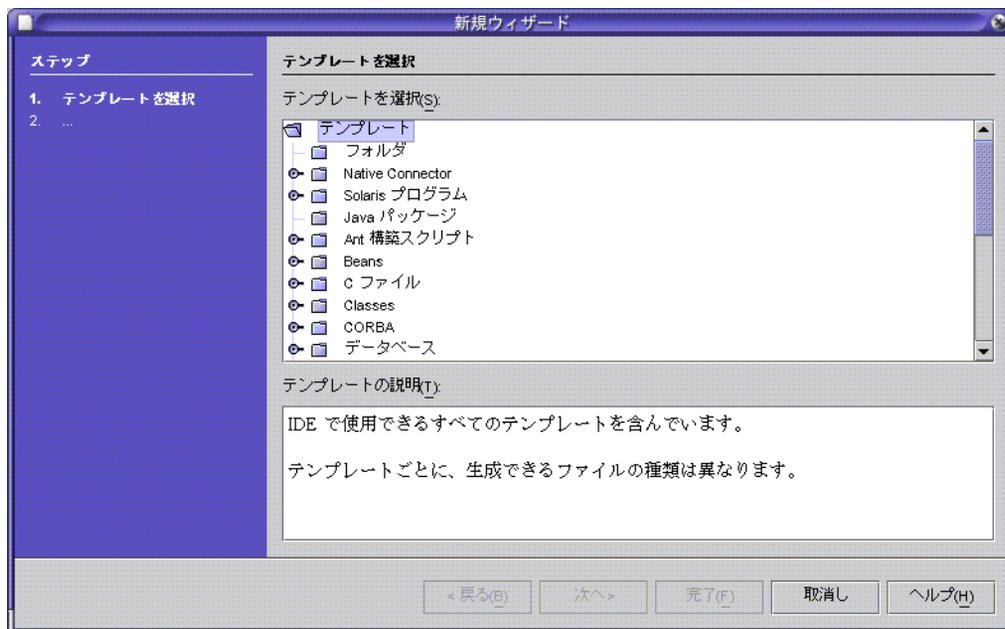
注 - Sun Java Studio 製品の設定、インストール、使用の詳細は、Sun Java Studio マニュアルに記載されています。このマニュアルは、<http://www.sun.com/software/sundev/jde/documentation/index.html> Web サイトより参照できます。

---

- 1 Sun Java Studio の「ファイル」メニューから「新規」を選択するか、あるいはツールバーの「新規」アイコンをクリックします。



「新規ウィザード」画面が表示されます。



- 「テンプレートを選択」区画で、必要に応じて下方向へスクロールし、「その他」フォルダの横に表示されている鍵マークをクリックします。

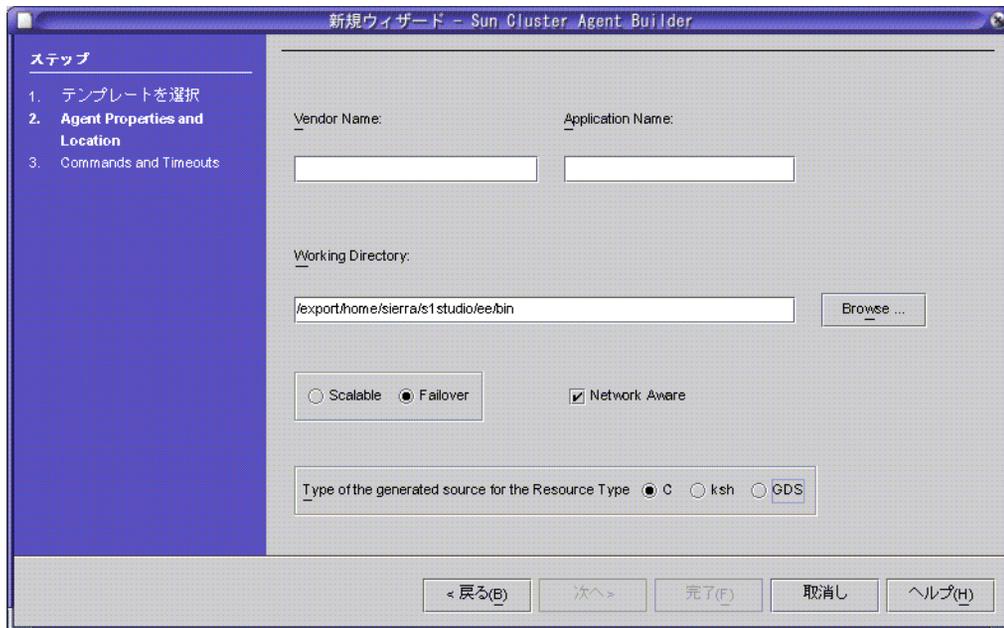
  その他

「その他」フォルダが開きます。



- 「その他」フォルダから **Sun Cluster Agent Builder** を選択し、「次へ」をクリックします。

Sun Java Studio 起動のための Cluster Agent モジュール最初の「新規ウィザード - Sun Cluster Agent Builder」画面が表示されます。



## Cluster Agent モジュールの使用

Cluster Agent モジュールは、Agent Builder ソフトウェアと同様に使用できます。インタフェースは英語版の Agent Builder ソフトウェアと全く同じです。たとえば次の図では、英語版 Agent Builder ソフトウェアの「Create」画面と Cluster Agent モジュールの最初の「新規ウィザード - Sun Cluster Agent Builder」画面には同じフィールドと選択肢が存在することがわかります。

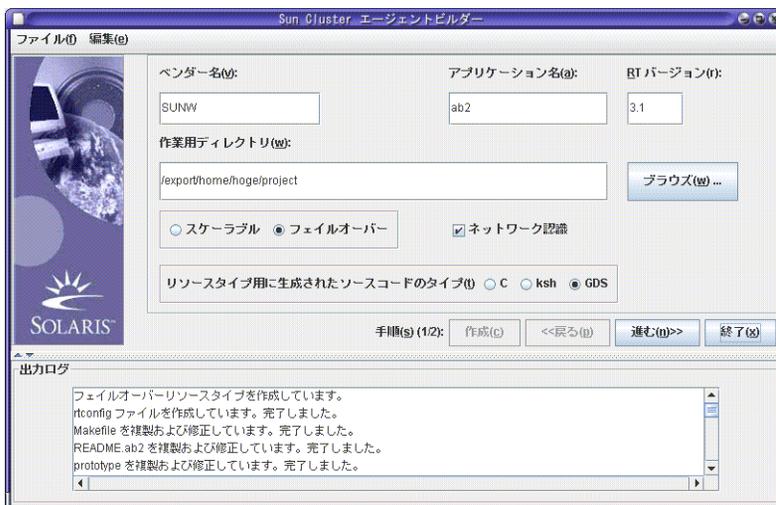


図 9-4 英語版 Agent Builder ソフトウェアの作成画面

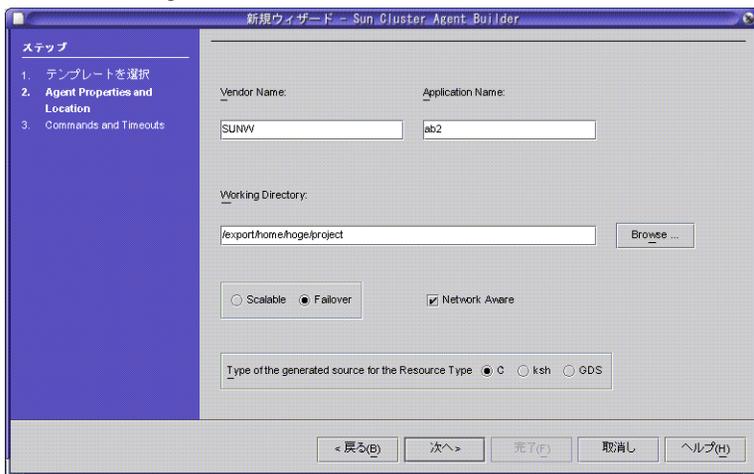


図 9-5 Cluster Agent モジュールの「新規ウィザード - Sun Cluster Agent Builder」画面

## Cluster Agent モジュールと Agent Builder の違い

Cluster Agent モジュールと Agent Builder は似ていますが、小さな違いがいくつかあります。

- Cluster Agent モジュールでは、2つ目の「新規ウィザード - Sun Cluster Agent Builder」画面で「完了」をクリックした時点でリソースタイプの作成と構成が完了します。最初の「新規ウィザード - Sun Cluster Agent Builder」画面で「次へ」をクリックした時点ではリソースタイプは作成されません。

Agent Builder では、「作成」画面で「作成」をクリックした時点でリソースタイプがただちに作成されます。また、「構成」画面で「構成」をクリックした時点でリソースタイプがただちに構成されます。

- 英語版 Agent Builder の「Output Log」領域に表示される情報は、Sun Java Studio 製品では別のウィンドウで表示されます。

# 汎用データサービス

---

この章では、汎用データサービス (GDS) の概要を述べてから、GDS を使用するサービスの作成方法について説明します。このサービスの作成には、Sun Cluster Agent Builder、または Sun Cluster 管理コマンドを使用します。

この章の内容は次のとおりです。

- 197 ページの「GDS の概念」
- 204 ページの「Agent Builder を使って、GDS を使用するサービスを作成」
- 210 ページの「Sun Cluster 管理コマンドを使って、GDS を使用するサービスを作成」
- 212 ページの「Agent Builder のコマンド行インタフェース」

## GDS の概念

GDS とは、簡単なネットワーク対応や非ネットワーク対応のアプリケーションを高可用性にしたり、スケラブルにしたりするための機構です。そのためには、これらのアプリケーションを Sun Cluster Resource Group Management (RGM) フレームワークに組み込みます。この機構では、アプリケーションの可用性やスケラビリティを高めるために一般的に行う必要がある、データサービスのコーディングは必要ありません。

GDS ベースのデータサービスを非大域ゾーンで実行するように構成できるのは、関連するアプリケーションも非大域ゾーンで実行するように構成する場合です。

GDS は、あらかじめコンパイルされた単一のデータサービスです。コールバックメソッド (rt\_callbacks) の実装やリソースタイプ登録ファイル (rt\_reg) など、コンパイル済みのデータサービスやそのコンポーネントを変更することはできません。

この節の内容は、次のとおりです。

- コンパイル済みリソースタイプ
- GDS を使用することの利点と欠点

- GDS を使用するサービスの作成方法
- GDS によるイベントのロギング
- 必須の GDS プロパティ
- 任意の GDS プロパティ

## コンパイル済みリソースタイプ

汎用データサービスのリソースタイプ `SUNW.gds` は、`SUNWscgds` パッケージに含まれています。このパッケージは、クラスタのインストール時に `scinstall` コーティリティーでインストールされます。`scinstall(1M)` のマニュアルページを参照してください。`SUNWscgds` パッケージには次のファイルが格納されています。

```
# 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
```

## GDS を使用することの利点と欠点

GDS を使用すると、Agent Builder のソースコード (`scdscreate(1HA)` のマニュアルページを参照) や Sun Cluster 管理コマンドを使用するのに比べ、次の利点があります。

- GDS は使い易いデータサービスです。
- GDS とそのメソッドはコンパイル済みであるため、変更できません。
- Agent Builder を使って、アプリケーション用のスクリプトを生成できます。これらのスクリプトは、複数のクラスタで再利用できる Solaris パッケージになっています。

GDS を使用すると多くの利点もありますが、GDS 機構の使用が適さない場合もあります。

- コンパイル済みリソースタイプを使用する場合よりも高度な制御が必要な場合。たとえば拡張プロパティを追加する場合や、デフォルト値を変更する場合など
- 特別な機能を追加するためにソースコードを変更する必要がある場合

## GDS を使用するサービスの作成方法

GDS を使用するサービスの作成方法は2通りあります。

- Agent Builder
- Sun Cluster 管理コマンド

### GDS と Agent Builder

Agent Builder を使用し、生成するソースコードのタイプとして GDS を選択します。特定のアプリケーションのリソースを設定するスクリプト群を生成するためにユーザーの入力が必要です。

### GDS と Sun Cluster 管理コマンド

この方法では、SUNWscgds に含まれているコンパイル済みデータサービスコードを使用します。ただし、クラスタ管理者は、Sun Cluster 管理コマンドを使ってリソースの作成と構成を行う必要があります。詳細は、clresource(1CL) のマニュアルページを参照してください。

### GDS ベースのサービスを作成する方法の選択

Sun Cluster のコマンドを発行するためには相当量の入力作業が必要になります。たとえば、210 ページの「Sun Cluster 管理コマンドを使って GDS ベースの高可用性サービスを作成する方法」や211 ページの「Sun Cluster 管理コマンドを使って GDS ベースのスケラブルサービスを作成する方法」を参照してください。

GDS と Agent Builder を使用する方法では、この処理が簡単になります。この方法では、生成されるスクリプトがユーザーに代わって scrgadm と scswitch コマンドを出力するからです。

## GDS によるイベントのロギング

GDS を使用すると、GDS から渡される関連情報を、GDS が起動するスクリプトにロギングできます。この情報には、起動、検証、確認、停止の各メソッドの状態やプロパティ変数が含まれます。この情報を使ってスクリプトの問題やエラーを診断したり、この情報をほかの目的に適用することができます。

GDS でロギングすべきメッセージのレベル(つまり、タイプ)の指定には、Log\_level プロパティを使用します(詳細は202 ページの「Log\_level プロパティ」を参照)。NONE、INFO、ERR を指定できます。

## GDS ログファイル

次の2つのGDSログファイルは、ディレクトリ `/var/cluster/logs/DS/resource-group-name/resource-name` に配置されています。

- `start_stop_log.txt` には、リソース起動メソッドや停止メソッドによって生成されるメッセージが含まれています。
- `probe_log.txt` には、リソースモニターによって生成されるメッセージが含まれています。

`start_stop_log.txt` に含まれる情報のタイプを、次の例に示します。

```
06/12/2006 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
06/12/2006 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

`probe_log.txt` に含まれる情報のタイプを、次の例に示します。

```
06/12/2006 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
06/12/2006 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
06/12/2006 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
06/12/2006 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

## 必須のGDSプロパティ

アプリケーションがネットワーク対応の場合は、`Start_command` 拡張プロパティと `Port_list` プロパティの両方を指定する必要があります。アプリケーションがネットワーク非対応の場合は、`Start_command` 拡張プロパティだけを指定します。

### Start\_command プロパティ

`Start_command` 拡張プロパティで指定する起動コマンドによって、アプリケーションが起動されます。このコマンドは、引数を備えたUNIXコマンドでなければなりません。コマンドは、アプリケーションを起動するシェルに直接渡すことができます。

### Port\_list プロパティ

`Port_list` プロパティは、アプリケーションが待機するポートのリストを指定します。`Port_list` プロパティは、Agent Builder によって生成される起動スクリプトか、`clresource` コマンド (Sun Cluster 管理コマンドを使用する場合) に指定されていなければなりません。

## 任意の GDS プロパティ

任意の GDS プロパティには、「システム定義プロパティ」と「拡張プロパティ」の両方が含まれます。システム定義プロパティは、Sun Cluster により提供されるプロパティの標準セットです。RTR ファイルで定義されているプロパティは、拡張プロパティと呼ばれます。

任意の GDS プロパティには次のものがあります。

- Child\_mon\_level 拡張プロパティ (管理コマンドでのみ使用)
- Failover\_enabled 拡張プロパティ
- Log\_level 拡張プロパティ
- Network\_resources\_used プロパティ
- Probe\_command 拡張プロパティ
- Probe\_timeout 拡張プロパティ
- Start\_timeout プロパティ
- Stop\_command 拡張プロパティ
- Stop\_signal 拡張プロパティ
- Stop\_timeout プロパティ
- Validate\_command 拡張プロパティ
- Validate\_timeout プロパティ

### Child\_mon\_level プロパティ

---

注 - Sun Cluster 管理コマンドを使用する場合は、Child\_mon\_level プロパティを使用できます。Agent Builder を使用する場合は、このプロパティは使用できません。

---

このプロパティは、プロセスモニター機能 (PMF) を通じて監視されるプロセスを制御します。このプロパティは、フォークされた子プロセスをどのようなレベルで監視するかを表します。このプロパティは、pmfadm コマンドの -c 引数と同等の働きをします。詳細は、pmfadm(1M) のマニュアルページを参照してください。

このプロパティを省略するか、このプロパティにデフォルト値の -1 を指定することは、pmfadm コマンドで --c オプションを省略するのと同じ効果があります。つまり、すべての子プロセスとその子孫プロセスが監視されます。

### Failover\_enabled プロパティ

この拡張プロパティは、リソースのフェイルオーバー動作を制御します。この拡張プロパティに TRUE を設定すると、アプリケーションは、再起動回数が Retry\_interval 秒間に Retry\_count を超えるとフェイルオーバーされます。

このプロパティに FALSE を設定すると、再起動回数が Retry\_interval 秒間に Retry\_count を超えてもアプリケーションの再起動や、別のノードまたはゾーンへのフェイルオーバーは行われません。

このプロパティを使用すると、アプリケーションリソースによるリソースグループのフェイルオーバーを防ぐことができます。このプロパティのデフォルト値は TRUE です。

---

注 - Failover\_mode のほうがフェイルオーバー動作をよりよく制御できるので、将来的には Failover\_enabled 拡張プロパティの代わりに Failover\_mode プロパティを使用します。詳細は、r\_properties(5) のマニュアルページで、Failover\_mode の LOG\_ONLY および RESTART\_ONLY の値の説明を参照してください。

---

## Log\_level プロパティ

このプロパティは、GDS によって記録される診断メッセージのレベル(つまり、タイプ)を指定します。このプロパティには、NONE、INFO、または ERR を指定できます。NONE を指定すると、診断メッセージはロギングされません。INFO を指定すると、情報メッセージだけがロギングされます。ERR を指定すると、エラーメッセージだけがロギングされます。デフォルトでは、診断メッセージはロギングされません (NONE)。

## Network\_resources\_used プロパティ

このプロパティは、リソースによって使用される論理ホスト名と共有アドレスネットワークリソースのリストを指定します。このプロパティのデフォルト値は null です。アプリケーションを1つ以上の特定のアドレスにバインドする必要がある場合は、このプロパティを指定してください。このプロパティを省略するか Null を指定すると、アプリケーションはすべてのアドレスで待機します。

GDS リソースを作成する前には、LogicalHostname または SharedAddress リソースがすでに構成されている必要があります。LogicalHostname または SharedAddress リソースの構成方法については、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』を参照してください。

値を指定する場合は、1つまたは複数のリソース名を指定します。個々のリソース名には、1つ以上の LogicalHostname リソースか1つ以上の SharedAddress リソースを含めることができます。詳細は、r\_properties(5) のマニュアルページを参照してください。

## Probe\_command プロパティ

このプロパティは、特定のアプリケーションの状態を周期的にチェックする検証コマンドを指定します。このコマンドは、引数を備えた UNIX コマンドでなければなりません。コマンドは、アプリケーションを検証するシェルに直接渡されます。アプリケーションが正常に実行されていれば、検証コマンドは終了ステータスとして 0 を返します。

検証コマンドの終了ステータスは、アプリケーションの障害の重大度を判断するために使用されます。終了状態 (検証状態) は、0 (正常) から 100 (全面的な障害) までの整数である必要があります。検証ステータスは、特殊な値として 201 をとることができます。この場合、アプリケーションは、Failover\_enabled が FALSE に設定されている場合を除き、直ちにフェイルオーバーされます。GDS 検証アルゴリズムは、この検証ステータスを使って、アプリケーションをローカルに再起動するか、フェイルオーバーするかを決定します。詳細は、scds\_fm\_action(3HA) のマニュアルページを参照してください。終了ステータスが 201 の場合には、アプリケーションは直ちにフェイルオーバーされます。

検証コマンドを省略すると、GDS はそれ自身の簡単な検証を行います。この検証は、Network\_resources\_used プロパティや scds\_get\_netaddr\_list() 関数の出力から得られる一連の IP アドレスに対してアプリケーションに接続します。詳細は、scds\_get\_netaddr\_list(3HA) のマニュアルページを参照してください。接続に成功すると、接続が直ちに切断されます。接続と切断が両方とも成功すれば、アプリケーションは正常に動作しているものとみなされます。

---

注 - GDS で提供される検証は、全機能を備えたアプリケーション固有の検証の単純な代替物ではありません。

---

### Probe\_timeout プロパティ

このプロパティは、検証コマンドのタイムアウト値を指定します。詳細は、[202 ページの「Probe\\_command プロパティ」](#)を参照してください。Probe\_timeout のデフォルトは 30 秒です。

### Start\_timeout プロパティ

このプロパティは、起動コマンドの起動タイムアウトを指定します。詳細は、[200 ページの「Start\\_command プロパティ」](#)を参照してください。Start\_timeout のデフォルトは 300 秒です。

### Stop\_command プロパティ

このプロパティは、アプリケーションを停止し、アプリケーションが完全に停止したあとでのみ戻る必要があるコマンドを指定します。このコマンドは、アプリケーションを停止するシェルに直接渡すことができる完全な UNIX コマンドでなければなりません。

Stop\_command 拡張プロパティが指定されていると、GDS 停止メソッドは、停止タイムアウトの 80% を指定して停止コマンドを起動します。さらに、GDS 停止メソッドは、停止コマンドの起動結果がどうであれ、停止タイムアウトの 15% を指定して SIGKILL を送信します。タイムアウトの残り 5% は、処理のオーバーヘッドのために使用されます。

停止コマンドが省略されていると、GDS は、`Stop_signal` に指定されたシグナルを使ってアプリケーションを停止しようとします。

### Stop\_signal プロパティ

このプロパティは、PMF を通じてアプリケーションを停止するためのシグナルを識別する値を指定します。指定可能な整数値のリストについては、[signal\(3HEAD\)](#) のマニュアルページを参照してください。デフォルト値は 15 です (SIGTERM)。

### Stop\_timeout プロパティ

このプロパティは、停止コマンドのタイムアウトを指定します。詳細は、[203 ページの「Stop\\_command プロパティ」](#)を参照してください。Stop\_timeout のデフォルトは 300 秒です。

### Validate\_command プロパティ

このプロパティは、アプリケーションを検証するために呼び出されるコマンドへの絶対パスを指定します。絶対パスを指定しない場合、アプリケーションは検証されません。

### Validate\_timeout プロパティ

このプロパティは、検証コマンドのタイムアウトを指定します。詳細は、[204 ページの「Validate\\_command プロパティ」](#)を参照してください。Validate\_timeout のデフォルトは 300 秒です。

## Agent Builder を使って、GDS を使用するサービスを作成

Agent Builder を使って、GDS を使用するサービスを作成できます。Agent Builder の詳細については、[第 9 章](#)を参照してください。

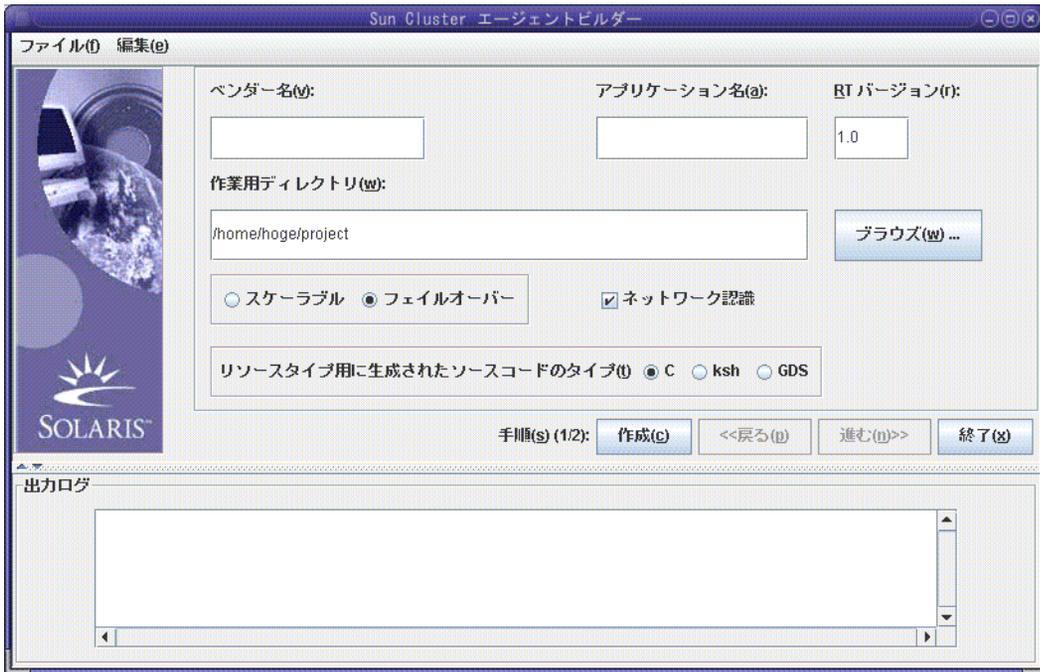
## GDS ベースのスクリプトの作成と構成

### ▼ Agent Builder を起動し、スクリプトを作成する

- 1 スーパーユーザーになるか、RBAC 承認 `solaris.cluster.modify` を提供する役割になります。
- 2 Agent Builder を起動します。

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

### 3 「Agent Builder Create」画面が表示されます。



- 4 ベンダー名を入力します。
- 5 アプリケーション名を入力します。

---

注 - Solaris 9 OS 以降では、ベンダー名とアプリケーション名の両方を合わせて 10 文字以上を指定できます。この組み合わせは、スクリプトのパッケージ名として使用されます。

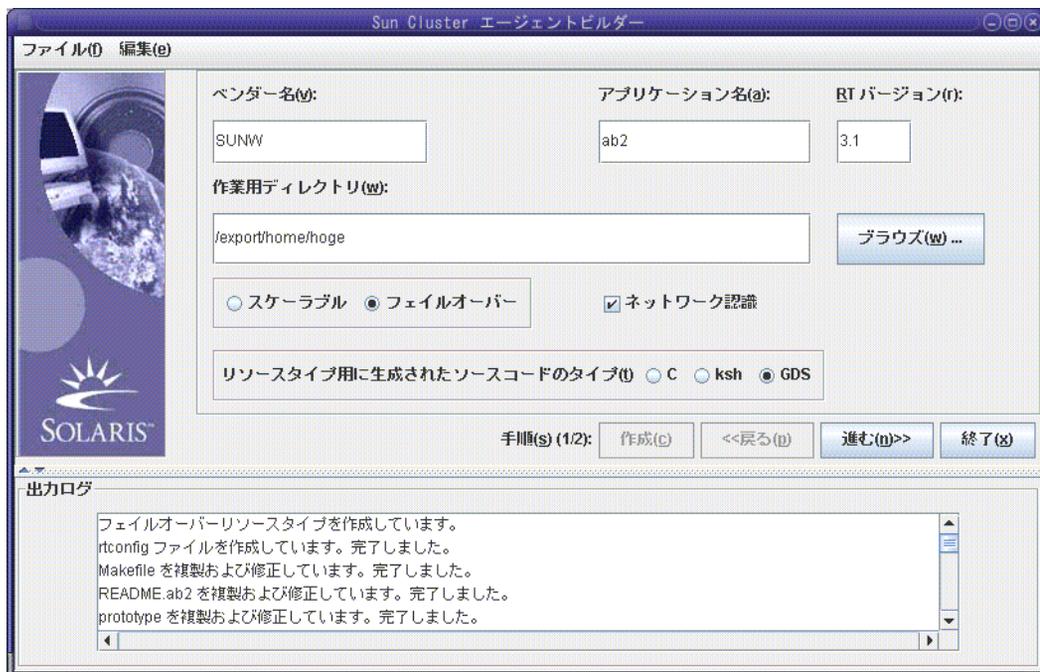
---

- 6 作業ディレクトリに移動します。  
パスを入力する代わりに、「ブラウズ」ドロップダウンメニューを使ってディレクトリを選択することもできます。
- 7 データサービスがスケラブルなのかフェイルオーバーなのかを選択します。  
GDS を作成するときには「ネットワーク認識」がデフォルトですので、これを選択する必要はありません。
- 8 「GDS」を選択します。
- 9 (省略可能) 表示されているデフォルト値から RT バージョンを変更します。

注- 「RTバージョン」フィールドで次の文字を使用することはできません。空白文字、タブ、スラッシュ (/)、バックスラッシュ (\)、アスタリスク (\*)、疑問符 (?)、コンマ (,)、セミコロン (;)、左角括弧 ( (、右角括弧 ( ) )。

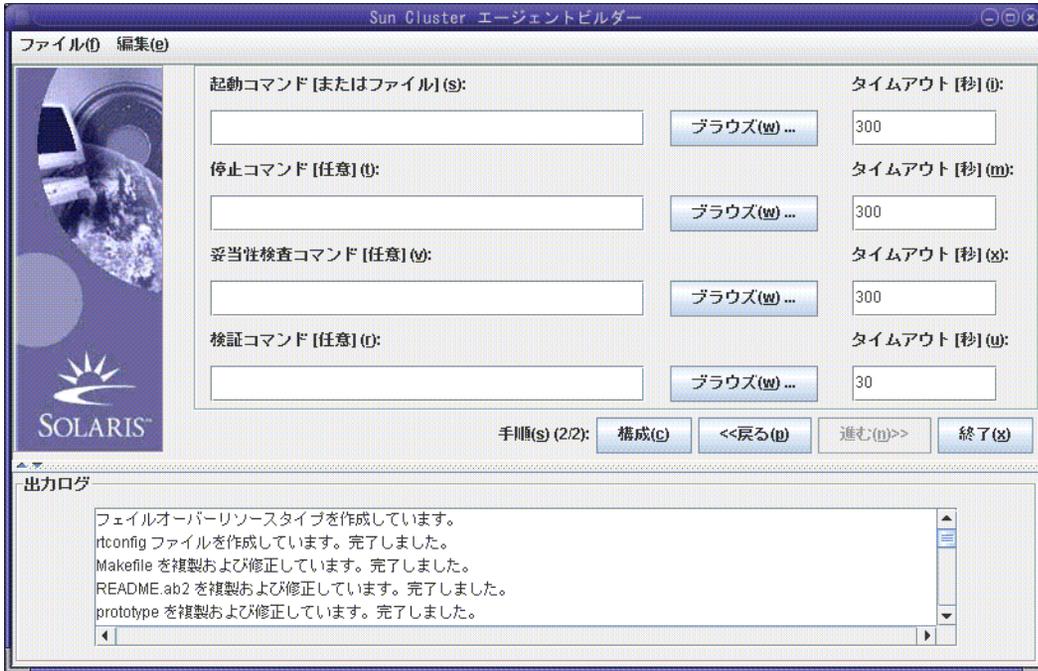
- 10 「作成」 をクリックします。

Agent Builder により、スクリプトが作成されます。結果が「出力ログ」領域に表示されます。



「作成」 ボタンがグレー表示されていることに注意してください。これで、スクリプトの構成を始めることができます。

- 11 「Next」 をクリックします。  
「構成」 画面が表示されます。



## ▼ スクリプトを構成する方法

スクリプトの作成が終わったら、新しいサービスを構成する必要があります。

- 1 起動コマンドの場所を入力するか、「ブラウズ」をクリックして起動コマンドの場所を指定します。  
プロパティー変数を指定できます。プロパティー変数については、181 ページの「[プロパティー変数の使用](#)」を参照してください。
- 2 (省略可能) 停止コマンドの場所を入力するか、「ブラウズ」をクリックして停止コマンドの場所を指定します。  
プロパティー変数を指定できます。プロパティー変数については、181 ページの「[プロパティー変数の使用](#)」を参照してください。
- 3 (省略可能) 確認コマンドの場所を入力するか、「ブラウズ」をクリックして確認コマンドの場所を指定します。  
プロパティー変数を指定できます。プロパティー変数については、181 ページの「[プロパティー変数の使用](#)」を参照してください。

- 4 (省略可能) 検証コマンドの場所を入力するか、「ブラウズ」をクリックして検証コマンドの場所を指定します。  
プロパティー変数を指定できます。プロパティー変数については、[181 ページの「プロパティー変数の使用」](#)を参照してください。
- 5 (省略可能) 起動、停止、確認、検証コマンドの新しいタイムアウト値を指定します。
- 6 「構成」をクリックします。  
Agent Builder によりスクリプトが構成されます。

---

注 - Agent Builder は、ベンダー名とアプリケーション名を連結してパッケージ名を作成します。

---

スクリプトのパッケージが作成され、次のディレクトリに置かれます。

*working-dir/vendor-name-application/pkg*

たとえば、*/export/wdir/NETapp/pkg* のようになります。

- 7 クラスタの各ノード上で、スーパーユーザーになるか、**RBAC 承認** `solaris.cluster.modify` を提供する役割になります。
- 8 クラスタの各ノード上で、完成したパッケージをインストールします。

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

pkgadd によって以下のファイルがインストールされます。

```
/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-マニュアルページとスクリプト名は、以前に「Create」画面で入力したアプリケーション名の前にスクリプト名を付けたものに対応します (たとえば、startapp のようになります)。

---

- 9 クラスタのいずれかのノードでリソースを構成し、アプリケーションを起動します。

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

startapp スクリプトの引数は、リソースのタイプがフェイルオーバーかスケーラブルかで異なります。

---

注-入力する必要があるコマンド行を判別するには、カスタマイズしたマニュアルページを検査するか、startapp スクリプトを引数なしで実行して使用法の説明文を表示してください。

マニュアルページを表示するには、マニュアルページへのパスを指定する必要があります。たとえば、startapp(1M) のマニュアルページを表示する場合は、次のように入力します。

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

使用法の説明文を表示するには、次のように入力します。

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

---

## Agent Builder からの出力

Agent Builder は3つのスクリプトと、パッケージ作成時の入力に基づく構成ファイルを生成します。構成ファイルには、リソースグループとリソースタイプの名前が指定されます。

4つのスクリプトは次のとおりです。

- 起動スクリプト: リソースを構成し、RGM の制御下にあるアプリケーションを起動します。
- 停止スクリプト: アプリケーションを停止し、リソースやリソースグループを停止します。
- 削除スクリプト: 起動スクリプトによって作成されたリソースやリソースグループを削除します。

これらのスクリプトのインタフェースや動作は、Agent Builder によって非 GDS ベースのデータサービス用に生成されるユーティリティスクリプトのものと同じです。これらのスクリプトは、複数のクラスタで再利用できる Solaris パッケージに含まれています。

構成ファイルをカスタマイズすれば、(通常は `scrgadm` および `scswitch` コマンドへの引数として指定される) リソースグループやその他の引数の独自の名前を指定できます。スクリプトをカスタマイズしないと、Agent Builder がこれらの引数に対しデフォルト値を設定します。

## Sun Cluster 管理コマンドを使って、GDS を使用するサービスを作成

この節では、GDS に引数をどのように入力するかについて説明します。既存の Sun Cluster 管理コマンド、たとえば `clresourcetype`、`clresourcegroup`、`clresource` などを使って、GDS の保守と管理を行います。

スクリプトが適切な機能を提供している場合は、この節で述べる低位レベルの管理コマンドを使用する必要はありません。ただし、GDS ベースのリソースをより細かく制御する必要がある場合は、低位レベルの管理コマンドを使用できます。これらのコマンドはスクリプトによって実行されます。

### ▼ Sun Cluster 管理コマンドを使って GDS ベースの高可用性サービスを作成する方法

- 1 スーパーユーザーになるか、RBAC 承認 `solaris.cluster.modify` を提供する役割になります。
- 2 リソースタイプ `SUNW.gds` を登録します。

```
# clresourcetype register SUNW.gds
```

- 3 LogicalHostname リソースとフェイルオーバーサービス自体を含むリソースグループを作成します。  

```
# clresourcegroup create haapp_rg
```
- 4 LogicalHostname リソースのリソースを作成します。  

```
# clreslogicalhostname create -g haapp_rg hhead
```
- 5 フェイルオーバーサービス自体のリソースを作成します。  

```
# clresource create -g haapp_rg -t SUNW.gds
  -p Validate_command="/export/app/bin/configtest" \
  -p Scalable=false -p Start_timeout=120 \
  -p Stop_timeout=120 -p Probe_timeout=120 \
  -p Port_list="2222/tcp" \
  -p Start_command="/export/ha/appctl/start" \
  -p Stop_command="/export/ha/appctl/stop" \
  -p Probe_command="/export/app/bin/probe" \
  -p Child_mon_level=0 -p Network_resources_used=hhead \
  -p Failover_enabled=TRUE -p Stop_signal=9 haapp_rs
```
- 6 リソースグループ haapp\_rg を、管理された状態でオンラインにします。  

```
# clresourcegroup online -M haapp_rg
```

## ▼ Sun Cluster 管理コマンドを使って GDS ベースのスケラブルサービスを作成する方法

- 1 スーパーユーザーになるか、RBAC 承認 solaris.cluster.modify を提供する役割になります。
- 2 リソースタイプ SUNW.gds を登録します。  

```
# clresourcetype register SUNW.gds
```
- 3 SharedAddress リソースのリソースグループを作成します。  

```
# clresourcegroup create sa_rg
```
- 4 SharedAddress リソース hhead をリソースグループ sa\_rg 内に作成します。  

```
# clressharedaddress create -g sa_rg hhead
```
- 5 スケラブルサービスのリソースグループを作成します。  

```
# clresourcegroup create -S -p RG_dependencies=sa_reg app_rg
```

- 6 スケーラブルサービスのリソースを作成します。
 

```
# clresource create -g app_rg -t SUNW.gds
  -p Validate_command="/export/app/bin/configtest" \
  -p Scalable=TRUE -p Start_timeout=120 \
  -p Stop_timeout=120 -p Probe_timeout=120 \
  -p Port_list="2222/tcp" \
  -p Start_command="/export/app/bin/start" \
  -p Stop_command="/export/app/bin/stop" \
  -p Probe_command="/export/app/bin/probe" \
  -p Child_mon_level=0 -p Network_resource_used=hhead \
  -p Failover_enabled=TRUE -p Stop_signal=9 app_rs
```
- 7 ネットワークリソースを含むリソースグループをオンラインにします。
 

```
# clresourcegroup online sa_reg
```
- 8 リソースグループ app\_rg を、管理された状態でオンラインにします。
 

```
# clresourcegroup online -M app_reg
```

## Agent Builder のコマンド行インタフェース

Agent Builder には、GUI が提供するのと同じ機能を提供するコマンド行インタフェースが組み込まれています。コマンド行インタフェースは `scdscreate` と `scdsconfig` コマンドからなります。詳細は、`scdscreate(1HA)` と `scdsconfig(1HA)` のマニュアルページを参照してください。

### ▼ コマンド行バージョンの Agent Builder を使って、GDS を使用するサービスを作成する

この節では、204 ページの「Agent Builder を使って、GDS を使用するサービスを作成」と同じ手順を、コマンド行インタフェースを使ってどのように実行するかについて説明します。

- 1 スーパーユーザーになるか、RBAC 承認 `solaris.cluster.modify` を提供する役割になります。
- 2 サービスを作成します。
  - フェイルオーバーサービスの場合:
 

```
# scdscreate -g -V NET -T app -d /export/wdir
```
  - スケーラブルサービスの場合:
 

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

---

注--d引数は任意です。この引数を指定しないと、現在のディレクトリが作業ディレクトリになります。

---

### 3 サービスを構成します。

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

プロパティー変数を指定できます。プロパティー変数については、[181 ページの「プロパティー変数の使用」](#)を参照してください。

---

注- 起動コマンド (scdsconfig -s) のみ必須です。ほかのオプションと引数はすべて任意です。

---

### 4 クラスタの各ノード上で、完成したパッケージをインストールします。

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

pkgadd によって以下のファイルがインストールされます。

```
/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- マニュアルページとスクリプト名は、以前に「Create」画面で入力したアプリケーション名の前にスクリプト名を付けたものに対応します (たとえば、startapp のようになります)。

---

### 5 クラスタのいずれかのノードでリソースを構成し、アプリケーションを起動します。

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

startapp スクリプトの引数は、リソースのタイプがフェイルオーバーかスケーラブルかで異なります。

---

注- 入力する必要があるコマンド行を判別するには、カスタマイズしたマニュアルページを検査するか、startapp スクリプトを引数なしで実行して使用法の説明文を表示してください。

マニュアルページを表示するには、マニュアルページへのパスを指定する必要があります。たとえば、startapp(1M) のマニュアルページを表示する場合は、次のように入力します。

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

使用法の説明文を表示するには、次のように入力します。

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

```
The resource name of LogicalHostname or SharedAddress must be specified.
```

```
For failover services:
```

```
Usage: startapp -h logicalhostname  
       -p port-and-protocol-list  
       [-n ipmpgroup/adapter-list]
```

```
For scalable services:
```

```
Usage: startapp -h shared-address-name  
       -p port-and-protocol-list  
       [-l load-balancing-policy]  
       [-n ipmpgroup/adapter-list]  
       [-w load-balancing-weights]
```

---

## DSDL API 関数

---

この章では、データサービス開発ライブラリ (Data Service Development Library: DSDL) の API 関数の一覧を示し、概要を述べます。個々の DSDL 関数の詳細については、そのマニュアルページ (3HA) を参照してください。DSDL は、C インタフェースのみを提供します。スクリプトベースの DSDL インタフェースはありません。

この章の内容は次のとおりです。

- 215 ページの「汎用関数」
- 217 ページの「プロパティ関数」
- 217 ページの「ネットワークリソースアクセス関数」
- 219 ページの「PMF 関数」
- 220 ページの「障害監視関数」
- 220 ページの「ユーティリティ関数」

### 汎用関数

このカテゴリの関数は、さまざまな機能を提供します。

これらの関数では、次の操作を行うことができます。

- DSDL 環境を初期化します。
- リソースタイプ、リソース、リソースグループの名前と、拡張プロパティの値を取得します。
- リソースグループをフェイルオーバーおよび再起動し、リソースを再起動します。
- エラー文字列をエラーメッセージに変換します。
- タイムアウトを適用してコマンドを実行します。

## 初期化関数

次の関数は、呼び出しメソッドを初期化します。

- `scds_initialize(3HA)` – リソースを割り当て、DSDL環境を初期化します。
- `scds_close(3HA)` – `scds_initialize()` によって割り当てられたリソースを解放します。

## 取得関数

次の関数は、ゾーン、リソースタイプ、リソース、リソースグループ、および拡張プロパティについての情報を取得します。

- `scds_get_zone_name(3HA)` – 自身の代わりにメソッドが実行されているゾーンの名前を取得します。
- `scds_get_resource_type_name(3HA)` – 呼び出しプログラム用のリソースタイプの名前を取得します。
- `scds_get_resource_name(3HA)` – 呼び出しプログラム用のリソースの名前を取得します。
- `scds_get_resource_group_name(3HA)` – 呼び出しプログラム用のリソースグループの名前を取得します。
- `scds_get_ext_property(3HA)` – 指定された拡張プロパティの値を取得します。
- `scds_free_ext_property(3HA)` – `scds_get_ext_property()` によって割り当てられたメモリを解放します。

次の関数は、リソースによって使用される `SUNW.HASStoragePlus` リソースの状態情報を取得します。

`scds_hasp_check(3HA)` – リソースによって使用される `SUNW.HASStoragePlus` リソースの状態情報を取得します。当該リソース用に定義されている `Resource_dependencies` または `Resource_dependencies_weak` のシステム属性を使用することによって、当該リソースが依存しているすべての `SUNW.HASStoragePlus` リソース状態 (オンラインであるか、オンラインでないか) についての情報が得られます。詳細は、`SUNW.HASStoragePlus(5)` のマニュアルページを参照してください。

## フェイルオーバー関数と再起動関数

次の関数は、リソースまたはリソースグループをフェイルオーバーまたは再起動します。

- `scds_failover_rg(3HA)` – リソースグループをフェイルオーバーします。
- `scds_restart_rg(3HA)` – リソースグループを再起動します。
- `scds_restart_resourc(3HA)` – リソースを再起動します。

## 実行関数

次の関数は、タイムアウトを適用してコマンドを実行し、エラーコードをエラーメッセージに変換します。

- `scds_timerun(3HA)` - タイムアウト値のもとでコマンドを実行します。
- `scds_error_string(3HA)` および `scds_error_string_i18n(3HA)` - エラーコードを、エラー文字列に変換します。`scds_error_string()` から返された文字列は、英語で表示されます。`scds_error_string_i18n()` から返された文字列は、LC\_MESSAGES ロケールカテゴリで指定されているその国および地域の言語によって表示されます。
- `scds_svc_wait(3HA)` - 指定されたタイムアウト期間、監視プロセスが終了するのを待ちます。

## プロパティ関数

このカテゴリの関数は、関連するリソースタイプ、リソース、およびリソースグループ(よく使用される一部の拡張プロパティも含む)に固有のプロパティにアクセスするのに便利なAPIを提供します。DSDLは、`scds_initialize()` を使用してコマンド行引数を解析します。`scds_initialize(3HA)` 関数は、関連するリソースタイプ、リソース、およびリソースグループのさまざまなプロパティをキャッシュに入れます。

次の関数を始めとするこれらの関数の説明は、`scds_property_functions(3HA)` のマニュアルページにあります。

- `scds_get_ext_property-name`
- `scds_get_rg_property-name`
- `scds_get_rs_property-name`
- `scds_get_rt_property-name`

## ネットワークリソースアクセス関数

このカテゴリの関数は、リソースとリソースグループが使用するネットワークリソースの取得、出力、解放を行います。ここで説明する `scds_get_` 関数は、RMAPI関数を使用して `Network_resources_used` や `Port_list` などのプロパティを照会しなくても、ネットワークリソースを取得できる便利な方法を提供します。`scds_print_name()` 関数は、`scds_get_name()` 関数から戻されたデータ構造から値を出力します。`scds_free_name()` 関数は、`scds_get_name()` 関数が割り当てたメモリーを解放します。

## ホスト名関数

次の関数はホスト名を扱います。

- `scds_get_rs_hostnames(3HA)` – リソースによって使用されているホスト名のリストを取得します。
- `scds_get_rg_hostnames(3HA)` – リソースグループ内のネットワークリソースによって使用されているホスト名のリストを取得します。
- `scds_print_net_list(3HA)` – ホスト名リストの内容を `syslog(3C)` に書き込みます。この関数は、一般的にデバッグ目的で使用します。
- `scds_free_net_list(3HA)` – `scds_get_rs_hostnames()` または `scds_get_rg_hostnames()` によって割り当てられたメモリーを解放します。

## ポートリスト関数

次の関数はポートリストを扱います。

- `scds_get_port_list(3HA)` – リソースによって使用されているポート/プロトコルのペアのリストを取得します。
- `scds_print_port_list(3HA)` – ポートプロトコルリストの内容を `syslog(3C)` に書き込みます。100%!!!You typically use this function for debugging.
- `scds_free_port_list(3HA)` – `scds_get_port_list()` によって割り当てられたメモリーを解放します。

## ネットワークアドレス関数

次の関数はネットワークアドレスを扱います。

- `scds_get_netaddr_list(3HA)` – リソースによって使用されているネットワークアドレスのリストを取得します。
- `scds_print_netaddr_list(3HA)` – ネットワークアドレスリストの内容を `syslog(3C)` に書き込みます。100%!!!You typically use this function for debugging.
- `scds_free_netaddr_list(3HA)` – `scds_get_netaddr_list()` によって割り当てられたメモリーを解放します。

## TCP 接続関数を使用する障害監視

このカテゴリの関数は、TCP ベースの監視を行います。通常、障害モニターはこれらの関数を使用して、サービスとの単純ソケット接続を確立し、サービスのデータを読み書きしてサービスの状態を確認したあと、サービスとの接続を切断します。

これらの関数には以下が含まれます。

- `scds_fm_tcp_connect(3HA)` – IPv4 アドレッシングだけを使用するプロセスへの TCP 接続を確立します。
- `scds_fm_net_connect(3HA)` – IPv4 か IPv6 アドレッシングのどちらかを使用するプロセスへの TCP 接続を確立します。
- `scds_fm_tcp_read(3HA)` – TCP 接続を使って、監視されているプロセスからデータを読み取ります。
- `scds_fm_tcp_write(3HA)` – TCP 接続を使って、監視されているプロセスにデータを書き込みます。
- `scds_simple_probe(3HA)` – プロセスへの TCP 接続を確立し、停止することによってプロセスを検証します。この関数は IPv4 アドレスだけを扱います。
- `scds_simple_net_probe(3HA)` – プロセスへの TCP 接続を確立し、停止することによってプロセスを検証します。この関数は、IPv4 または IPv6 アドレスを扱います。
- `scds_fm_tcp_disconnect(3HA)` – 監視されているプロセスへの接続を停止します。この関数は IPv4 アドレスだけを扱います。
- `scds_fm_net_disconnect(3HA)` – 監視されているプロセスへの接続を停止します。この関数は、IPv4 または IPv6 アドレスを扱います。

## PMF 関数

この関数は、Process Monitor Facility (PMF) 機能をカプセル化します。PMF 経由の監視における DSDL モデルは、`pmfadm` に対して暗黙のタグ値を作成および使用します。詳細は、`pmfadm(1M)` のマニュアルページを参照してください。

また、PMF 機能は、`Restart_interval`、`Retry_count`、および `action_script` 用の暗黙値も使用します (`pmfadm` の `-t`、`-n`、および `-a` オプション)。もっとも重要な点は、DSDL が PMF によって検出されたプロセス障害履歴を、障害モニターによって検出されたアプリケーション障害履歴に結びつけ、再起動またはフェイルオーバーのどちらを行うかを決定することです。

このカテゴリには次のような関数があります。

- `scds_pmf_get_status(3HA)` – 指定するインスタンスが PMF 制御のもとで監視されているかどうかを判別します。
- `scds_pmf_restart_fm(3HA)` – PMF を使って障害モニターを再起動します。
- `scds_pmf_signal(3HA)` – PMF 制御のもとで動作するプロセスツリーに、指定するシグナルを送信します。

- `scds_pmf_start(3HA)`および`scds_pmf_star(3HA)` – PMF 制御のもと、障害監視をはじめとする指定されたプログラムを実行します。`scds_pmf_start_env()` 関数は、`scds_pmf_start()` 関数と同じ処理の実行に加え、実行されるプログラムに指定された環境も渡します。
- `scds_pmf_stop(3HA)` – PMF 制御のもとで動作しているプロセスを停止します。
- `scds_pmf_stop_monitoring(3HA)` – PMF 制御のもとで動作しているプロセスの監視を停止します。

## 障害監視関数

このカテゴリの関数は、障害履歴を保持し、その履歴を `Retry_count` および `Retry_interval` プロパティと関連付けて評価することにより、障害監視の事前定義モデルを提供します。

このカテゴリには次のような関数があります。

- `scds_fm_sleep(3HA)` – 障害モニター制御ソケットに関するメッセージを待ちます。
- `scds_fm_action(3HA)` – 検証の完了後にアクションをとります。
- `scds_fm_print_probes(3HA)` – 検証状態の情報をシステムログに書き込みます。

## ユーティリティー関数

以下の関数は、メッセージやデバッグ用メッセージをシステムログに書き込むために使用します。

- `scds_syslog(3HA)` – メッセージをシステムログに書き込みます。
- `scds_syslog_debug(3HA)` – デバッグメッセージをシステムログに書き込みます。

## クラスタ再構成通知プロトコル

---

この章では、Cluster Reconfiguration Notification Protocol (CRNP) について説明します。CRNPを使用することで、フェイルオーバー用のアプリケーションや拡張性のあるアプリケーションを「クラスタ対応」として設定できます。具体的には、Sun Cluster 再構成イベントにアプリケーションを登録し、それらのイベントの後続の非同期通知を受け取ることができます。イベント通知の受信登録が可能なのは、クラスタの内部で動作するデータサービスと、クラスタの外部で動作するアプリケーションです。イベントは、クラスタ内のメンバーシップに変化があった場合と、リソースグループまたはリソースの状態に変化があった場合に生成されます。

---

注-SUNW.Eventのリソースタイプ実装は、高可用性を備えたSun ClusterのCRNPサービスを提供します。このリソース型実装については、SUNW.Event(5)のマニュアルページを参照してください。

---

この章の内容は次のとおりです。

- 221 ページの「CRNP の概念」
- 225 ページの「クライアントをサーバーに登録する方法」
- 227 ページの「クライアントに対するサーバーの応答方法」
- 229 ページの「サーバーがクライアントにイベントを配信する方法」
- 232 ページの「CRNPによるクライアントとサーバーの認証」
- 233 ページの「CRNPを使用するJavaアプリケーションの作成例」

### CRNP の概念

CRNP は、標準の7層 OSI (Open System Interconnect) プロトコルスタックにおけるアプリケーション層、プレゼンテーション層、およびセッション層を定義します。トランスポート層はTCPでなければならず、ネットワーク層はIPでなければなりません。CRNPは、データリンク層および物理層とは無関係です。CRNP内で交換されるアプリケーション層メッセージはすべて、XML 1.0をベースとしたものです。

注-CRNP は、大域ゾーンでのみ実行できます。

## CRNP の動作

CRNP は、クラスタ再構成イベント生成、クラスタへの配信、それらのイベントを要求しているクライアントへの送信を行うメカニズムとデーモンを提供します。

クライアントとの通信を行うのは、cl\_apid デーモンです。クラスタ再構成イベントの生成は、Sun Cluster Resource Group Manager (RGM) によって行われます。このデーモンは、syseventd を使用して各ローカルノードにイベントを転送します。cl\_apid デーモンは、TCP/IP 上で XML (Extensible Markup Language) を使用して要求クライアントとの通信を行います。

次の図は、CRNP コンポーネント間のイベントの流れを簡単に示したものです。この図では、一方のクライアントはクラスタノード2で動作し、他方のクライアントはクラスタに属していないコンピュータ上で動作しています。

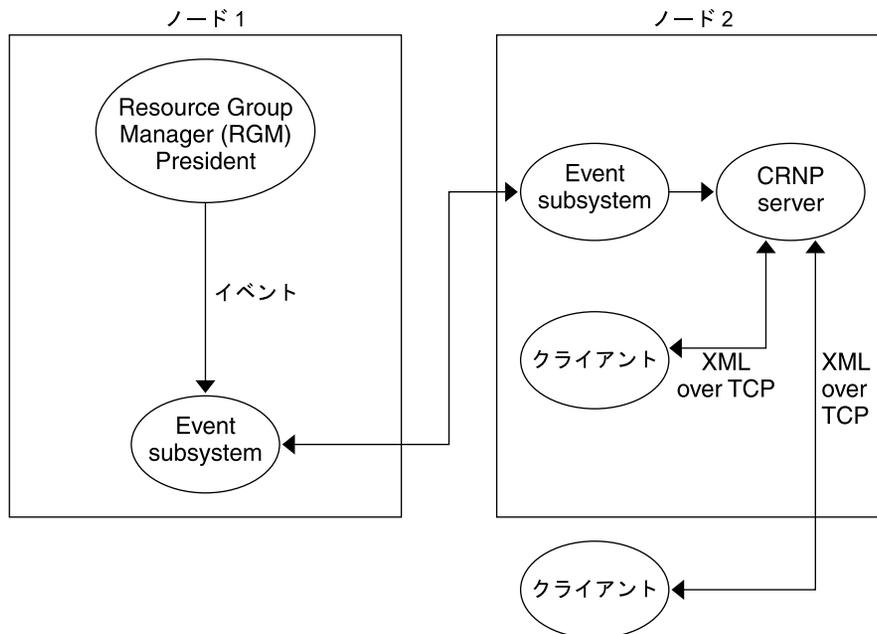


図 12-1 CRNP コンポーネント間のイベントの流れ

## CRNP のセマンティクス

クライアントは、サーバーへ登録メッセージ (SC\_CALLBACK\_RG) を送信することによって通信を開始します。この登録メッセージは、通知を受信したいイベントタイプと、イベントの配信先として使用できるポートを指定するものです。登録用接続のソース IP と指定ポートから、コールバックアドレスが構成されます。

クライアントが配信を希望しているイベントがクラスタ内で生成されるたびに、サーバーはこのコールバックアドレス (IP とポート) を持つクライアントと通信を行い、イベント (SC\_EVENT) をクライアントに配信します。サーバーには、そのクラスタ内で稼動している高可用マシンが使用されます。サーバーは、クラスタの再起動後も維持されるストレージにクライアントの登録情報を格納します。

登録解除を行う場合、クライアントはサーバーに登録メッセージ (REMOVE\_CLIENT メッセージが入った SC\_CALLBACK\_RG) を送信します。サーバーから SC\_REPLY メッセージを受け取ったあとで、クライアントは接続を閉じることができます。

次の図は、クライアントとサーバー間の通信の流れを示します。

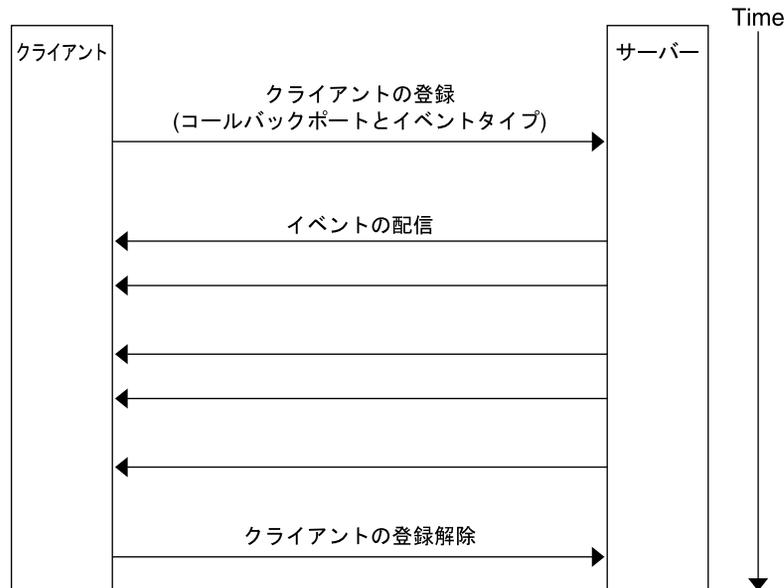


図 12-2 クライアントとサーバー間の通信の流れ

## CRNP メッセージのタイプ

CRNP は、3 種類の XML ベースのメッセージを使用します。次の表に、これらのメッセージの使用法を示します。これらのメッセージタイプの内容と使用法の詳細は、この章で後述します。

CRNP メッセージのタイプ	説明
SC_CALLBACK_REG	<p>このメッセージには、4 つのフォーム、ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS、およびREMOVE_EVENTS を指定できます。これらの各フォームには、次の情報が含まれます。</p> <ul style="list-style-type: none"> <li>■ プロトコルバージョン</li> <li>■ ASCII 形式で示されたコールバックポート (バイナリ形式ではない)</li> </ul> <p>ADD_CLIENT、ADD_EVENTS、およびREMOVE_EVENTS には、バインドされていないイベントタイプリストも含まれます。これらの各フォームには、次の情報が含まれます。</p> <ul style="list-style-type: none"> <li>■ イベントクラス</li> <li>■ イベントサブクラス (省略可能)</li> <li>■ 名前と値がペアになったリスト (省略可能)</li> </ul> <p>イベントクラスとイベントサブクラスにより一意の「イベントタイプ」が定義されます。SC_CALLBACK_REG のクラスを生成する DID (ドキュメントタイプ定義) は、SC_CALLBACK_REG です。この DTD の詳細は、<a href="#">付録 F</a> を参照してください。</p>
SC_REPLY	<p>このメッセージには次の情報が含まれます。</p> <ul style="list-style-type: none"> <li>■ プロトコルバージョン</li> <li>■ エラーコード</li> <li>■ エラーメッセージ:</li> </ul> <p>SC_REPLY のクラスを生成する DTD は SC_REPLY です。この DTD の詳細は、<a href="#">付録 F</a> を参照してください。</p>
SC_EVENT	<p>このメッセージには次の情報が含まれます。</p> <ul style="list-style-type: none"> <li>■ プロトコルバージョン</li> <li>■ イベントクラス</li> <li>■ イベントサブクラス</li> <li>■ ベンダー</li> <li>■ バブリッシャー</li> <li>■ 名前と値のペアリスト (名前と値をペアにした 0 個以上のデータ構造) <ul style="list-style-type: none"> <li>■ 名前 (文字列)</li> <li>■ 値 (文字列または文字列配列)</li> </ul> </li> </ul> <p>SC_EVENT 内の値はタイプとしては分類されていません。SC_EVENT のクラスを生成する DTD は SC_EVENT です。この DTD の詳細は、<a href="#">付録 F</a> を参照してください。</p>

# クライアントをサーバーに登録する方法

この節では、サーバーの設定、クライアントの識別、アプリケーション層とセッション層での情報送信、エラー状況などについて説明します。

## 管理者によるサーバー設定の前提

システム管理者は、汎用 IP アドレス (クラスタ内の特定のマシン専用でない IP アドレス) とポート番号を使用してサーバーを構成し、クライアントとなるマシンにこのネットワークアドレスを公開する必要があります。CRNP では、クライアントがこのサーバー名をどのように取得するかは定義されていません。管理者は、ネーミングサービスを使用することも (この場合、クライアントは動的にサーバーのネットワークアドレスを検出できる)、ネットワーク名を構成ファイルに追加してクライアントに読み取らせることもできます。サーバーは、クラスタ内でフェイルオーバーリソースタイプとして動作します。

## サーバーによるクライアントの識別方法

各クライアントは、そのコールバックアドレス (IP アドレスとポート番号) で識別されます。ポートは `SC_CALLBACK_REG` メッセージで指定され、IP アドレスは登録用の TCP 接続から取得されます。CRNP は、同じコールバックアドレスを持つ後続の `SC_CALLBACK_REG` メッセージは同じクライアントから送信されたと想定します。これは、メッセージの送信元であるソースポートが異なる場合でも同様です。

## クライアントとサーバー間での `SC_CALLBACK_REG` メッセージの受け渡し方法

クライアントは、サーバーの IP アドレスとポート番号に対して TCP 接続を開くことによって登録を開始します。TCP 接続が確立され書き込みの用意ができたところで、クライアントはその登録メッセージを送信する必要があります。この登録メッセージは正しい書式の `SC_CALLBACK_REG` メッセージでなければならず、メッセージの前後に余分なバイトを含めることはできません。

バイトがすべてストリームに書き込まれたあと、クライアントはサーバーから応答を受け取ることができるように接続をオープン状態に保つ必要があります。クライアントが不正な書式のメッセージを送信した場合、サーバーはそのクライアントを登録せず、クライアントに対してエラー応答を送信します。しかし、サーバーが応答を送信する前にクライアントがソケット接続を閉じた場合、サーバーはそのクライアントを正常なクライアントとして登録します。

クライアントは、いつでもサーバーと通信を行うことができます。サーバーと通信を行うごとに、クライアントは `SC_CALLBACK_REG` メッセージを送信する必要があります。書式が不正なメッセージ、順不同のメッセージ、無効なメッセージなどを受け取った場合、サーバーはクライアントにエラー応答を送信します。

クライアントは、それ自体が `ADD_CLIENT` メッセージを送信するまでは `ADD_EVENTS`、`REMOVE_EVENTS`、`REMOVE_CLIENT` メッセージを送信できません。クライアントは、それ自体が `ADD_CLIENT` メッセージを送信しないかぎり `REMOVE_CLIENT` メッセージを送信できません。

クライアントが `ADD_CLIENT` メッセージを送信したが、そのクライアントがすでに登録されていたという場合は、サーバーがこのメッセージを黙認することがあります。このような場合、サーバーは報告なしに古いクライアント登録を削除し、2つめの `ADD_CLIENT` メッセージに指定された新しいクライアント登録に置き換えます。

通常、クライアントはその起動時に `ADD_CLIENT` メッセージを送信することによって、サーバーに一度だけ登録を行います。また、登録の解除もサーバーに `REMOVE_CLIENT` メッセージを送信して一度だけ行います。しかし、CRNP はクライアントが必要に応じてイベントタイプリストを動的に変更できるだけの柔軟性を備えています。

## SC\_CALLBACK\_REG メッセージの概念

`ADD_CLIENT`、`REMOVE_CLIENT`、`ADD_EVENTS`、および `REMOVE_EVENTS` メッセージには、それぞれイベントリストが含まれます。次の表は、CRNP が受け付けるイベントタイプを、必要となる名前と値のペアと共に示して説明しています。

クライアントが以下の作業のどちらか一方を行うと、サーバーはクライアントに通知することなくこれらのメッセージを無視します。

- まだ登録が行われていないイベントタイプを1つ以上指定する `REMOVE_EVENTS` メッセージを送信する
- 同じイベントタイプを2度登録する

クラスとサブクラス	名前と値のペア	説明
<code>EC_cluster</code>	必須: なし	クラスタメンバーシップの変更(ノードの停止またはクラスタへの結合)に関連するあらゆるイベントに登録する
<code>ESC_cluster_membership</code>	(省略可能) なし	
<code>EC_cluster</code>	次の条件で1つ必要:	<code>rg_name</code> (リソースグループ名)のあらゆる状態変更イベントに登録する
<code>ESC_cluster_rg_state</code>	<code>rg_name</code>	
	値のタイプ: 文字列 (省略可能) なし	

クラスとサブクラス	名前と値のペア	説明
EC_Cluster	次の条件で1つ必要:	r_name (リソース名) のあらゆる状態変更イベントに登録する
ESC_cluster_r_state	r_name 値のタイプ: 文字列 (省略可能)なし	
EC_Cluster なし	必須: なし (省略可能)なし	あらゆる Sun Cluster イベントに登録する

## クライアントに対するサーバーの応答方法

登録要求を受信したサーバーは、登録を処理したあと、クライアントが開いた TCP 接続上で SC\_REPLY メッセージを送信します。このあとサーバーは接続を閉じます。クライアントは、サーバーから SC\_REPLY メッセージを受信するまで TCP 接続をオープン状態に保つ必要があります。

クライアントは次のような作業を行います。

1. サーバーに対して TCP 接続を開きます。
2. 接続が「writable (書き込み可能)」になるまで待機します。
3. SC\_CALLBACK\_REG メッセージ (このメッセージには ADD\_CLIENT メッセージが入っている) を送信します。
4. サーバーから SC\_REPLY メッセージが到着するのを待機します。
5. サーバーから SC\_REPLY メッセージを受け取ります。
6. サーバーが接続を閉じたことを示すインジケータを受信します (ソケットから 0 バイトを読み取る)。
7. 接続を閉じます。

その後クライアントは以下の作業を行います。

1. サーバーに対して TCP 接続を開きます。
2. 接続が「writable (書き込み可能)」になるまで待機します。
3. SC\_CALLBACK\_REG メッセージ (このメッセージには REMOVE\_CLIENT メッセージが入っている) を送信します。
4. サーバーから SC\_REPLY メッセージが到着するのを待機します。
5. サーバーから SC\_REPLY メッセージを受け取ります。
6. サーバーが接続を閉じたことを示すインジケータを受信します (ソケットから 0 バイトを読み取る)。
7. 接続を閉じます。

クライアントから SC\_CALLBACK\_REG メッセージを受け取るたびに、サーバーは同じ接続に SC\_REPLY メッセージを送信します。このメッセージは、処理が正常に完了したか失敗したかを示すものです。SC\_REPLY メッセージの XML ドキュメントタイプ定義とこのメッセージ内で示されるエラーメッセージについては、[372 ページ](#)の「SC\_REPLY XML DTD」を参照してください。

## SC\_REPLY メッセージの内容

SC\_REPLY メッセージは、処理が正常に完了したか失敗したかを示します。このメッセージには、CRNP メッセージのバージョン、ステータスコード、およびステータスコードの詳細を説明したステータスメッセージが含まれます。次の表は、ステータスコードの値を説明しています。

ステータスコード	説明
OK	メッセージは正常に処理されました。
RETRY	一時的なエラーのためにクライアントの登録はサーバーに拒否されました。クライアントは別の引数を使用して登録をもう一度試す必要があります。
LOW_RESOURCE	クライアントのリソースが少ないため、クライアントはあとでもう一度試す必要があります。クラスタのクラスタ管理者は、クラスタのリソースを増やすこともできます。
SYSTEM_ERROR	重大な問題が発生しました。クラスタのクラスタ管理者に連絡してください。
FAIL	承認の失敗などの問題が発生し、登録が失敗しました。
MALFORMED	XML 要求の形式が正しくないため解析が失敗しました。
INVALID	XML 要求が無効です (XML 仕様を満たしていない)。
VERSION_TOO_HIGH	メッセージのバージョンが高すぎて、メッセージを正常に処理できませんでした。
VERSION_TOO_LOW	メッセージのバージョンが低すぎて、メッセージを正常に処理できませんでした。

## クライアントによるエラー状況の処理

通常、SC\_CALLBACK\_REG メッセージを送信するクライアントは登録の成功または失敗を知らせる応答を受け取ります。

しかし、クライアントが登録を試みる際にサーバーからの SC\_REPLY メッセージの送信を妨げるエラーが発生することがあります。この場合、エラーが発生する前に登録が正常に完了すること、登録が失敗すること、あるいは登録処理が行われないうまま終了することもあります。

サーバーはクラスタ内においてフェイルオーバー (高可用) サーバーとして機能する必要があるため、このエラー状況はサービスの終了を意味するわけではありません。実際、サーバーは新しく登録されたクライアントに対してすぐにイベント送信を開始できます。

これらの状況を修復するには、クライアントは次の作業を行う必要があります。

- SC\_REPLY メッセージを待機している登録用接続にアプリケーションレベルのタイムアウトを強制します(このあと、登録を再試行する必要があります)。
- イベントコールバックの登録を行う前に、イベント配信用のコールバック IP アドレスとポート番号で待機を開始します。クライアントは、登録確認メッセージとイベント配信を同時に待機することになります。確認メッセージを受信する前にイベントを受信し始めた場合は、クライアントはそのまま登録接続を閉じる必要があります。

## サーバーがクライアントにイベントを配信する方法

クラスタ内でイベントが生成されると、CRNP サーバーはそのタイプのイベントを要求したすべてのクライアントにイベントの配信を行います。この配信では、クライアントのコールバックアドレスに SC\_EVENT メッセージが送信されます。各イベントの配信は、新たな TCP 接続で行われます。

クライアントが ADD\_CLIENT メッセージまたは ADD\_EVENT メッセージが入った SC\_CALLBACK\_REG メッセージを通してイベントタイプの配信登録を行うと、サーバーはただちにクライアントに対してそのタイプの最新イベントを送信します。クライアントは、後続のイベントを送信するシステムの現在の状態を判断できます。

クライアントに対して TCP 接続を開始する際に、サーバーはその接続に SC\_EVENT メッセージを1つだけ送信します。サーバーは全二重通信を閉じます。

クライアントは次のような作業を行います。

1. サーバーが TCP 接続を開始するのを待機します。
2. サーバーからの着信接続を受け入れます。
3. サーバーから SC\_EVENT メッセージが到着するのを待機します。
4. サーバーからの SC\_EVENT メッセージを読み取ります。
5. サーバーが接続を閉じたことを示すインジケータを受信します (ソケットから 0 バイトを読み取る)。
6. 接続を閉じます。

すべてのクライアントが登録を終了した時点で、それらのクライアントはイベント配信のための着信接続を受け入れるために常にコールバックアドレス (IP アドレスとポート番号) で待機する必要があります。

クライアントとの通信に失敗してイベントを配信できなかった場合、サーバーはユーザーが設定してある回数と周期に従ってイベントの配信を繰り返し試みます。それらの試行がすべて失敗に終わった場合、そのクライアントはサーバーのクライアントリストから削除されます。イベントをそれ以上受け取るためには、クライアントは `ADD_CLIENT` メッセージが入った `SC_CALLBACK_REG` メッセージを別途送信して登録をもう一度行う必要があります。

## イベント配信の保証

クラスタ内では、クライアントごとに配信順序を守るという方法で、総合的にイベント生成を順序付けます。たとえば、クラスタ内でイベント A の生成後イベント B が生成された場合、クライアント X はイベント A を受け取ってからイベント B を受け取ります。しかし、全クライアントに対するイベント配信の全体的な順序付けは保持されません。つまり、クライアント Y はクライアント X がイベント A を受け取る前にイベント A と B の両方を受け取る可能性があります。この方法では、低速のクライアントのために全クライアントへの配信が停滞するということがありません。

サーバーが配信するイベントはすべて (サブクラス用の最初のイベントとサーバーエラーのあとに発生するイベントを除く)、クラスタが生成する実際のイベントに応答して発生します。ただし、クラスタで生成されるイベントを見逃すようなエラーが発生する場合は、サーバーは各イベントタイプの現在のシステム状態を示すイベントをそれらのイベントタイプごとに生成します。各イベントは、そのイベントタイプの配信登録を行なったクライアントに送信されます。

イベント配信は、「1回以上」というセマンティクスに従って行われます。つまり、サーバーは1台のクライアントに対して同じイベントを複数回送信できます。この許可は、サーバーが一時的に停止して復帰した際に、クライアントが最新の情報を受け取ったかどうかをサーバーが判断できないという場合に不可欠なものです。

## SC\_EVENT メッセージの内容

SC\_EVENT メッセージには、クラスタ内で生成されて SC\_EVENT XML メッセージ形式に合うように変換された実際のメッセージが入っています。次の表は、CRNP が配信するイベントタイプ (名前と値のペア、パブリッシャー、ベンダーなど) を説明したものです。

注 - `state_list` の配列要素は、`node_list` の配列要素と同期をとるように配置されます。つまり、`node_list` 配列内で最初に出現しているノードの状態は、`state_list` 配列の先頭に示されます。

`ev_` で始まるほかの名前や、それらの名前に関連した値が存在する場合がありますが、クライアントによる使用を意図したものではありません。

クラスとサブクラス	パブリッシャーとベンダー	名前と値のペア
EC_Cluster	パブリッシャー: rgm	名前: <code>node_list</code>
ESC_cluster_membership	ベンダー: SUNW	値のタイプ: 文字配列 名前: <code>state_list</code>  <code>state_list</code> には、ASCII 形式の数字だけが入っています。各数字は、クラスタにおけるそのノードの現在のインカーネーション番号を示します。この番号が前のメッセージで受信した番号と同じである場合、ノードとクラスタの関係は変化していません (離脱、結合、または再結合が行われていない)。インカーネーション番号が -1 の場合、ノードはクラスタのメンバーではありません。インカーネーション番号が負の値以外の数字である場合、ノードはクラスタのメンバーです。 値のタイプ: 文字配列
EC_Cluster	パブリッシャー: rgm	名前: <code>rg_name</code>
ESC_cluster_rg_state	ベンダー: SUNW	値のタイプ: 文字列 名前: <code>node_list</code> 値のタイプ: 文字配列 名前: <code>state_list</code>  <code>state_list</code> には、リソースグループの状態を示す文字列が入っています。有効な値は、 <code>scha_cmds(1HA)</code> コマンドで取得できる値です。 値のタイプ: 文字配列

クラスとサブクラス	パブリッシャーとベンダー	名前と値のペア
EC_Cluster	パブリッシャー: rgm	名前: r_name
ESC_cluster_r_state	ベンダー: SUNW	値のタイプ: 文字列 名前: node_list 値のタイプ: 文字配列 名前: state_list  state_list には、リソースの状態を示す文字列が入っています。有効な値は、scha_cmds(1HA) コマンドで取得できる値です。  値のタイプ: 文字配列

## CRNP によるクライアントとサーバーの認証

サーバーは、TCP ラッパーを使用してクライアントの認証を行います。この場合、登録メッセージのソース IP アドレス (これはイベントの配信先であるコールバック IP アドレスとしても使用される) がサーバー側の「許可されたユーザー」リストに含まれていなければなりません。ソース IP アドレスと登録メッセージが「拒否されたクライアント」リストに存在してはなりません。ソース IP アドレスと登録メッセージがリスト中に存在しない場合、サーバーは要求を拒否し、クライアントに対してエラー応答を返します。

サーバーが SC\_CALLBACK\_REG ADD\_CLIENT メッセージを受け取る場合、そのクライアントの後続の SC\_CALLBACK\_REG メッセージには最初のメッセージ内のものと同じソース IP アドレスが含まれていなければなりません。

この条件を満たさない SC\_CALLBACK\_REG を受信した場合、CRNP サーバーは次のどちらかを実行します。

- 要求を無視し、クライアントにエラー応答を送信する
- その要求が新しいクライアントからのものであると想定する (SC\_CALLBACK\_REG メッセージの内容にもとづいて判断)

このセキュリティーメカニズムは、正規クライアントの登録の解除を試みるサービス拒否攻撃の防止に役立ちます。

クライアントも、同様のサーバー認証を行う必要があります。クライアントは、それ自体が使用した登録 IP アドレスおよびポート番号と同じソース IP アドレスおよびポート番号を持つサーバーからのイベント配信を受け入れるだけです。

CRNP サービスのクライアントはクラスタを保護するファイアウォール内に配置されるのが一般的なため、CRNP にセキュリティーメカニズムは提供されていません。

## CRNP を使用する Java アプリケーションの作成例

以下の例は、CRNP を使用する `CrnpClient` というシンプルな Java アプリケーションを作成する方法を示しています。このアプリケーションでは、クラスタ上の CRNP サーバーへのイベントコールバックの登録、イベントコールバックの待機、イベントの処理 (内容の出力) を行い、終了前にイベントコールバック要求の登録解除を行います。終了前にイベントコールバック要求の登録解除を行います。

この例を参照する場合は、以下の点に注意してください。

- このアプリケーション例では、JAXP (Java API for XML Processing) を使用して XML の生成と解析を行なっています。この例は JAXP の使用方法を説明したものではありません。JAXP の詳細は、<http://java.sun.com/xml/jaxp/index.html> で説明しています。
- この例は、付録 G に示されている完全なアプリケーションコードを断片的に示したものです。この章の例は個々の概念を効果的に示すことをねらっており、付録 G に示されている完全なアプリケーションコードと多少異なります。
- また、簡潔に示すため、この章の例ではコード例からコメントを除いてあります。付録 G に示されている完全なアプリケーションコードにはコメントが含まれています。
- この例に示しているアプリケーションは終了するだけでほとんどのエラー状況に対応できるものですが、ユーザーが実際に使用するアプリケーションではエラーを徹底的に処理する必要があります。

### ▼ 環境を設定する

- 1 JAXP と、正しいバージョンの Java コンパイラおよび **Virtual Machine** をダウンロードし、インストールを行います。

作業手順は、<http://java.sun.com/xml/jaxp/index.html> に示されています。

---

注 - この例は、バージョン 1.3.1 以降の Java を必要とします。

---

- 2 ソースファイルが置かれているディレクトリから、次のように入力します。

```
% javac -classpath jaxp-root/dom.jar:jaxp-root/jaxp-api. \
jar:jaxp-rootsax.jar:jaxp-root/xalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xsltc.jar -sourcepath . source-filename.java
```

上記コマンドの `jaxp-root` には、JAXP jar ファイルが置かれているディレクトリの絶対パスまたは相対パスを指定してください。`source-filename` には、Java ソースファイルの名前を指定してください。

コンパイルコマンド行の `classpath` は、コンパイラで JAXP クラスを見つけるための指定です。

- 3 アプリケーションの実行時に、アプリケーションが正しい JAXP クラスファイルを読み込むことができるように classpath を指定します (classpath の最初のパスは現在のディレクトリ)。

```
% java -cp .:jaxp-root/dom.jar:jaxp-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xslt.jar source-filename arguments
```

以上で環境の構成が終了し、アプリケーションの開発を行える状況となります。

## ▼ アプリケーション開発を開始する

アプリケーション例のこの段階では、コマンド行引数を解析して CrnpClient オブジェクトの構築を行うメインメソッドを使用し、CrnpClient という基本的なクラスを作成します。このオブジェクトは、コマンド行引数をこのクラスに渡し、ユーザーがアプリケーションを終了するのを待って CrnpClient で shutdown を呼び出し、その後終了します。

CrnpClient クラスのコンストラクタは、次に示す作業を実行する必要があります。

- オブジェクトを処理する XML を設定する
  - イベントコールバックを待機するスレッドを作成する
  - CRNP サーバーと通信し、イベントコールバックを受け取る登録をする
- 上記のロジックを実装する Java コードを作成します。

次の例は、CrnpClient クラスのスケルトンコードを示しています。コンストラクタ内で参照される 4 つのヘルパーメソッドと停止メソッドの実装は、この章で後述します。ここでは、ユーザーが必要とするパッケージをすべてインポートするコードを示しています。

```
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;
    }
```

メンバー変数についての詳細はこの章で後述します。

## ▼ コマンド行引数を解析する

- コマンド行引数の解析方法については、[付録G](#)内のコードを参照してください。

## ▼ イベント受信スレッドを定義する

イベント受信はコード内で個別のスレッドで行われるようにする必要があります。これは、イベントスレッドがイベントコールバックを待機している間アプリケーションが継続してほかの作業を行えるようにするためです。

---

注 - XML の設定についてはこの章で後述します。

---

- 1 コード内で、`ServerSocket` を作成してソケットにイベントが到着するのを待機する `EventReceptionThread` という `Thread` サブクラスを定義します。サンプルコードのこの部分では、イベントの読み取りもイベントの処理も行われません。イベントの読み取りと処理についてはこの章で後述します。`EventReceptionThread` は、ワイルドカード IP アドレス上に `ServerSocket` を作成します。`EventReceptionThread` は、`CrnpClient` オブジェクトにイベントを送信して処理できるように、`CrnpClient` オブジェクトに対する参照も維持します。

```
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();
            // ソケットストリームからイベントを作成し、処理する。
            sock.close();
        }
        // 到達不能

    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}

/* プライベートメンバー変数 */
private ServerSocket listeningSock;
private CrnpClient client;
}

```

## 2 createEvtRecepThr オブジェクトを構築します。

```

private void createEvtRecepThr() throws Exception
{
    evtThr = new EventReceptionThread(this);
    evtThr.start();
}

```

## ▼ コールバックの登録と登録解除を行う

登録は以下の作業によって行います。

- 登録用の IP アドレスとポートに対して基本的な TCP ソケットを開く
- XML 登録メッセージを作成する
- ソケット上で XML 登録メッセージを送信する
- ソケットから XML 応答メッセージを読み取る
- ソケットを閉じる

### 1 上記のロジックを実装する Java コードを作成します。

以下のコード例は、CrnpClient クラスの registerCallbacks メソッド (CrnpClient コンストラクタによって呼び出される) の実装を示しています。

createRegistrationString() と readRegistrationReply() の呼び出しの詳細は、この章で後述します。

regIp と regPort は、コンストラクタによって設定されるオブジェクトメンバーです。

```

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 unregister メソッドを実装します。

このメソッドは、CrnpClient の shutdown メソッドによって呼び出されます。createUnregistrationString の実装の詳細はこの章で後述します。

```

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();
}

```

## ▼ XML を生成する

以上で、アプリケーション構造の設定と、通信用のコードの作成が終了しました。次は、XML の生成と解析を行うコードを作成する必要があります。初めに、SC\_CALLBACK\_REG XML 登録メッセージを生成するコードを作成します。

SC\_CALLBACK\_REG メッセージは、登録のタイプ (ADD\_CLIENT、REMOVE\_CLIENT、ADD\_EVENTS または REMOVE\_EVENTS)、コールバックポート、および要求するイベントの一覧から構成されます。各イベントはクラスとサブクラスから構成され、名前と値のペアリストが続きます。

この例のこの段階では、登録タイプ、コールバックポート、および登録イベントの一覧を格納する CallbackReg クラスを作成します。このクラスは、それ自体を SC\_CALLBACK\_REG XML メッセージにシリアル化することもできます。

このクラスには、クラスメンバーから SC\_CALLBACK\_REG XML メッセージ文字列を作成する convertToXml という興味深いメソッドがあります。このメソッドを使用したコードの詳細は、<http://java.sun.com/xml/jaxp/index.html> の JAXP ドキュメントに記載されています。

次のコード例は、Event クラスの実装を示しています。CallbackReg クラスは、イベントを 1 つ保存してそのイベントを XML Element に変換できる Event クラスを使用します。

**1** 上記のロジックを実装する **Java** コードを作成します。

```
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) {
        // 指定されたオプションを持つパーサーを構築できない。
        pce.printStackTrace();
        System.exit(1);
    }

    // root 要素を作成する。
    Element root = (Element) document.createElement("SC_CALLBACK_REG");

    // 属性を追加する。
    root.setAttribute("VERSION", "1.0");
    root.setAttribute("PORT", port);
    root.setAttribute("regType", regType);

    // イベントを追加する。
    for (int i = 0; i < regEvents.size(); i++) {
        Event tempEvent = (Event)
            (regEvents.elementAt(i));
        root.appendChild(tempEvent.createXmlElement(document));
    }
    document.appendChild(root);

    // 全体を文字列に変換する。
    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 Event クラスと NVPair クラスを実装します。  
CallbackReg クラスは、NVPair クラスを使用する Event クラスを使用します。

```
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;
}
```

## ▼ 登録メッセージと登録解除メッセージを作成する

XML メッセージを生成するヘルパークラスの作成が終了したところで、次は `createRegistrationString` メソッドの実装を記述します。このメソッドは、`registerCallbacks` メソッドによって呼び出されます (詳細は、[237 ページの「コールバックの登録と登録解除を行う」](#))。

`createRegistrationString` は、`CallbackReg` オブジェクトを構築し、その登録タイプとポートを設定します。続いて、`createRegistrationString` は、`createAllEvent`、`createMembershipEvent`、`createRgEvent`、および `createREvent` ヘルパーメソッドを使用して各種のイベントを構築します。各イベントは、`CallbackReg` オブジェクトが作成されたあとでこのオブジェクトに追加されます。最後に、

`createRegistrationString` は `CallbackReg` オブジェクト上で `convertToXml` メソッドを呼び出し、`String` 形式の XML メッセージを取得します。

`regs` メンバー変数は、ユーザーがアプリケーションに指定するコマンド行引数を格納します。5つ目以降の引数は、アプリケーションが登録を行うイベントを指定します。4つ目の引数は登録のタイプを指定しますが、この例では無視されています。付録 G に挙げられている完全なコードでは、この4つ目の引数の使用方法も示されています。

## 1 上記のロジックを実装する Java コードを作成します。

```
private String createRegistrationString() throws Exception
{
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    cbReg.setRegType(CallbackReg.ADD_CLIENT);

    // イベントを追加する
    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 登録解除文字列を作成します。

イベントを指定する必要がない分、登録解除文字列の作成は登録文字列の作成よりも簡単です。

```
private String createUnregistrationString() throws Exception
{
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
    String xmlStr = cbReg.convertToXml();
    return (xmlStr);
}
```

## ▼ XML パーサーの設定方法

以上で、アプリケーションの通信用コードと XML 生成コードの生成が終わります。CrnpClient コンストラクタは、setupXmlProcessing メソッドを呼び出します。この

メソッドは、`DocumentBuilderFactory` オブジェクトを作成し、そのオブジェクトに各種の解析プロパティを設定します。このメソッドの詳細は、JAXP ドキュメントに記載されています。<http://java.sun.com/xml/jaxp/index.html> を参照してください。

- 上記のロジックを実装する **Java** コードを作成します。

```
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // 検証を行う必要はない。
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // コメントと空白文字は無視したい。
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // CDATA セクションを TEXT ノードに結合する。
    dbf.setCoalescing(true);
}
```

## ▼ 登録応答を解析する

登録メッセージまたは登録解除メッセージに回答して CRNP サーバーが送信する `SC_REPLY XML XML` メッセージを解析するには、`RegReply` ヘルパークラスが必要です。このクラスは、XML ドキュメントから構築できます。このクラスは、ステータスコードとステータスメッセージのアクセスを提供します。サーバーからの XML ストリームを解析するには、新しい XML ドキュメントを作成してそのドキュメントの解析メソッドを使用する必要がありますこのメソッドの詳細は、<http://java.sun.com/xml/jaxp/index.html> の JAXP ドキュメントに記載されています。

- 1 上記のロジックを実装する **Java** コードを作成します。

`readRegistrationReply` メソッドは、新しい `RegReply` クラスを使用します。

```
private void readRegistrationReply(InputStream stream) throws Exception
{
    // ドキュメントビルダーを作成する。
    DocumentBuilder db = dbf.newDocumentBuilder();
    db.setErrorHandler(new DefaultHandler());

    // 入力ファイルを解析する。
    Document doc = db.parse(stream);

    RegReply reply = new RegReply(doc);
}
```

```
        reply.print(System.out);
    }
}
```

## 2 RegReply クラスを実装します。

retrieveValues メソッドは XML ドキュメント内の DOM ツリーを回り、ステータスコードとステータスメッセージを抽出します。詳細は、<http://java.sun.com/xml/jaxp/index.html> の JAXP ドキュメントに記載されています。

```
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;

        // SC_REPLY 要素を見つける
        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);

        // statusCode 属性の値を取得する。
    }
}
```

```

        statusCode = ((Element)n).getAttribute("STATUS_CODE");

        // SC_STATUS_MSG 要素を検出する
        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;
        }
        // TEXT セクションを取得する (存在する場合)。
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // 1 つも存在しなくてもエラーではないため、そのまま戻る。
            return;
        }

        // 値を取得する。
        statusMsg = n.getNodeValue();
    }

    private String statusCode;
    private String statusMsg;
}

```

## ▼ コールバックイベントを解析する

最後のステップは、実際のコールバックイベントの解析と処理です。この作業をスムーズに行うため、[238 ページの「XML を生成する」](#)で作成した `Event` クラスを変更します。このクラスを使用して、XML ドキュメントから `Event` を構築し、XML `Element` を作成できます。この変更は、XML ドキュメントを受け付ける別のコンストラクタ、`retrieveValues` メソッド、2 つの新たなメンバー変数 (`vendor` と `publisher`)、全フィールドのアクセッサメソッド、および出力メソッドを必要とします。

### 1 上記のロジックを実装する Java コードを作成します。

このコードは、[245 ページの「登録応答を解析する」](#)で説明している `RegReply` クラスのコードに似ていることに注目してください。

```

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;

        // SC_EVENT 要素を検出する
        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);

        //
        // CLASS、SUBCLASS、VENDOR、およびPUBLISHER
        // 属性の値を取得する。
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // すべての nv ペアを取得する。
        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 XML 解析をサポートする、NVPair クラスのコンストラクタとメソッドを別途実装します。

手順 1 で Event クラスに変更を加えたため、NVPair クラスにも類似した変更を加える必要があります。

```

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;

    // NAME 要素を検出する。
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }
}

```

```
// TEXT セクションを取得する。
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;
}

// 値を取得する。
name = n.getNodeValue();

// ここで要素を取得する。
nl = elem.getElementsByTagName("VALUE");
if (nl.getLength() != 1) {
    System.out.println("Error in parsing: can't find "
        + "VALUE node.");
    return;
}
// TEXT セクションを取得する
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;
}

// 値を取得する
value = n.getNodeValue();
}

public String getName()
{
    return (name);
}

public String getValue()
{
    return (value);
}
}
```

- 3 EventReceptionThread でイベントコールバックを待機する while ループを実装します。

EventReceptionThread については、[236 ページの「イベント受信スレッドを定義する」](#)を参照してください。

```
while(true) {
    Socket sock = listeningSock.accept();
    Document doc = db.parse(sock.getInputStream());
```

```
        Event event = new Event(doc);
        client.processEvent(event);
        sock.close();
    }
```

## ▼ アプリケーションを実行する

- 1 スーパーユーザーになるか、`solaris.cluster.modifyRBAC` の承認を提供する役割になります。
- 2 アプリケーションを実行します。

```
# java CrnpClient crnpHost crnpPort localPort ...
```

CrnpClient アプリケーションの完全なコードは、[付録 G](#) に示されています。





## 標準プロパティ

---

この付録では、標準リソースタイプ、リソース、リソースグループプロパティについて説明します。また、システム定義プロパティの変更および拡張プロパティの作成に使用するリソースプロパティ属性についても説明します。

---

注-リソースタイプ、リソース、リソースグループのプロパティ名は、大文字と小文字が区別されません。プロパティ名を指定する際には、大文字と小文字を任意に組み合わせることができます。

---

この付録の内容は次のとおりです。

- 253 ページの「資源タイプのプロパティ」
- 264 ページの「リソースのプロパティ」
- 285 ページの「リソースグループのプロパティ」
- 301 ページの「リソースプロパティの属性」

### 資源タイプのプロパティ

以下に、Sun Cluster ソフトウェアによって定義されるリソースタイププロパティを示します。

プロパティ値は以下のように分類されます。

- 必須。プロパティはリソースタイプ登録 (RTR) ファイルに明示的な値を必要とします。そうでない場合、プロパティが属するオブジェクトは作成できません。空白文字または空の文字列を値として指定することはできません。
- 条件付。RTR ファイル内に宣言を必要とするプロパティです。宣言がない場合、RGM はこのプロパティを作成しません。したがって、このプロパティを管理ユーティリティから利用することはできません。空白文字または空の文字列を値として指定できます。プロパティが RTR ファイル内で宣言されており、値が指定されていない場合には、RGM はデフォルト値を使用します。
- 条件付/明示。RTR ファイル内に宣言と明示的な値を必要とするプロパティです。宣言がない場合、RGM はこのプロパティを作成しません。したがって、このプロパティを管理ユーティリティから利用することはできません。空白文字または空の文字列を値として指定することはできません。
- 任意。RTR ファイル内に宣言できるプロパティです。プロパティが RTR ファイル内で宣言されていない場合は、RGM がこれを作成し、デフォルト値を与えます。プロパティが RTR ファイル内で宣言されており、値が指定されていない場合は、RGM は、プロパティが RTR ファイル内で宣言されないときのデフォルト値と同じ値を使用します。
- 照会のみ-管理ツールから直接設定できません。

Installed\_nodes と RT\_system 以外のリソースタイププロパティは、管理ユーティリティで更新を行うことはできません。また、Installed\_nodes は RTR ファイル内に宣言できないため、クラスタ管理者のみが設定できます。RT\_system には RTR ファイル内で初期値を割り当てることができ、またクラスタ管理者が設定することもできます。

以下にプロパティ名とその説明を示します。

---

注-リソースタイププロパティ名 (API\_version や Boot など) は、大文字と小文字の区別がありません。プロパティ名を指定する際には、大文字と小文字を任意に組み合わせることができます。

---

#### API\_version (integer)

このリソースタイプの実装のサポートに必要なリソース管理 API の最小バージョン。

次に、Sun Cluster の各リリースがサポートする API\_version の最大値を要約します。

3.1 以前	2
3.1 10/03	3
3.1 4/04	4

3.1 9/04	5
3.1 8/05	6
3.2	7
3.2 2/08	8

RTR ファイルにおいて `API_version` に 2 より大きな値を宣言した場合、そのリソースタイプは、宣言した値より小さな最大バージョンしかサポートしないバージョンの Sun Cluster にはインストールされません。たとえば、あるリソースタイプに `API_version=7` を宣言すると、このリソースタイプは、3.2 より前にリリースされた Sun Cluster のバージョンにはインストールされません。

---

注- このプロパティを宣言しないか、このプロパティをデフォルト値 (2) に設定すると、データサービスは Sun Cluster 3.0 以降の Sun Cluster の任意のバージョンにインストールできます。

---

カテゴリ: 任意  
 デフォルト: 2  
 調整: NONE

#### Boot (string)

任意のコールバックメソッド。RGM がノードまたはゾーン上で実行するプログラムのパスを指定します。このプログラムは、このリソースタイプが管理対象になっているとき、クラスタの結合または再結合を行います。このメソッドは、Init メソッドと同様に、このタイプのリソースを初期化します。

カテゴリ: 条件付/明示  
 デフォルト: デフォルトなし  
 調整: NONE

#### Failover (boolean)

このプロパティを TRUE に設定した場合、このタイプのリソースは、複数のノードまたはゾーンで同時にオンラインになる可能性があるどのグループ内でも構成できません。

このリソースタイプのプロパティは、次のように Scalable リソースのプロパティと一緒に使用します。

Failover リソースタイプの値	Scalable リソースの値	説明
TRUE	TRUE	この非論理的な組み合わせは指定しないでください。
TRUE	FALSE	この組み合わせは、フェイルオーバーサービスに対して指定します。
FALSE	TRUE	この組み合わせは、ネットワーク負荷分散に SharedAddress リソースを使用するスケラブルサービスに指定します。  SharedAddress については、『Sun Cluster の概念 (Solaris OS 版)』を参照してください。  スケラブルリソースグループは、非大域ゾーンで動作するように構成できます。ただし、同一ノードの複数の非大域ゾーンで動作するようには構成しないでください。
FALSE	FALSE	この組み合わせは、ネットワーク負荷分散を使用しないマルチマスターサービスを選択する場合に使用します。  このタイプのスケラブルサービスはゾーン内で使用可能です。

Scalable については、r\_properties (5)のマニュアルページと、『Sun Cluster の概念 (Solaris OS 版)』の第3章「重要な概念 - システム管理者とアプリケーション開発者」を参照してください。

カテゴリ: 任意

デフォルト: FALSE

調整: NONE

#### Fini (string)

任意のコールバックメソッド。この型のリソースを RGM 管理の対象外にするときに、RGM によって実行されるプログラムのパスです。

Fini メソッドは、通常、Init メソッドにより実行された初期化を元に戻します。

次のような状態が生じた場合にリソースが管理対象外となる各ノードまたはゾーンで、RGMはFiniを実行します。

- リソースを含むリソースグループが管理対象外状態に切り替わる。この場合、ノードリストのすべてのノードおよびゾーン上で、RGMはFiniメソッドを実行します。
- 管理されているリソースグループからリソースが削除される。この場合、ノードリストのすべてのノードおよびゾーン上で、RGMはFiniメソッドを実行します。
- リソースを含むリソースグループのノードリストからノードまたはゾーンが削除される。この場合、削除されたノードまたはゾーン上でのみ、RGMはFiniメソッドを実行します。

「ノードリスト」はリソースグループのNodelistまたはリソースタイプのInstalled\_nodes リストのいずれかです。「ノードリスト」がリソースグループのNodelistとリソースタイプのInstalled\_nodes リストのどちらを指すかは、リソースタイプのInit\_nodes プロパティの設定に依存します。Init\_nodes プロパティはRG\_nodelistまたはRT\_installed\_nodesに設定できます。大部分のリソースタイプでは、Init\_nodesはデフォルトであるRG\_nodelistに設定されます。この場合、InitメソッドとFiniメソッドは両方とも、リソースグループのNodelistで指定されているノードおよびゾーン上で実行されます。

Initメソッドが実行する初期化の種類は、次のように、ユーザーが実装したFiniメソッドが実行する必要があるクリーンアップの種類を定義します。

- ノード固有の構成のクリーンアップ。
- クラスタ全体の構成のクリーンアップ。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Global\_zone (boolean)

RTRファイルで宣言されている場合、このリソースタイプのメソッドが大域ゾーンで実行されるかどうかを示すブール値。このプロパティにTRUEが設定されている場合、リソースを含むリソースグループが非大域ゾーンで動作しているときでも、メソッドは大域ゾーンで実行されます。このプロパティにTRUEを設定するのは、ネットワークアドレスやファイルシステムなど、大域ゾーンから管理できるサービスに対してだけです。



注意 - 信頼できる既知のソースであるリソースタイプを除いて、`Global_zone` プロパティに `TRUE` が設定されているリソースタイプは登録しないでください。このプロパティに `TRUE` を設定したリソースタイプは、ゾーン分離をすり抜け、危険があります。

カテゴリ: 任意  
 デフォルト: `FALSE`  
 調整: `ANYTIME`

#### `Init(string)`

任意のコールバックメソッド。この型のリソースを RGM 管理対象にするときに、RGM によって実行されるプログラムのパスです。

カテゴリ: 条件付/明示  
 デフォルト: デフォルトなし  
 調整: `NONE`

#### `Init_nodes(enum)`

RGM が `Init`、`Fini`、`Boot`、`Validate` メソッドをコールするノードまたはゾーンを示します。指定できる値は、リソースをマスターできるノードまたはゾーンのみを指定する `RG_PRIMARYES`、またはこのリソース型がインストールされるすべてのノードまたはゾーンを指定する `RT_INSTALLED_NODES` のいずれかです。

カテゴリ: 任意  
 デフォルト: `RG_PRIMARYES`  
 調整: `NONE`

#### `Installed_nodes(string_array)`

リソースタイプを実行できるクラスタノードまたはゾーンの名前のリスト。このプロパティは RGM によって自動的に作成されます。クラスタ管理者は値を設定できます。RTR ファイル内には宣言できません。

カテゴリ: このプロパティはクラスタ管理者が構成できます。  
 デフォルト: すべてのクラスタノードおよびゾーン  
 調整: `ANYTIME`

#### `Is_logical_hostname(boolean)`

`TRUE` は、このリソースタイプが、フェイルオーバーインターネットプロトコル (IP) アドレスを管理する `LogicalHostname` リソースタイプのいずれかのバージョンであることを示します。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

#### Is\_shared\_address (boolean)

TRUE は、このリソースタイプが、共有インターネットプロトコル (IP) アドレスを管理する共有アドレスリソースタイプのいずれかのバージョンであることを示します。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

#### Monitor\_check (string)

任意のコールバックメソッド。障害モニターの要求によってこのリソースタイプのフェイルオーバーを実行する前に、RGM によって実行されるプログラムのパスです。ノードまたはゾーン上でモニターチェックプログラムがゼロ以外の値とともに終了した場合、GIVEOVER タグ付きで `scha_control` を呼び出した結果としてのそのノードまたはゾーンへのフェイルオーバーの試みは阻止されます。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Monitor\_start (string)

任意のコールバックメソッド。この型のリソースの障害モニターを起動するために RGM によって実行されるプログラムのパスです。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Monitor\_stop (string)

`Monitor_start` が設定されている場合、必須のコールバックメソッドになります。この型のリソースの障害モニターを停止するために RGM によって実行されるプログラムのパスです。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Pkglist (string\_array)

リソースタイプのインストールに含まれている任意のパッケージリスト。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

**Postnet\_stop(string)**

任意のコールバックメソッド。この型のリソースがネットワークアドレスリソースに依存している場合、このネットワークアドレスリソースのStopメソッドの呼び出し後にRGMによって実行されるプログラムのパスです。ネットワークインタフェースが停止するように構成されたあと、このメソッドはStopアクションを実行する必要があります。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

**Prenet\_start(string)**

任意のコールバックメソッド。この型のリソースがネットワークアドレスリソースに依存している場合、このネットワークアドレスリソースのStartメソッドの呼び出し前にRGMによって実行されるプログラムのパスです。このメソッドは、ネットワークインタフェースが構成される前に必要なStartアクションを行います。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

**Proxy(boolean)**

このタイプのリソースがプロキシリソースかどうかを示すブール値です。

「プロキシリソース」は、リソースの状態を Oracle Cluster Ready Services (CRS) などの別のクラスタフレームワークからインポートする Sun Cluster リソースです。Oracle クラスタウェア CRS として現在知られている Oracle CRS は、クラスタ環境向けのプラットフォームに依存しないシステムサービスセットです。

プロキシリソースタイプは、Prenet\_startメソッドを使用して、外部のプロキシリソースの状態を監視するデーモンを起動します。Postnet\_stopメソッドは、この監視デーモンを停止します。この監視デーモンは、CHANGE\_STATE\_ONLINE または CHANGE\_STATE\_OFFLINE タグとともに scha\_control コマンドを実行し、プロキシリソースの状態をそれぞれOnline またはOffline に設定します。scha\_control() 関数も、同じように SCHA\_CHANGE\_STATE\_ONLINE および SCHA\_CHANGE\_STATE\_OFFLINE タグを使用します。詳細は、scha\_control(1HA) と scha\_control(3HA) のマニュアルページを参照してください。

TRUE に設定されている場合、リソースはプロキシリソースです。

カテゴリ: 任意

デフォルト: FALSE

調整: NEVER

#### Resource\_list(string\_array)

リソースタイプの全リソースのリストです。クラスタ管理者はこのプロパティを直接設定しません。ただし、クラスタ管理者がこの型のリソースをリソースグループに追加したり、リソースグループから削除した場合、RGMはこのプロパティを更新します。

カテゴリ: 照会のみ

デフォルト: 空のリスト

調整: NONE

#### Resource\_type(string)

リソースタイプの名前。現在登録されているリソースタイプ名を表示するには、次のコマンドを使用します。

#### **resourcetype show +**

Sun Cluster 3.1 および Sun Cluster 3.2 では、リソースタイプ名にバージョンが含まれます(必須)。

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

リソースタイプ名はRTRファイル内に指定された3つのプロパティ *vendor-id*、*resource-type*、*rt-version* で構成されます。**resourcetype** コマンドは、ピリオド(.)とコロン(:)のプロパティの間に挿入します。リソースタイプの名前の最後の部分、*rt-version* には、**RT\_version** プロパティと同じ値が入ります。*vendor\_id* が一意であることを保証するためには、リソース型を作成した会社の株式の略号を使用します。Sun Cluster 3.1 以前に登録されたリソースタイプ名では、引き続き次の構文を使用します。

*vendor-id.resource-type*

カテゴリ: 必要

デフォルト: 空の文字列

調整: NONE

#### RT\_basedir(string)

コールバックメソッドの相対パスを補完するディレクトリパスです。このパスは、リソースタイプパッケージのインストールディレクトリに設定する必要があります。このパスには、スラッシュ(/)で開始する完全なパスを指定する必要があります。

カテゴリ: 必須(絶対パスでないメソッドパスがある場合)

デフォルト: デフォルトなし

調整: NONE

RT\_description(string)

リソース型の簡単な説明です。

カテゴリ: 条件付き

デフォルト: 空の文字列

調整: NONE

RT\_system(boolean)

リソースタイプの RT\_system プロパティが TRUE の場合、そのリソースタイプは削除できません (**resourcetype unregister resource-type-name**)。このプロパティは、LogicalHostname など、クラスタのインフラをサポートするリソースタイプを間違っって削除してしまうことを防ぎます。しかし、RT\_system プロパティはどのリソース型にも適用できます。

RT\_system プロパティが TRUE に設定されたリソース型を削除するには、まず、このプロパティを FALSE に設定する必要があります。クラスタサービスをサポートするリソースを持つリソース型を削除するときには注意してください。

カテゴリ: 任意

デフォルト: FALSE

調整: ANYTIME

RT\_version(string)

Sun Cluster 3.1 リリース以降では、このリソースタイプの実装を特定する必須バージョン文字列。Sun Cluster 3.0 ではこのプロパティは任意でした。RT\_version は完全なリソースタイプ名のサフィックスコンポーネントです。

カテゴリ: 条件付き/明示または必須

デフォルト: デフォルトなし

調整: NONE

Single\_instance(boolean)

TRUE は、この型のリソースがクラスタ内に 1 つだけ存在できることを示します。

カテゴリ: 任意

デフォルト: FALSE

調整: NONE

Start(string)

コールバックメソッド。この型のリソースを起動するために RGM によって実行されるプログラムのパスです。

カテゴリ: RTR ファイルで Prenet\_start メソッドが宣言されていないかぎり  
必須

デフォルト: デフォルトなし

調整: NONE

#### Stop (string)

コールバックメソッド。この型のリソースを停止するために RGM によって実行されるプログラムのパスです。

カテゴリ: RTR ファイルで Postnet\_stop メソッドが宣言されていないかぎり  
必須

デフォルト: デフォルトなし

調整: NONE

#### Update (string)

任意のコールバックメソッド。この型の実行中のリソースのプロパティが変更されたときに、RGM によって実行されるプログラムのパスです。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Validate (string)

任意のコールバックメソッド。この型のリソースのプロパティ値を検査するために RGM により実行されるプログラムのパスです。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

#### Vendor\_ID (string)

Resource\_type を参照してください。

カテゴリ: 条件付き

デフォルト: デフォルトなし

調整: NONE

## リソースのプロパティ

この節では、Sun Cluster ソフトウェアによって定義されているリソースプロパティについて説明します。

プロパティ値は以下のように分類されます。

- 必須。クラスタ管理者は、管理ユーティリティでリソースを作成するときに、必ず値を指定する必要があります。
- 任意。クラスタ管理者がリソースグループの作成時に値を指定しないと、システムのデフォルト値が使用されます。
- 条件付。RGM は、RTR ファイル内にプロパティが宣言されている場合にかぎりプロパティを作成します。宣言されていない場合プロパティは存在せず、クラスタ管理者はこれを利用できません。RTR ファイルで宣言されている条件付きのプロパティは、デフォルト値が RTR ファイル内で指定されているかどうかによって、必須または任意になります。詳細は、各条件付きプロパティの説明を参照してください。
- 照会のみ。管理ツールで直接設定することはできません。

301 ページの「リソースプロパティの属性」で説明されている Tunable 属性は、次のように、リソースプロパティを更新できるかどうか、および、いつ更新できるかを示します。

FALSE または NONE	しない
TRUE または ANYTIME	すべての時刻
AT_CREATION	リソースをクラスタに追加するとき
WHEN_DISABLED	リソースが無効なとき

以下にプロパティ名とその説明を示します。

### Affinity\_timeout(integer)

リソース内のサービスのクライアント IP アドレスからの接続は、この時間(秒数)内に同じサーバーノードまたはゾーンに送信されます。

このプロパティは、Load\_balancing\_policy が Lb\_sticky または Lb\_sticky\_wild の場合にかぎり有効です。さらに、Weak\_affinity が FALSE に設定されている必要があります。

このプロパティは、スケーラブルサービス専用です。

カテゴリ:	任意
デフォルト:	デフォルトなし
調整:	ANYTIME

各コールバックメソッドの `Boot_timeout` (integer)

RGMがメソッドの呼び出しに失敗したと判断するまでの時間(秒)。特定のリソースタイプに関して、タイムアウトのプロパティはRTRファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTRファイルにメソッド自体が宣言されている場合は3600(1時間)

調整: ANYTIME

`Cheap_probe_interval` (integer)

リソースの即時障害検証の呼び出しの間隔(秒数)。このプロパティはRGMによって作成されます。RTRファイルに宣言されている場合にかぎり、クラスタ管理者は使用を許可されます。RTRファイル内でデフォルト値が指定されている場合、このプロパティは任意です。

RTRファイル内にTunable属性が指定されていない場合、このプロパティのTunable値はWHEN\_DISABLEDになります。

カテゴリ: 条件付き

デフォルト: デフォルトなし

調整: WHEN\_DISABLED

## 拡張プロパティ

そのリソースのタイプのRTRファイルで宣言される拡張プロパティ。リソースタイプの実装によって、これらのプロパティを定義します。拡張プロパティに設定可能な各属性については、[301 ページの「リソースプロパティの属性」](#)を参照してください。

カテゴリ: 条件付き

デフォルト: デフォルトなし

調整: 特定のプロパティに依存

`Failover_mode` (enum)

リソースが正常に開始または停止できなかった場合、またはリソースモニターが正常ではないリソースを検出し、その結果再起動またはフェイルオーバーを要求する場合にRGMが取る回復アクションを変更します。

NONE、SOFT、またはHARD(メソッドの失敗)

これらの設定が影響するのは、起動または停止メソッド(`PreNet_start`、`Start`、`Monitor_stop`、`Stop`、`Postnet_stop`)が失敗した場合のフェイルオーバー動作のみです。RESTART\_ONLY設定とLOG\_ONLY設定は、リソースモニターが`scha_control`コマンドまたは`scha_control()`関数の実行を開始できるかどうかにも影響します。詳細は、`scha_control(1HA)` および `scha_control(3HA)` のマニュアルページを参照

してください。NONE は、前述の起動メソッドまたは停止メソッドが失敗する場合に RGM が何の復旧処理も行わないことを示します。SOFT または HARD は、Start または Pernet\_start メソッドが失敗した場合、RGM はリソースのグループを別のノードまたはゾーンに再配置することを示します。Start または Pernet\_start の失敗に関しては、SOFT と HARD は同じになります。

停止メソッド (Monitor\_stop、Stop、または Postnet\_stop) の失敗に関しては、SOFT は NONE と同じになります。これらの停止メソッドのいずれかが失敗したときに Failover\_mode が HARD に設定されている場合、RGM はノードまたはゾーンをリブートして、強制的にリソースグループをオフライン状態にします。これにより RGM は別のノードまたはゾーンでグループの起動を試みるのが可能になります。

#### RESTART\_ONLY または LOG\_ONLY

起動メソッドまたは停止メソッドが失敗する場合にフェイルオーバー動作に影響を与える NONE、SOFT、HARD とは異なり、RESTART\_ONLY と LOG\_ONLY はすべてのフェイルオーバー動作に影響を与えます。フェイルオーバー動作には、モニター起動 (scha\_control) によるリソースやリソースグループの再起動や、リソースモニター (scha\_control) によって開始されるギブオーバーなどがあります。

RESTART\_ONLY は、モニターが scha\_control を実行してリソースまたはリソースグループを再起動できることを意味します。RGM では、Retry\_interval の間に Retry\_count 回数だけ再起動を試行できます。Retry\_count の回数を超えると、それ以上の再起動は許可されません。

---

注-Retry\_count の負の値は、リソースタイプによっては適用できませんが、リソースを無制限に再起動できることを指定します。より確実に無制限の再起動を指定するには、次の手順を実行します。

- Retry\_interval に 1 や 0 などの小さい値を指定します。
- Retry\_count に 1000 などの大きい値を指定します。

リソースタイプが Retry\_count および Retry\_interval プロパティを宣言しない場合は、リソースは回数の制限なく再起動できます。

---

Failover\_mode が LOG\_ONLY に設定されている場合、リソースの再起動またはギブオーバーは許可されます。Failover\_mode を LOG\_ONLY に設定するのは、Failover\_mode を RESTART\_ONLY に設定し、Retry\_count をゼロに設定するのと同じことです。

#### RESTART\_ONLY または LOG\_ONLY (メソッドの失敗)

Pernet\_start、Start、Monitor\_stop、Stop、または Postnet\_stop メソッドが失敗した場合、RESTART\_ONLY と LOG\_ONLY は NONE と同じことです。つまり、ノードまたはゾーンのフェイルオーバーやリブートはどちらも行われません。

## データサービスに対する Failover\_mode 設定の影響

Failover\_mode の各設定がデータサービスに及ぼす影響は、データサービスが監視されているかどうか、およびデータサービスが Data Services Development Library (DSDL) に基づいているかどうかによって決まります。

- データサービスが Monitor\_start メソッドを実装し、リソースの監視が有効である場合、データサービスは監視されます。RGM は、リソースそれ自体を起動したあとで Monitor\_start メソッドを実行することにより、リソースモニターを起動します。リソースモニターはリソースが正常であるかどうかを検証します。検証が失敗した場合、リソースモニターは scha\_control() 関数を呼び出すことにより、再起動またはフェイルオーバーを要求できます。DSDL ベースのリソースの場合、検証によりデータサービスの部分的な障害(機能低下)または完全な障害が明らかになる場合があります。部分的な障害が繰り返し蓄積されると、完全な障害になります。
- データサービスが監視されないのは、データサービスが Monitor\_start メソッドを提供しないか、リソースの監視が無効になっている場合です。
- DSDL ベースのデータサービスには、Agent Builder や GDS により開発されたデータサービス、または DSDL を直接使用して開発されたデータサービスが含まれます。HA Oracle など一部のデータサービスは、DSDL を使用せずに開発されています。

### NONE、SOFT、または HARD (検証の失敗)

Failover\_mode が NONE、SOFT、または HARD に設定されていて、データサービスが監視対象の DSDL ベースのサービスであり、検証が完全に失敗した場合、モニターは scha\_control() 関数を呼び出してリソースの再起動を要求します。検証が失敗し続ける場合、リソースは Retry\_interval 期間内の Retry\_count の最大回数まで再起動されます。Retry\_count の再起動数に到達したあとも検証が再び失敗した場合、モニターは別のノードまたはゾーンに対してリソースのグループのフェイルオーバーを要求します。

Failover\_mode が NONE、SOFT、または HARD に設定され、データサービスが監視対象外の DSDL ベースのサービスである場合、検出される唯一の障害はリソースのプロセスツリーの故障のみです。リソースのプロセスツリーが故障すると、リソースが再起動されます。

データサービスが DSDL ベースのサービスではない場合、再起動またはフェイルオーバー動作は、リソースモニターがどのようにコード化されているかによって決まります。たとえば Oracle リソースモニターは、リソースまたはリソースグループを再起動するか、リソースグループのフェイルオーバーを行うことで回復します。

### RESTART\_ONLY (検証の失敗)

Failover\_mode が RESTART\_ONLY に設定され、データサービスが監視対象の DSDL ベースのサービスである場合、検証が完全に失敗すると、リソースは Retry\_interval の期間内に Retry\_count の回数再起動されます。しかし、Retry\_count を超えた時点で、リソースモニターは終了してリソース状態を FAULTED に設定し、状態メッセージ「Application faulted, but not restarted. Probe quitting.」を生成します。この時点で監視はまだ有効ですが、リソースがクラスタ管理者により修復および再起動されるまで、リソースは事実上監視対象外になります。

Failover\_mode が RESTART\_ONLY に設定され、データサービスが監視対象外の DSDL ベースのサービスである場合、プロセスツリーが故障すると、リソースは再起動されません。

監視対象データサービスが DSDL ベースのデータサービスではない場合、回復動作はリソースモニターがどのようにコード化されているかに依存します。Failover\_mode を RESTART\_ONLY に設定した場合には、Retry\_interval 内で Retry\_count に指定されている回数だけ scha\_control() 関数を呼び出してリソースまたはリソースグループを再起動できます。リソースモニターが Retry\_count を超えると、再起動は失敗します。モニターが scha\_control() 関数を呼び出してフェイルオーバーを要求する場合、その要求も同様に失敗します。

#### LOG\_ONLY (検証の失敗)

Failover\_mode がデータサービスに対して LOG\_ONLY に設定されている場合、すべての scha\_control() はリソースまたはリソースグループの再起動を要求するか、除外されているグループのフェイルオーバーを要求します。データサービスが DSDL ベースである場合、検証が完全に失敗した場合メッセージが記録されますが、リソースは再起動されません。プローブが Retry\_interval 内で Retry\_count の回数よりも多く完全に失敗した場合、リソースモニターは終了し、リソースのステータスを FAULTED に設定し、ステータスメッセージ「Application faulted, but not restarted. Probe quitting.」を生成します。この時点で監視はまだ有効ですが、リソースがクラスタ管理者により修復および再起動されるまで、リソースは事実上監視対象外になります。

Failover\_mode が LOG\_ONLY に設定されていて、データサービスが監視対象外の DSDL ベースのサービスであり、プロセスツリーが故障した場合、メッセージが記録されますが、リソースは再起動されません。

監視対象データサービスが DSDL ベースのデータサービスではない場合、回復動作はリソースモニターがどのようにコード化されているかに依存します。Failover\_mode を LOG\_ONLY に設定した場合は、リソースまたはリソースグループの再起動あるいはグループのフェイルオーバーを行う scha\_control() 要求はすべて失敗します。

カテゴリ: 任意

デフォルト: NONE

調整: ANYTIME

各コールバックメソッドの `Finis_timeout (integer)`

RGMがメソッドの呼び出しに失敗したと判断するまでの時間(秒)。特定のリソースタイプに関して、タイムアウトのプロパティはRTRファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTRファイルにメソッド自体が宣言されている場合は3600(1時間)

調整: ANYTIME

各コールバックメソッドの `Init_timeout (integer)`

RGMがメソッドの呼び出しに失敗したと判断するまでの時間(秒)。特定のリソースタイプに関して、タイムアウトのプロパティはRTRファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTRファイルにメソッド自体が宣言されている場合は3600(1時間)

調整: ANYTIME

`Load_balancing_policy (string)`

使用する負荷均衡ポリシーを定義する文字列。このプロパティは、スケーラブルサービス専用です。RTRファイルに `Scalable` プロパティが宣言されている場合、RGMは自動的にこのプロパティを作成します。`Load_balancing_policy`には次の値を設定できます。

`Lb_weighted`(デフォルト)。`Load_balancing_weights` プロパティで設定されているウエイトに従って、さまざまなノードに負荷が分散されます。

`Lb_sticky`。スケーラブルサービスの指定のクライアント(クライアントのIPアドレスで識別される)は、常に同じクラスタノードに送信されます。

`Lb_sticky_wild`。ワイルドスティッキーサービスのIPアドレスに接続する `Lb_sticky_wild` で指定されたクライアントのIPアドレスは、IPアドレスが到着するポート番号とは無関係に、常に同じクラスタノードに送られます。

カテゴリ: 条件付き/任意

デフォルト: `Lb_weighted`

調整: `AT_CREATION`

`Load_balancing_weights (string_array)`

このプロパティは、スケーラブルサービス専用です。RTRファイルに `Scalable` プロパティが宣言されている場合、RGMは自動的にこのプロパティを作成します。形式は、「`weight@node, weight@node`」になります。`weight`は指定のノード

(*node*) に対する負荷分散の相対的な割り当てを示す整数になります。ノードに分散される負荷の割合は、すべてのウエイトの合計でこのノードのウエイトを割った値になります。たとえば、`1@1,3@2` は、ノード 1 が負荷の 1/4 を受け取り、ノード 2 が負荷の 3/4 を受け取ることを指定します。デフォルトの空の文字列(“”)は、一定の分散を指定します。明示的にウエイトを割り当てられていないノードのウエイトは、デフォルトで 1 になります。

RTR ファイル内に `Tunable` 属性が指定されていない場合、このプロパティの `Tunable` 値は `ANYTIME` になります。このプロパティを変更すると、新しい接続時にのみ分散が変更されます。

カテゴリ: 条件付き/任意

デフォルト: 空の文字列(“”)

調整: `ANYTIME`

各コールバックメソッドの `Monitor_check_timeout` (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は `3600` (1 時間)

調整: `ANYTIME`

各コールバックメソッドの `Monitor_start_timeout` (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は `3600` (1 時間)

調整: `ANYTIME`

各コールバックメソッドの `Monitor_stop_timeout` (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は `3600` (1 時間)

調整: `ANYTIME`

**Monitored\_switch (enum)**

クラスタ管理者が管理ユーティリティを使ってモニターを有効または無効にすると、RGMによってEnabledまたはDisabledに設定されます。Disabledに設定されている場合、リソースの監視は停止されますが、リソースそれ自体はオンラインのままになります。監視が再度有効になるまで、Monitor\_startメソッドは呼び出されません。リソースが、モニターのコールバックメソッドを持っていない場合は、このプロパティは存在しません。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

**Network\_resources\_used (string\_array)**

このリソースが依存関係を持っている論理ホスト名または共有アドレスネットワークリソースのリスト。このリストには、プロパティ Resource\_dependencies、Resource\_dependencies\_weak、Resource\_dependencies\_restart、またはResource\_dependencies\_offline\_restartに現れるすべてのネットワークアドレスリソースが含まれます。

RTR ファイルに Scalable プロパティが宣言されている場合、RGMは自動的にこのプロパティを作成します。ScalableがRTRファイルで宣言されていない場合、Network\_resources\_usedはRTRファイルで明示的に宣言されていないかぎり使用できません。

このプロパティは、リソース依存関係プロパティの設定に基づいて、RGMにより自動的に更新されます。このプロパティを直接設定する必要はありません。しかし、このプロパティにリソース名を追加する場合、そのリソース名は自動的にResource\_dependenciesプロパティに追加されます。また、このプロパティからリソース名を削除する場合、そのリソース名は自動的に、そのリソースが現れるすべてのリソース依存関係プロパティから削除されます。

カテゴリ: 条件付き/任意

デフォルト: 空のリスト

調整: ANYTIME

**各クラスタノードまたはゾーン上の Num\_resource\_restarts (integer)**

過去  $n$  秒以内にこのリソースで発生した再起動要求の数。 $n$ は、Retry\_intervalプロパティの値です。

再起動要求は、次に示す呼び出しのいずれかです。

- RESOURCE\_RESTART 引数を持つ scha\_control(1HA) コマンド。
- SCHA\_RESOURCE\_RESTART 引数を持つ scha\_control(3HA) 関数。
- RESOURCE\_IS\_RESTARTED 引数を持つ scha\_control コマンド。
- SCHA\_RESOURCE\_IS\_RESTARTED 引数を持つ scha\_control 関数。

リソースが次に示す処理のいずれかを実行した場合、RGMは、ある特定のノードまたはゾーン上にある特定のリソースに対して再起動カウンタを必ず0にリセットします。

- GIVEOVER 引数を持つ `scha_control` コマンド。
- SCHA\_GIVEOVER 引数を持つ `scha_control` 関数。

カウンタは、ギブオーバーの試行が成功した場合でも失敗した場合でもリセットされます。

リソース型が `Retry_interval` プロパティを宣言していない場合、この型のリソースに `Num_resource_restarts` プロパティを使用できません。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: 説明を参照

各クラスターノードまたはゾーン上の `Num_rg_restarts` (integer)

過去  $n$  秒以内にこのリソースに対して発生したりソースグループ再起動要求の数。  $n$  は、`Retry_interval` プロパティの値です。

リソースグループ再起動要求は、次に示す呼び出しのいずれかです。

- RESTART 引数を持つ `scha_control(1HA)` コマンド。
- SCHA\_RESTART 引数を持つ `scha_control(3HA)` 関数。

リソース型が `Retry_interval` プロパティを宣言していない場合、この型のリソースに `Num_resource_restarts` プロパティを使用できません。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: 説明を参照

`On_off_switch` (enum)

クラスター管理者が管理ユーティリティを使ってリソースを有効または無効にすると、RGMによって `Enabled` または `Disabled` に設定されます。無効に設定されている場合、リソースはオフラインにされ、再度有効にされるまでコールバックは実行されません。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

`Port_list` (string\_array)

サーバーが待機するポートの番号リストです。ポート番号には、スラッシュ (/) と、そのポートで使用されるプロトコルが付加されます(たとえば、`Port_list=80/tcp` や `Port_list=80/tcp6,40/udp6` など)。

プロトコルには、次のものを指定できます。

- tcp (TCP IPv4)
- tcp6 (TCP IPv6)
- udp (UDP IPv4)
- udp6 (UDP IPv6)

Scalable プロパティが RTR ファイルで宣言されている場合、RGM は自動的に Port\_list を作成します。それ以外の場合、このプロパティは RTR ファイルで明示的に宣言されていないかぎり使用できません。

Apache 用にこのプロパティを設定する方法は、『Sun Cluster Data Service for Apache Guide for Solaris OS』を参照してください。

カテゴリ: 条件付き/必須

デフォルト: デフォルトなし

調整: ANYTIME

各コールバックメソッドの Postnet\_stop\_timeout (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は 3600 (1 時間)

調整: ANYTIME

各コールバックメソッドの Prenet\_start\_timeout (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は 3600 (1 時間)

調整: ANYTIME

R\_description (string)

リソースの簡単な説明。

カテゴリ: 任意

デフォルト: 空の文字列

調整: ANYTIME

**Resource\_dependencies (string\_array)**

リソースが強い依存関係を持っているリソースのリスト。強い依存関係は、メソッド呼び出しの順序を決定します。

リソースの依存関係を有するリソースは依存しているリソースと呼ばれ、依存されているリソースと呼ばれるリスト内のリソースがオンラインでないと依存しているリソースを起動することはできません。依存しているリソースと、リスト内のいずれかの依存されているリソースが同時に起動した場合、RGMは、リスト内の依存されているリソースが起動するまで依存しているリソースの起動を待ちます。依存されているリソースが起動しないと、依存しているリソースはオフラインのままになります。依存されているリソースが起動しない場合があるのは、リスト内の依存されているリソースのリソースグループがオフラインのままであるか、**Start\_failed** 状態であるためです。異なるリソースグループ内の依存されているリソースが起動に失敗したり、無効またはオフラインになったりしていることが原因で、依存しているリソースがオフラインのままになっている場合、依存しているリソースのグループは **Pending\_online\_blocked** 状態になります。起動に失敗した、無効である、またはオフラインである同じリソースグループ内の依存されているリソースに、依存しているリソースが依存関係を持っている場合、リソースグループは **Pending\_online\_blocked** 状態にはなりません。

同じリソースグループ内では、デフォルトとして、アプリケーションリソースがネットワークアドレスリソースに対して暗黙的に強いリソース依存性を持っています。詳細は、[285 ページの「リソースグループのプロパティ」](#)の **Implicit\_network\_dependencies** を参照してください。

同じリソースグループ内では、依存性の順序に従って **Prenet\_start** メソッドが **Start** メソッドより先に実行されます。同様に、**Postnet\_stop** メソッドは **Stop** メソッドよりあとに実行されます。異なるリソースグループ内では、依存されるリソースが **Prenet\_start** と **Start** を実行してから、依存するリソースが **Prenet\_start** を実行します。依存されているリソースは、依存しているリソースグループが **Stop** および **Postnet\_stop** を完了するまで待機してから、**Stop** を実行します。

依存関係の範囲を指定するには、このプロパティを指定するときに、次の修飾子の中括弧 {} を含めてリソース名に付加します。

- |                     |  |
|---------------------|--|
| <b>{LOCAL_NODE}</b> | 指定される依存関係をノード単位に限定します。依存関係の動作は、同じノード上でのみ依存先のリソースに影響されます。依存しているリソースは、依存先のリソースが同じノードで起動されるまで待機します。停止と再起動で同じような状況になります。       |
| <b>{ANY_NODE}</b>   | 指定された依存関係を任意のノードに拡張します。依存関係の動作は、どのノードでも依存先のリソースに影響されます。依存しているリソースは、自分が起動する前に依存先のリソースが主ノードで起動するまで待機します。停止と再起動で同じような状況になります。 |

`{FROM_RG_AFFINITIES}` リソース依存関係の範囲が、リソースが属するリソースグループの `RG_affinities` 関係から派生することを指定します。依存しているリソースのグループが依存されているリソースのグループに対して肯定的なアフィニティを持っていて、リソースグループが同一ノード上で起動または停止する場合、依存関係は `{LOCAL_NODE}` であるとみなされます。そのような肯定的なアフィニティが存在しない場合、または異なるノード上でグループが起動する場合、依存関係は `{ANY_NODE}` です。

同じリソースグループ内の2つのリソース間のリソース依存関係は、常に `{LOCAL_NODE}` です。

修飾子を指定しない場合は、`FROM_RG_AFFINITIES` がデフォルトで使用されます。

カテゴリ: 任意

デフォルト: 空のリスト

調整: ANYTIME

#### `Resource_dependencies_offline_restart (string_array)`

`Resource_dependencies_offline_restart` リソースがオフライン再起動依存関係を持つ、同じまたは異なるグループ内のリソースのリストです。

このプロパティは、`Resource_dependencies` とまったく同じように動作します。ただし、このリソースは、オフライン再起動依存関係リスト内のいずれかのリソースが停止されると、停止されます。オフライン再起動依存関係リスト内のそのリソースが続けて再起動されると、このリソースも再起動されます。

リスト内の任意のリソースの起動に失敗した場合、このリソースは起動されません。このリソースと、リストのリソースの1つが同時に起動されると、RGMは、リストのリソースが始動してからこのリソースを起動します。このリソースの `Resource_dependencies` リスト内のリソースが始動しない場合(たとえば、リスト内のリソースのリソースグループがオフラインのままであったり、リスト内のリソースが `Start_failed` 状態にある場合)には、このリソースもオフラインのままです。このリソースが依存する、別のリソースグループのリソースが始動しないために、このリソースがオフラインのままの状態である場合、このリソースのグループは `Pending_online_blocked` 状態に入ります。

このリソースが、リストのリソースと同時にオフラインにされる場合は、このリソースが停止されてから、リストのほかのリソースが停止されます。しかし、このリソースがオンラインのままであったり、停止に失敗した場合でも、リストのリソースは停止されます。

ノードまたはゾーン上にある「依存先の」リソースで障害が発生した場合、RGMは該当ノードまたはゾーン上の該当リソースをオフラインにします。また、RGMは、すべての依存先のリソースのオフライン再起動依存リソースで再起動をトリ

ガーすることによって、これらをオフライン化します。クラスタ管理者が障害を解決し、依存先のリソースを再度有効にすると、RGM は、リソースのオフライン再起動依存リソースも再度オンラインにします。

依存関係の範囲を指定するには、このプロパティを指定するときに、次の修飾子の中括弧 {} を含めてリソース名に付加します。

{LOCAL_NODE}	指定される依存関係をノード単位に限定します。依存関係の動作は、同じノード上でのみ依存先のリソースに影響されます。依存しているリソースは、依存先のリソースが同じノードで起動されるまで待機します。停止と再起動で同じような状況になります。
{ANY_NODE}	指定された依存関係を任意のノードに拡張します。依存関係の動作は、どのノードでも依存先のリソースに影響されます。依存しているリソースは、自分が起動する前に依存先のリソースが主ノードで起動するまで待機します。停止と再起動で同じような状況になります。
{FROM_RG_AFFINITIES}	リソース依存関係の範囲が、リソースが属するリソースグループの <code>RG_affinities</code> 関係から派生することを指定します。依存しているリソースのグループが依存されているリソースのグループに対して肯定的なアフィニティを持っていて、リソースグループが同一ノード上で起動または停止する場合、依存関係は {LOCAL_NODE} であるとみなされます。そのような肯定的なアフィニティが存在しない場合、または異なるノード上でグループが起動する場合、依存関係は {ANY_NODE} です。

同じリソースグループ内の2つのリソース間のリソース依存関係は、常に {LOCAL\_NODE} です。

修飾子を指定しない場合は、FROM\_RG\_AFFINITIES がデフォルトで使用されます。

カテゴリ: 任意  
 デフォルト: 空のリスト  
 調整: ANYTIME

#### Resource\_dependencies\_restart(string\_array)

リソースが再起動の依存関係を持っているリソースのリスト。再起動の依存関係は、メソッド呼び出しの順序を決定します。

このプロパティの動作は Resource\_dependencies とよく似ていますが、1点例外があります。依存されているリソースと呼ばれる、再起動の依存関係リストのリソースが再起動すると、依存しているリソースと呼ばれるリソースの依存関係を有するリソースが再起動します。リスト内の依存されているリソースがオンライ

ンに戻ったあと、RGMは依存しているリソースを停止し、再起動します。このような再起動動作が発生するのは、依存しているリソースと依存されているリソースを含むリソースグループがオンラインのままである場合です。

リソースの依存関係を有するリソースは依存しているリソースと呼ばれ、依存されているリソースと呼ばれるリスト内のリソースがオンラインでないと依存しているリソースを起動することはできません。依存しているリソースと、リスト内のいずれかの依存されているリソースが同時に起動した場合、RGMは、リスト内の依存されているリソースが起動するまで依存しているリソースの起動を待ちます。依存されているリソースが起動しないと、依存しているリソースはオフラインのままになります。依存されているリソースが起動しない場合があるのは、リスト内の依存されているリソースのリソースグループがオフラインのままであるか、`Start_failed`状態であるためです。異なるリソースグループ内の依存されているリソースが起動に失敗したり、無効またはオフラインになったりしていることが原因で、依存しているリソースがオフラインのままになっている場合、依存しているリソースのグループは`Pending_online_blocked`状態になります。起動に失敗した、無効である、またはオフラインである同じリソースグループ内の依存されているリソースに、依存しているリソースが依存関係を持っている場合、リソースグループは`Pending_online_blocked`状態にはなりません。

依存関係の範囲を指定するには、このプロパティを指定するときに、次の修飾子を中括弧 `{}` を含めてリソース名に付加します。

- |                                   |   |
|-----------------------------------|---|
| <code>{LOCAL_NODE}</code>         | 指定される依存関係をノード単位に限定します。依存関係の動作は、同じノード上でのみ依存先のリソースに影響されます。依存しているリソースは、依存先のリソースが同じノードで起動されるまで待機します。停止と再起動で同じような状況になります。  |
| <code>{ANY_NODE}</code>           | 指定された依存関係を任意のノードに拡張します。依存関係の動作は、どのノードでも依存先のリソースに影響されます。依存しているリソースは、自分が起動する前に依存先のリソースが主ノードで起動するまで待機します。停止と再起動で同じような状況になります。  |
| <code>{FROM_RG_AFFINITIES}</code> | リソース依存関係の範囲が、リソースが属するリソースグループの <code>RG_affinities</code> 関係から派生することを指定します。依存しているリソースのグループが依存されているリソースのグループに対して肯定的なアフィニティを持っていて、リソースグループが同一ノード上で起動または停止する場合、依存関係は <code>{LOCAL_NODE}</code> であるとみなされます。そのような肯定的なアフィニティが存在しない場合、または異なるノード上でグループが起動する場合、依存関係は <code>{ANY_NODE}</code> です。 |

同じリソースグループ内の2つのリソース間のリソース依存関係は、常に {LOCAL\_NODE} です。

修飾子を指定しない場合は、FROM\_RG\_AFFINITIES がデフォルトで使用されます。

カテゴリ: 任意

デフォルト: 空のリスト

調整: ANYTIME

#### Resource\_dependencies\_weak(string\_array)

リソースが弱い依存関係を持っているリソースのリスト。弱い依存関係は、メソッド呼び出しの順序を決定します。

依存しているリソースと呼ばれる、リソースの依存関係を有するリソースの Start メソッドの前に、RGM は、依存されているリソースと呼ばれるこのリスト内のリソースの Start メソッドを呼び出します。RGM は、依存されているリソースの Stop メソッドの前に、依存しているリソースの Stop メソッドを呼び出します。依存するリソースは、依存されるリソースが起動に失敗した場合やオフラインのままになっている場合でも、起動することができます。

Resource\_dependencies\_weak リストの依存しているリソースと依存されているリソースが同時に起動した場合、RGM は、リスト内の依存されているリソースが起動するまで、依存しているリソースの起動を待機します。リスト内の依存されているリソースが起動しない場合でも (たとえば、リスト内の依存されているリソースのリソースグループがオフラインのままであったり、リスト内の依存されているリソースが Start\_failed 状態である場合)、依存しているリソースは起動します。依存しているリソースの Resource\_dependencies\_weak リストのリソースが起動する際に、依存しているリソースのリソースグループが一時的に Pending\_online\_blocked 状態に入ることがあります。リストのすべての依存されているリソースが起動した時点、または起動に失敗した時点で、依存しているリソースは起動し、そのグループは再度 Pending\_online 状態になります。

同じリソースグループ内では、依存性の順序に従って Prenet\_start メソッドが Start メソッドより先に実行されます。同様に、Postnet\_stop メソッドは Stop メソッドよりあとに実行されます。異なるリソースグループ内では、依存されるリソースが Prenet\_start と Start を実行してから、依存するリソースが Prenet\_start を実行します。依存されているリソースは、依存しているリソースグループが Stop および Postnet\_stop を完了するまで待機してから、Stop を実行します。

依存関係の範囲を指定するには、このプロパティを指定するときに、次の修飾子を中括弧 {} を含めてリソース名に付加します。

{LOCAL\_NODE} 指定される依存関係をノード単位に限定します。依存関係の動作は、同じノード上でのみ依存先のリソースに影響されます。依存しているリソースは、依存先のリソース

スが同じノードで起動されるまで待機します。停止と再起動で同じような状況になります。

{ANY\_NODE}

指定された依存関係を任意のノードに拡張します。依存関係の動作は、どのノードでも依存先のリソースに影響されます。依存しているリソースは、自分が起動する前に依存先のリソースが主ノードで起動するまで待機します。停止と再起動で同じような状況になります。

{FROM\_RG\_AFFINITIES}

リソース依存関係の範囲が、リソースが属するリソースグループの `RG_affinities` 関係から派生することを指定します。依存しているリソースのグループが依存されているリソースのグループに対して肯定的なアフィニティを持っていて、リソースグループが同一ノード上で起動または停止する場合、依存関係は {LOCAL\_NODE} であるとみなされます。そのような肯定的なアフィニティが存在しない場合、または異なるノード上でグループが起動する場合、依存関係は {ANY\_NODE} です。

同じリソースグループ内の2つのリソース間のリソース依存関係は、常に LOCAL\_NODE です。

修飾子を指定しない場合は、FROM\_RG\_AFFINITIES がデフォルトで使用されます。

カテゴリ: 任意

デフォルト: 空のリスト

調整: ANYTIME

Resource\_name(string)

リソースインスタンスの名前です。この名前はクラスタ構成内で一意にする必要があります。リソースが作成されたあとで変更はできません。

カテゴリ: 必要

デフォルト: デフォルトなし

調整: NONE

Resource\_project\_name(string)

リソースに関連付けられた Solaris プロジェクト名。このプロパティは、CPU の共有、クラスタデータサービスのリソースプールといった Solaris のリソース管理機能に適用できます。RGM は、リソースをオンラインにすると、このプロジェクト名を持つ関連プロセスを起動します。このプロパティが指定されていない場合、プロジェクト名は、リソースを含むリソースグループの `RG_project_name` プロパティから取得されます (`rg_properties(5)` のマニュアルページを参照)。どちらのプロパティも指定されなかった場合、RGM は事前定義済みのプロジェクト名 `default` を使用します。プロジェクトデータベース内に存在するプロジェクト名を

指定する必要があります (projects(1) のマニュアルページと『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照)。

このプロパティは Solaris 9 OS からサポートされるようになりました。

---

注- このプロパティへの変更は、リソースが次回起動されるときに有効になります。

---

カテゴリ: 任意  
デフォルト: NULL  
調整: ANYTIME

各クラスタノードまたはゾーン上の Resource\_state (enum)  
RGM が判断した各クラスタノードまたはゾーン上のリソースの状態。この状態には、Online、Offline、Start\_failed、Stop\_failed、Monitor\_failed、Online\_not\_monitored、Starting、Stopping があります。

ユーザーはこのプロパティを構成できません。

カテゴリ: 照会のみ  
デフォルト: デフォルトなし  
調整: NONE

Retry\_count (integer)

起動に失敗したリソースをモニターが再起動する回数です。

Retry\_count を超えると、特定のデータサービス、および Failover\_mode プロパティの設定に応じて、モニターは次のいずれかのアクションを実行します。

- リソースが障害状態であったとしても、リソースグループが現在の主ノードまたはゾーン上にとどまることを許可する
- 別のノードまたはゾーンへのリソースグループのフェイルオーバーを要求します。

このプロパティは RGM によって作成されます。このプロパティは RTR ファイルに宣言されている場合にかぎり、クラスタ管理者は使用を許可されます。RTR ファイル内でデフォルト値が指定されている場合、このプロパティは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティの Tunable 値は WHEN\_DISABLED になります。

---

注- このプロパティにマイナスの値を指定すると、モニターは無限回リソースの再起動を試みます。

ただし、一部のリソースタイプでは、`Retry_count` に負の値を設定できません。より確実に無制限の再起動を指定するには、次の手順を実行します。

- `Retry_interval` に 1 や 0 などの小さい値を指定します。
  - `Retry_count` に 1000 などの大きい値を指定します。
- 

カテゴリ: 条件付き

デフォルト: 上記を参照

調整: `WHEN_DISABLED`

#### `Retry_interval` (integer)

失敗したリソースを再起動するまでの秒数。リソースモニターは、このプロパティと `Retry_count` を組み合わせて使用します。このプロパティは RGM によって作成されます。RTR ファイルに宣言されている場合にかぎり、クラスタ管理者は使用を許可されます。RTR ファイル内でデフォルト値が指定されている場合、このプロパティは任意です。

RTR ファイル内に `Tunable` 属性が指定されていない場合、このプロパティの `Tunable` 値は `WHEN_DISABLED` になります。

カテゴリ: 条件付き

デフォルト: デフォルトなし (上記を参照)

調整: `WHEN_DISABLED`

#### `Scalable` (boolean)

リソースがスケーラブルであるかどうか、つまり、リソースが Sun Cluster ソフトウェアのネットワーク負荷分散機能を使用するかどうかを表します。

---

注- スケーラブルなリソースグループ (ネットワーク負荷分散を使用) を、非大域ゾーンで動作するよう構成することができます。ただし、そのようなスケーラブルなリソースグループを実行できるのは、物理ノードごとに 1 つのゾーン内だけです。

---

このプロパティが RTR ファイルで宣言されている場合は、そのタイプのリソースに対して、RGM は、次のスケーラブルサービスプロパティを自動的に作成します。 `Affinity_timeout`、`Load_balancing_policy`、`Load_balancing_weights`、`Network_resources_used`、`Port_list`、`UDP_affinity`、`Weak_affinity`。これらのプ

ロパティは、RTR ファイル内で明示的に宣言されないかぎり、デフォルト値を持ちます。RTR ファイルに Scalable が宣言されている場合、このプロパティのデフォルトは TRUE です。

RTR ファイルにこのプロパティが宣言されている場合、AT\_CREATION 以外の Tunable 属性の割り当ては許可されません。

RTR ファイルにこのプロパティが宣言されていない場合、このリソースはスケラブルではないため、このプロパティを調整することはできません。RGM は、スケラブルサービスプロパティをいっさい設定しません。ただし、Network\_resources\_used および Port\_list プロパティは、RTR ファイルで明示的に宣言できます。これらのプロパティは、スケラブルサービスでも非スケラブルサービスでも有用です。

このリソースプロパティと Failover リソースタイププロパティの併用については、r\_properties(5)のマニュアルページで詳しく説明されています。

カテゴリ: 任意  
デフォルト: デフォルトなし  
調整: AT\_CREATION

各コールバックメソッドの Start\_timeout ( integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間(秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意  
デフォルト: RTR ファイルにメソッド自体が宣言されている場合は 3600 (1 時間)  
調整: ANYTIME

各クラスターノードまたはゾーン上の Status (enum)

scha\_resource\_setstatus コマンドまたは scha\_resource\_setstatus() 関数または scha\_resource\_setstatus\_zone() 関数を使用してリソースモニターにより設定されます。取り得る値は OK、DEGRADED、FAULTED、UNKNOWN、および OFFLINE です。リソースがオンラインまたはオフラインになったとき、RGM は自動的に Status 値を設定します (Status 値をリソースのモニターまたはメソッドが設定していない場合)。

カテゴリ: 照会のみ  
デフォルト: デフォルトなし  
調整: NONE

**各クラスターノードまたはゾーン上の Status\_msg (string)**

リソースモニターによって、Status プロパティと同時に設定されます。リソースがオンラインまたはオフラインにされると、RGMは自動的にこのプロパティを空文字列でリセットします。ただし、このプロパティがリソースのメソッドによって設定される場合を除きます。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

**各コールバックメソッドの Stop\_timeout (integer)**

RGMがメソッドの呼び出しに失敗したと判断するまでの時間(秒)。特定のリソースタイプに関して、タイムアウトのプロパティはRTRファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTRファイルにメソッド自体が宣言されている場合は3600(1時間)

調整: ANYTIME

**Thorough\_probe\_interval (integer)**

高オーバーヘッドのリソース障害検証の呼び出し間隔(秒)。このプロパティはRGMによって作成されます。RTRファイルに宣言されている場合にかぎり、クラスタ管理者は使用を許可されます。RTRファイル内でデフォルト値が指定されている場合、このプロパティは任意です。

RTRファイル内にTunable属性が指定されていない場合、このプロパティのTunable値はWHEN\_DISABLEDになります。

カテゴリ: 条件付き

デフォルト: デフォルトなし

調整: WHEN\_DISABLED

**Type (string)**

このリソースがインスタントであるリソースタイプ。

カテゴリ: 必要

デフォルト: デフォルトなし

調整: NONE

**Type\_version (string)**

現在このリソースに関連付けられているリソースタイプのバージョンを指定します。このプロパティはRTRファイル内に宣言できません。したがって、RGMによって自動的に作成されます。このプロパティの値は、リソースタイプの

RT\_version プロパティと等しくなります。リソースの作成時、Type\_version プロパティはリソースタイプ名の接尾辞として表示されるだけで、明示的には指定されません。リソースを編集すると、Type\_version プロパティが新しい値に変更されることがあります。

このプロパティの調整については、次の情報から判断されます。

- 現在のリソースタイプのバージョン
- RTR ファイル内の `#$upgrade_from` ディレクティブ

カテゴリ: 説明を参照

デフォルト: デフォルトなし

調整: 説明を参照

#### UDP\_affinity (boolean)

このプロパティが TRUE に設定されている場合、指定のクライアントからの UDP トラフィックはすべて、現在クライアントのすべての TCP トラフィックを処理している同じサーバーノードに送信されます。

このプロパティは、Load\_balancing\_policy が Lb\_sticky または Lb\_sticky\_wild の場合にかぎり有効です。さらに、Weak\_affinity が FALSE に設定されている必要があります。

このプロパティは、スケーラブルサービス専用です。

カテゴリ: 任意

デフォルト: デフォルトなし

調整: WHEN\_DISABLED

#### 各コールバックメソッドの Update\_timeout (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は 3600 (1 時間)

調整: ANYTIME

#### 各コールバックメソッドの Validate\_timeout (integer)

RGM がメソッドの呼び出しに失敗したと判断するまでの時間 (秒)。特定のリソースタイプに関して、タイムアウトのプロパティは RTR ファイルで宣言されているメソッドに対してのみ定義されます。

カテゴリ: 条件付き/任意

デフォルト: RTR ファイルにメソッド自体が宣言されている場合は 3600 (1 時間)

調整: ANYTIME

Weak\_affinity (boolean)

このプロパティが TRUE に設定されている場合、このプロパティにより弱い形式のクライアントアフィニティが有効になります。

弱い形式のクライアントアフィニティが有効になっている場合、特定のクライアントからの接続は、次の場合を除き、同じサーバーノードに送信されます。

- たとえば、障害モニターが再起動したとき、リソースがフェイルオーバーまたはスイッチオーバーしたとき、あるいは、ノードが障害の後にクラスタに参加し直したときにサーバーのリスナーが起動する場合。
- クラスタ管理者により管理アクションが実行されたため、スケーラブルリソースの Load\_balancing\_weights が変更された場合。

弱いアフィニティはメモリーの消費とプロセッササイクルの点で、デフォルトの形式よりもオーバーヘッドを低く抑えられます。

このプロパティは、Load\_balancing\_policy が Lb\_sticky または Lb\_sticky\_wild の場合にかぎり有効です。

このプロパティは、スケーラブルサービス専用です。

カテゴリ: 任意

デフォルト: デフォルトなし

調整: WHEN\_DISABLED

## リソースグループのプロパティ

以下に、Sun Cluster ソフトウェアにより定義されるリソースグループのプロパティを示します。

プロパティ値は以下のように分類されます。

- 必須。クラスタ管理者は、管理ユーティリティを使ってリソースグループを作成するときに値を指定してください。
- 任意。クラスタ管理者がリソースグループの作成時に値を指定しないと、システムのデフォルト値が使用されます。
- 照会のみ。管理ツールで直接設定することはできません。

以下にプロパティ名とその説明を示します。

**Auto\_start\_on\_new\_cluster (boolean)**

このプロパティは、新しいクラスタの形成時にリソースグループマネージャー (RGM) が自動的にリソースグループを起動するかどうかを制御します。デフォルトは TRUE です。

TRUE に設定した場合、クラスタのすべてのノードが同時に再起動すると、RGM はリソースグループを自動的に起動して `Desired primaries` を取得しようとします。

FALSE に設定されていると、クラスタが再起動されたとき、リソースグループは自動的に起動しません。 `clresourcegroup online` コマンドまたは同等の GUI 指令を使用して、最初にリソースグループが手動でオンラインに切り替えられるまで、リソースグループはオフラインのままになります。その後、このリソースグループは通常のフェイルオーバー動作を再開します。

カテゴリ: 任意

デフォルト: TRUE

調整: ANYTIME

**Desired primaries (integer)**

グループが同時に実行できるノードまたはゾーン数として望ましい値。

デフォルトは 1 です。 `Desired primaries` プロパティの値は、 `Maximum primaries` プロパティの値以下にしてください。

カテゴリ: 任意

デフォルト: 1

調整: ANYTIME

**Failback (boolean)**

ノードまたはゾーンがクラスタに結合した場合、グループがオンラインとなるノード群またはゾーン群を再計算するかどうかを示すブール値。再計算により、RGM は優先度の低いノードまたはゾーンをオフラインにし、優先度の高いノードまたはゾーンをオンラインにすることができます。

カテゴリ: 任意

デフォルト: FALSE

調整: ANYTIME

**Global\_resources\_used (string\_array)**

クラスタファイルシステムがこのリソースグループ内のリソースによって使用されるかどうかを指定します。クラスタ管理者はアスタリスク (\*) か空文字列 ("") を指定できます。すべてのグローバルリソースを指定するときはアスタリスク、グローバルリソースを一切指定しない場合は空文字列を指定します。

カテゴリ: 任意

デフォルト: すべてのグローバルリソース

調整: ANYTIME

#### Implicit\_network\_dependencies (boolean)

TRUE の場合、RGM は、グループ内のネットワークアドレスリソースで非ネットワークアドレスリソースに対する強い依存を強制します。このとき、RGM は、すべてのネットワークアドレスリソースを起動してからその他のリソースを起動します。また、グループ内のその他のすべてのリソースを停止してからネットワークアドレスリソースを停止します。ネットワークアドレスリソースには、論理ホスト名と共有アドレスリソースタイプがあります。

スケーラブルなリソースグループ内では、このプロパティの影響はありません。これは、スケーラブルなリソースグループにはネットワークアドレスリソースが含まれないからです。

カテゴリ: 任意

デフォルト: TRUE

調整: ANYTIME

#### Maximum primaries (integer)

グループを同時にオンラインにできるノードまたはゾーンの最大数です。

RG\_mode プロパティが Failover である場合、このプロパティの値は 1 以下にしてください。RG\_mode プロパティが Scalable である場合、1 より大きな値に設定できます。

カテゴリ: 任意

デフォルト: 1

調整: ANYTIME

#### Nodelist (string\_array)

リソースグループを優先度順にオンラインにできるクラスタノードまたはゾーンのリストです。これらのノードまたはゾーンは、リソースグループの潜在的な主ノードまたはマスターです。

カテゴリ: 任意

デフォルト: すべてのクラスタノードの順不同のリスト

調整: ANYTIME

#### Pathprefix (string)

リソースグループ内のリソースが重要な管理ファイルを書き込むことができるクラスタファイルシステム内のディレクトリ。一部のリソースの必須プロパティです。各リソースグループの Pathprefix は、一意にする必要があります。

カテゴリ: 任意

デフォルト: 空の文字列

調整: ANYTIME

#### Pingpong\_interval (integer)

次の条件で、RGMがリソースグループをオンラインにする場所を決定するとき使用する負でない整数値(秒)。

- 再構成が発生している場合。
- GIVEOVER引数付きで `scha_control` コマンドを実行した、または `SCHA_GIVEOVER` 引数付きで `scha_control()` 関数を実行した結果として。

再構成が発生したときは、`Pingpong_interval` で指定した秒数内に特定のノードまたはゾーン上で複数回、リソースグループがオンラインになれない場合があります。この障害が発生した原因は、リソースの `Start` または `Prenet_start` メソッドがゼロ以外で終了したか、タイムアウトしたかのどちらかです。その結果、そのノードまたはゾーンはリソースグループのホストとしては不適切と判断され、RGMは別のマスターを探します。

`scha_control` コマンドまたは `scha_control -0 GIVEOVER` コマンドが特定のノードまたはゾーン上でリソースによって実行され、それによりそのリソースグループが別のノードまたはゾーンにフェイルオーバーした場合、`Pingpong_interval` 秒が経過するまで、(`scha_control` コマンドが実行された)最初のノードまたはゾーンは、同じリソースによる別の `scha_control -0 GIVEOVER` の宛先になることはできません。

カテゴリ: 任意

デフォルト: 3600 (1時間)

調整: ANYTIME

#### Resource\_list (string\_array)

グループ内に含まれるリソースのリストです。クラスタ管理者はこのプロパティを直接設定しません。このプロパティは、クラスタ管理者がリソースグループにリソースを追加したりリソースグループからリソースを削除したりすると、RGMにより自動的に更新されます。

カテゴリ: 照会のみ

デフォルト: デフォルトなし

調整: NONE

#### RG\_affinities (string)

RGMは、別の特定のリソースグループの現在のマスターであるノードまたはゾーンにリソースグループを配置するか(肯定的なアフィニティーの場合)、あるいは、特定のリソースグループの現在のマスターでないノードまたはゾーン上にリソースグループを配置(否定的なアフィニティーの場合)しようとしています。

RG\_affinities には次の文字列を設定できます。

- ++ (強い肯定的なアフィニティー)
- + (弱い肯定的なアフィニティー)
- - (弱い否定的なアフィニティー)
- -- (強い否定的なアフィニティー)
- +++ (フェイルオーバー委託付きの強い肯定的なアフィニティー)

たとえば、RG\_affinities=+RG2,--RG3 は、このリソースグループが RG2 に対しては弱い肯定的なアフィニティーを持っており、RG3 に対しては強い否定的なアフィニティーを持っていることを示します。

RG\_affinities プロパティの使い方については、『Sun Cluster データサービスの計画と管理 (Solaris OS 版)』の第 2 章「データサービスリソースの管理」を参照してください。

カテゴリ: 任意

デフォルト: 空の文字列

調整: ANYTIME

#### RG\_dependencies (string\_array)

同じノードまたはゾーン上の別のグループをオンライン/オフラインにするときの優先順位を示すリソースグループのリスト (任意)。すべての強い RG\_affinities (肯定的と否定的) と RG\_dependencies が一緒のグループは、サイクルを含むことが許されません。

たとえば、リソースグループ RG1 の RG\_dependencies リストにリソースグループ RG2 がリストされている、つまり RG1 が RG2 に対してリソースグループの依存関係を持っているとします。

次のリストに、リソースグループ依存関係の影響を要約します。

- ノードまたはゾーンがクラスタに結合されると、そのノードまたはゾーンでは、RG2 のすべてのリソースに対する Boot メソッドが終わってから、RG1 のリソースに対する Boot メソッドが実行されます。
- RG1 と RG2 が両方とも同じノードまたはゾーン上で同時に PENDING\_ONLINE 状態である場合、RG2 内のすべてのリソースが自分の開始メソッドを完了するまで、RG1 内のどのリソースでも開始メソッド (Prenet\_start または Start) は実行されません。
- RG1 と RG2 が両方とも同じノードまたはゾーン上で同時に PENDING\_OFFLINE 状態である場合、RG1 内のすべてのリソースが自分の停止メソッドを完了するまで、RG2 内のどのリソースでも停止メソッド (Stop または Postnet\_stop) は実行されません。

- RG1 または RG2 の主ノードをスイッチする場合、それによって RG1 がいずれかのノードまたはゾーンでオンラインに、RG2 がすべてのノードまたはゾーンでオフラインになる場合は、このスイッチは失敗します。詳細は、`clresourcegroup(1CL)` および `clsetup(1CL)` のマニュアルページを参照してください。
- RG2 上で `Desired primaries` プロパティをゼロに設定した場合、RG1 上で `Desired primaries` プロパティをゼロより大きな値に設定することは許可されません。
- RG2 に対する `Auto_start_on_new_cluster` が `FALSE` に設定されている場合は、RG1 に対する `Auto_start_on_new_cluster` プロパティを `TRUE` に設定することはできません。

カテゴリ: 任意  
デフォルト: 空のリスト  
調整: ANYTIME

`RG_description` (string)  
リソースグループの簡単な説明です。

カテゴリ: 任意  
デフォルト: 空の文字列  
調整: ANYTIME

`RG_is_frozen` (boolean)  
あるリソースグループが依存している広域デバイスをスイッチオーバーするかどうかを表します。このプロパティが `TRUE` に設定されている場合、広域デバイスはスイッチオーバーされます。このプロパティが `FALSE` に設定されている場合、広域デバイスはスイッチオーバーされません。リソースグループが広域デバイスに依存するかどうかは、`Global_resources_used` プロパティの設定によります。

`RG_is_frozen` プロパティをユーザーが直接設定することはありません。`RG_is_frozen` プロパティは、広域デバイスのステータスが変わったときに、RGMによって更新されます。

カテゴリ: 任意  
デフォルト: デフォルトなし  
調整: NONE

`RG_mode` (enum)  
リソースグループがフェイルオーバーグループなのか、スケラブルグループなのかを指定します。この値が `Failover` であれば、RGM はグループの `Maximum primaries` プロパティの値を 1 に設定し、リソースグループのマスターを単一のノードまたはゾーンに制限します。

このプロパティの値が `Scalable` であれば、`Maximum primaries` プロパティは 1 より大きな値に設定されることがあります。その結果、このグループのマスターが同時に複数存在する可能性があります。`Failover` プロパティの値が `TRUE` のリソースを、`RG_mode` の値が `Scalable` のリソースグループに追加することはできません。

`Maximum primaries` が 1 である場合、デフォルトは `Failover` です。  
`Maximum primaries` が 1 より大きい場合、デフォルトは `Scalable` です。

カテゴリ: 任意

デフォルト: `Maximum primaries` の値によります。

調整: `NONE`

#### `RG_name` (string)

リソースグループの名前。これは必須プロパティです。この値は、クラスタ内で一意にしてください。

カテゴリ: 必要

デフォルト: デフォルトなし

調整: `NONE`

#### `RG_project_name` (string)

リソースグループに関連付けられた Solaris プロジェクト名 (`projects(1)` のマニュアルページを参照)。このプロパティは、CPU の共有、クラスタデータサービスのリソースプールといった Solaris のリソース管理機能に適用できます。RGM は、リソースグループをオンラインにすると、`Resource_project_name` プロパティセットを持たないリソース用として、このプロジェクト名下で関連プロセスを起動します (`r_properties(5)` のマニュアルページを参照)。指定されたプロジェクト名は、プロジェクトデータベースに存在する必要があります (`projects(1)` のマニュアルページ、および『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照)。

このプロパティは Solaris 9 OS からサポートされるようになりました。

---

注- このプロパティへの変更は、リソースの次回起動時に有効になります。

---

カテゴリ: 任意

デフォルト: テキスト文字列「default」

調整: `ANYTIME`

#### `RG_slm_cpu` (decimal number)

`RG_slm_type` プロパティが `AUTOMATED` に設定されている場合、この数は CPU シェアの数およびプロセッサセットのサイズの計算の基準になります。

---

注-RG\_slm\_cpu プロパティを使用できるのは、RG\_slm\_type が AUTOMATED に設定されている場合のみです。詳細は、「RG\_slm\_type プロパティ」を参照してください。

---

RG\_slm\_cpu プロパティの最大値は 655 です。小数点のあとには 2 桁を含めることができます。RG\_slm\_cpu プロパティには 0 を指定しないでください。シェアの値を 0 に設定すると、CPU 負荷が高い場合に、公平配分スケジューラ (FFS) によりリソースをスケジューリングできない場合があります。

リソースグループがオンラインである間に RG\_slm\_cpu プロパティに対して行う変更は、動的に考慮されます。

RG\_slm\_type プロパティは AUTOMATED に設定されているため、Sun Cluster は SCSLM\_resourcegroupname という名前のプロジェクトを作成します。resourcegroupname は、ユーザーがリソースグループに割り当てる実際の名前を表します。リソースグループに属するリソースの各メソッドは、このプロジェクトで実行されます。Solaris 10 から、これらのプロジェクトは、リソースグループのゾーンが大域ゾーンであれ非大域ゾーンであれ、リソースグループのゾーン内に作成されます。project(4) のマニュアルページを参照してください。

プロジェクト SCSLM\_resourcegroupname の project.cpu-shares 値は、RG\_slm\_cpu のプロパティ値の 100 倍です。RG\_slm\_cpu プロパティが設定されていない場合、このプロジェクトは値 1 の project.cpu-shares を使用して作成されます。RG\_slm\_cpu プロパティのデフォルト値は 0.01 です。

Solaris 10 OS から、RG\_slm\_pset\_type プロパティが DEDICATED\_STRONG または DEDICATED\_WEAK に設定されている場合、プロセッサセットのサイズの計算には RG\_slm\_cpu プロパティが使用されます。また、RG\_slm\_cpu プロパティは zone.cpu-shares の値の計算にも使用されます。

プロセッサセットについては、『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照してください。

カテゴリ: 任意

デフォルト: 0.01

調整: ANYTIME

RG\_slm\_cpu\_min (decimal number)

アプリケーションが動作できるプロセッサの最小数を決定します。

このプロパティは、次に示す要因がすべて真の場合だけ使用できます。

- `RG_slm_type` プロパティが `AUTOMATED` に設定されている
- `RG_slm_pset_type` プロパティが `DEDICATED_STRONG` または `DEDICATED_WEAK` に設定されている
- `RG_slm_cpu` プロパティが、`RG_slm_cpu_min` プロパティに対して設定されている値以上の値に設定されている
- Solaris 10 OS を使用している

`RG_slm_cpu_min` プロパティの最大値は 655 です。小数点のあとには 2 桁を含めることができます。`RG_slm_cpu_min` プロパティには 0 を指定しないでください。`RG_slm_cpu_min` および `RG_slm_cpu` プロパティは、それぞれ、Sun Cluster が生成するプロセッサセットに対して `pset.min` および `pset.max` の値を決定します。

リソースグループがオンラインである間にユーザーが `RG_slm_cpu` および `RG_slm_cpu_min` プロパティに対して行う変更は、動的に考慮されます。`RG_slm_pset_type` プロパティが `DEDICATED_STRONG` に設定され、使用できる CPU が十分でない場合、`RG_slm_cpu_min` プロパティに対してユーザーが要求した変更は無視されます。この場合は、警告メッセージが表示されます。次のスイッチオーバー時に、`RG_slm_cpu_min` プロパティが使用できる CPU が十分でない場合、CPU の不足によるエラーが発生する可能性があります。

プロセッサセットについては、『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照してください。

カテゴリ: 任意

デフォルト: 0.01

調整: ANYTIME

#### `RG_slm_type` (string)

システムリソースの使用状況を管理できるようにし、システムリソース管理用に Solaris オペレーティングシステムを設定する手順の一部を自動化します。

`RG_SLM_type` が取り得る値は `AUTOMATED` と `MANUAL` です。

`RG_slm_type` プロパティを `AUTOMATED` に設定した場合、リソースグループは CPU 使用率の制御とともに起動します。

その結果、Sun Cluster は次の処理を行います。

- `SCSLM_resourcegroupname` という名前のプロジェクトを作成します。このリソースグループ内のリソースのすべてのメソッドは、このプロジェクト内で実行されます。このプロジェクトは、このリソースグループ内のリソースのメソッドがノードまたはゾーンで初めて実行されるときに作成されます。
- プロジェクトと関連付けられている `project.cpu_shares` の値を、`RG_slm_cpu` プロパティの 100 倍の値に設定します。デフォルトでは、`project.cpu_shares` の値は 1 です。

- Solaris 10 OS からは、`zone.cpu_shares` を、すべてのオンラインリソースグループの `RG_slm_cpu` プロパティの合計の 100 倍に設定します。またこのプロパティは、当該ゾーン内で `RG_slm_type` を `AUTOMATED` に設定します。ゾーンは大域または非大域の場合があります。非大域ゾーンは、Sun Cluster が生成するプールにバインドされます。オプションで、`RG_slm_pset_type` プロパティが `DEDICATED_WEAK` または `DEDICATED_STRONG` に設定されている場合、Sun Cluster の生成されたプールは、Sun Cluster の生成されたプロセッサセットと関連付けられます。専用のプロセッサセットについては、`RG_slm_pset_type` プロパティの説明を参照してください。`RG_slm_type` プロパティを `AUTOMATED` に設定した場合、実行されるすべての処理はログに記録されます。

`RG_slm_type` プロパティを `MANUAL` に設定した場合、`RG_project_name` プロパティにより指定されているプロジェクト内でリソースグループが実行されません。

リソースプールとプロセッサセットについては、『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照してください。

---

注 -

- 58 文字を超えるリソースグループ名は指定しないでください。リソースグループ名が 58 文字を超える場合、CPU 制御を構成できなくなる、つまり、`RG_slm_type` プロパティに `AUTOMATED` を設定できなくなります。
- リソースグループ名にはダッシュ (-) を含めないでください。Sun Cluster ソフトウェアは、プロジェクトの作成時に、リソースグループ名にあるすべてのダッシュを下線 (\_) に置き換えます。たとえば、Sun Cluster が `rg-dev` というリソースグループに対して `SCSLM_rg_dev` というプロジェクトを作成する場合があります。Sun Cluster がリソースグループ `rg-dev` に対してプロジェクトを作成しようとするとき、`rg_dev` という名前のリソースグループがすでに存在する場合、衝突が発生します。

---

カテゴリ: 任意  
デフォルト: `manual`  
調整: `ANYTIME`

`RG_slm_pset_type(string)`  
専用のプロセッサセットの作成を可能にします。

このプロパティは、次に示す要因がすべて真の場合だけ使用できます。

- `RG_slm_type` プロパティが `AUTOMATED` に設定されている
- Solaris 10 OS を使用している
- リソースグループが非大域ゾーンで実行される

RG\_slm\_pset\_type の取り得る値は DEFAULT、DEDICATED\_STRONG、および DEDICATED\_WEAK です。

リソースグループが DEDICATED\_STRONG または DEDICATED\_WEAK として実行される場合は、そのリソースグループのノードリストには非大域ゾーンだけが存在するようにリソースグループを設定してください。

非大域ゾーンは、デフォルトプールである POOL\_DEFAULT 以外のプールに対して設定しないでください。ゾーン構成の詳細は、zonecfg(1M) のマニュアルページを参照してください。非大域ゾーンは、デフォルトプール以外のプールに動的にバインドしないでください。プールバインディングの詳細は、poolbind(1M) のマニュアルページを参照してください。バインドされた2つのプールの状態は、リソースグループ内のリソースのメソッドが起動されている場合だけ、確認されません。

DEDICATED\_STRONG と DEDICATED\_WEAK の値は、ノードリストに同じゾーンを持つリソースグループと相互に排他的です。同じゾーン内では、一部のリソースグループの RG\_slm\_pset\_type が DEDICATED\_STRONG に設定され、ほかのリソースグループの RG\_slm\_pset\_type が DEDICATED\_WEAK に設定されるように、リソースグループを構成することはできません。

RG\_slm\_pset\_type プロパティを DEDICATED\_STRONG に設定した場合、Sun Cluster は、RG\_slm\_type プロパティが AUTOMATED に設定されている場合に RG\_slm\_type プロパティにより実行されるアクション以外にも、次の処理を行います。

- プールを作成し、リソースグループが PRENET\_START メソッドと START メソッドの一方または両方に対して起動する非大域ゾーンにそのプールを動的にバインドする。
- 次の合計の間のサイズを持つプロセッサセットを作成する。
  - 当該リソースグループが起動するゾーンでオンラインであるすべてのリソースグループ内の RG\_slm\_cpu\_min プロパティの合計。
  - 当該ゾーンで実行中であるリソースグループ内の RG\_slm\_cpu プロパティの合計。

STOP メソッドまたは POSTNET\_STOP メソッドのいずれかが実行中である場合、Sun Cluster の生成されたプロセッサセットは破棄されます。リソースグループがゾーン内でオンラインでなくなった場合、そのプールは破棄され、非大域ゾーンはデフォルトのプール (POOL\_DEFAULT) にバインドされます。

- プロセッサセットをプールに関連付けます。
- ゾーンを実行しているすべてのリソースグループの RG\_slm\_cpu プロパティの合計の 100 倍に zone.cpu\_shares を設定します。

ユーザーが RG\_slm\_pset\_type プロパティを DEDICATED\_WEAK に設定した場合、リソースグループの動作は、RG\_slm\_pset\_type が DEDICATED\_STRONG に設定されてい

る場合と同じようになります。しかし、プロセッサセットの作成に十分なプロセッサを使用できない場合、プールはデフォルトのプロセッサセットに関連付けられます。

ユーザーが `RG_slm_pset_type` プロパティを `DEDICATED_STRONG` に設定し、またプロセッサセットの作成に十分なプロセッサを使用できない場合、エラーが発生します。その結果、リソースグループは当該ノードまたはゾーン上では起動しません。

CPU が割り当てられている場合、`DEFAULTPSETMIN` 最小サイズは `DEDICATED_STRONG` よりも優先されます。`DEDICATED_STRONG` は `DEDICATED_WEAK` よりも優先されます。ただし、`clnode` コマンドを使用してデフォルトのプロセッサのサイズを大きくし、また十分なプロセッサが使用できない場合、この優先順位は無視されます。`DEFAULTPSETMIN` プロパティの詳細については、`clnode(1CL)` のマニュアルページを参照してください。

`clnode` コマンドは、デフォルトのプロセッサセットに最小限の CPU を動的に割り当てます。ユーザーが指定した CPU の数が使用できない場合、Sun Cluster は定期的にこの数の CPU を割り当てようとします。それに失敗すると、CPU の最小数が割り当てられるまで、Sun Cluster はデフォルトのプロセッサセットにより少ない数の CPU を割り当てようとします。このアクションは一部の `DEDICATED_WEAK` プロセッサセットを破棄する場合がありますが、`DEDICATED_STRONG` プロセッサセットを破棄することはありません。

ユーザーが `RG_slm_pset_type` プロパティを `DEDICATED_STRONG` に設定したリソースグループを起動する場合、`DEDICATED_WEAK` プロセッサセットと関連付けられたプロセッサセットが破棄される場合があります。このリソースグループがこのような動作を行う場合があるのは、両方のプロセッサセットのノードまたはゾーン上で十分な CPU が使用できない場合です。この場合、`DEDICATED_WEAK` プロセッサセットで動作しているリソースグループのプロセスは、デフォルトのプロセッサセットに関連付けられます。

`DEDICATED_STRONG` または `DEDICATED_WEAK` の間で `RG_slm_pset_type` プロパティの値を交換するには、まずその値をデフォルトに設定します。

CPU 制御に対して構成されたリソースグループが非大域ゾーンでオンラインではない場合、CPU シェアの値はそのゾーンの `zone.cpu-shares` に設定されます。デフォルトでは、`zone.cpu-shares` は 1 に設定されています。ゾーン構成の詳細については、`zonecfg(1M)` のマニュアルページを参照してください。

ユーザーが `RG_slm_pset_type` プロパティを `DEFAULT` に設定すると、Sun Cluster は `SCSLM_pool_zonename` という名前のプールを作成しますが、プロセッサセットは作成しません。この場合、`SCSLM_pool_zonename` はデフォルトのプロセッサセットに関連付けられます。ゾーンに割り当てられるシェアは、そのゾーン内のすべてのリソースグループの `RG_slm_cpu` の値の合計と等しくなります。

リソースプールとプロセッサセットについては、『System Administration Guide: Solaris Containers-Resource Management and Solaris Zones』を参照してください。

カテゴリ: 任意

デフォルト: default

調整: ANYTIME

各クラスタノードまたはゾーン上の RG\_state (enum)

RGMにより UNMANAGED、ONLINE、OFFLINE、PENDING\_ONLINE、PENDING\_OFFLINE、ERROR\_STOP\_FAILED、ONLINE\_FAULTED、または PENDING\_ONLINE\_BLOCKED に設定され、各クラスタノードまたはゾーン上のグループの状態を表します。

ユーザーはこのプロパティを構成できません。しかし、clresourcegroup コマンドを実行することによって、あるいは同等の clsetup や Sun Cluster Manager コマンドを使用して、このプロパティを間接的に設定することは可能です。RGM の制御下にはないときは、グループは UNMANAGED 状態で存在することができます。

各状態の説明は次のとおりです。

---

注- すべてのノードまたはゾーンに適用される UNMANAGED 状態を除き、状態は個別のノードまたはゾーンにのみ適用されます。たとえば、リソースグループがノード A のゾーン 1 では OFFLINE であり、ノード B のゾーン 2 では PENDING\_ONLINE である場合があります。

---

#### UNMANAGED

新しく作成されたリソースグループの最初の状態や、前に管理されていたリソースグループの状態。そのグループのリソースに対して Init メソッドがまだ実行されていないか、そのグループのリソースに対して Fini メソッドがすでに実行されています。

このグループは RGM によって管理されていません。

#### ONLINE

リソースグループはノードまたはゾーン上ですでに起動されています。つまり、各リソースに適用可能な起動メソッド Preenet\_start、Start、および Monitor\_start は、グループ内のすべての有効なリソースに対して正常に実行されました。

OFFLINE	リソースグループはノードまたはゾーンですでに停止されています。つまり、各リソースに適用可能な停止メソッド <code>Monitor_stop</code> 、 <code>Stop</code> 、および <code>Postnet_stop</code> はグループ内のすべての有効なリソースに対して正常に実行されました。さらに、リソースグループがノードまたはゾーンで最初に起動されるまでは、グループにこの状態が適用されます。
PENDING_ONLINE	リソースグループはノードまたはゾーン上で起動中です。各リソースに適用可能な起動メソッド <code>Prenet_start</code> 、 <code>Start</code> 、および <code>Monitor_start</code> はグループ内の有効なリソースに対して実行中です。
PENDING_OFFLINE	リソースグループはノードまたはゾーン上で停止中です。各リソースに適用可能な停止メソッド <code>Monitor_stop</code> 、 <code>Stop</code> 、および <code>Postnet_stop</code> はグループ内の有効なリソースに対して実行中です。
ERROR_STOP_FAILED	リソースグループ内の1つまたは複数のリソースが停止に失敗し、 <code>Stop_failed</code> 状態になっています。グループのほかのリソースがオンラインまたはオフラインである可能性があります。 <code>ERROR_STOP_FAILED</code> 状態がクリアされるまで、このリソースグループはノードまたはゾーン上での起動が許可されません。  <code>clresource clear</code> などの管理コマンドを使用して <code>Stop_failed</code> リソースを手動で終了させ、その状態を <code>OFFLINE</code> に再設定します。
ONLINE_FAULTED	リソースグループは <code>PENDING_ONLINE</code> で、このノードまたはゾーン上での起動が完了しています。ただし、1つまたは複数のリソースが <code>START_FAILED</code> 状態または <code>FAULTED</code> 状態で終了しています。
PENDING_ONLINE_BLOCKED	リソースグループは、完全な起動を行うことに失敗しました。これは、リソースグループの1つまたは複数のリソースが、ほかのリソースグループのリソース

に対して強いリソース依存性があり、それが満たされていないためです。このようなリソースは `OFFLINE` のままになります。リソースの依存関係が満たされている場合、リソースグループは自動的に `PENDING_ONLINE` 状態に戻ります。

カテゴリ: 照会のみ  
 デフォルト: デフォルトなし  
 調整: NONE

#### `Suspend_automatic_recovery` (boolean)

リソースグループの自動復旧が中断されるかどうかを指定するブール値です。クラスタ管理者が自動復旧を再開するコマンドを明示的に実行するまで、中断されたリソースグループが自動的に再開またはフェイルオーバーされることはありません。中断されたデータサービスは、オンラインかオフラインにかかわらず、現在の状態のままとなります。指定したノードまたはゾーン上でリソースグループの状態を手作業で切り替えることもできます。また、リソースグループ内の個々のリソースも有効または無効にできます。

`Suspend_automatic_recovery` プロパティに `TRUE` が設定されると、リソースグループの自動復旧は中断されます。このプロパティが `FALSE` に設定されると、リソースグループの自動復旧が再開され、アクティブになります。

このプロパティを直接設定することはありません。RGM は、クラスタ管理者がリソースグループの自動復旧を中断または再開したときに

`Suspend_automatic_recovery` プロパティの値を変更します。クラスタ管理者は、`clresourcegroup suspend` コマンドで自動復旧を中断します。クラスタ管理者は、`clresourcegroup resume` コマンドで自動復旧を再開します。RG\_system プロパティの設定にかかわらず、リソースグループは中断または再開できます。

カテゴリ: 照会のみ  
 デフォルト: FALSE  
 調整: NONE

#### `RG_system` (boolean)

リソースグループの `RG_system` プロパティの値が `TRUE` の場合、そのリソースグループとそのリソースグループ内のリソースに関する特定の操作が制限されます。この制限は、重要なリソースグループやリソースを間違えて変更または削除してしまうことを防ぐためにあります。このプロパティにより影響を受けるのは `clresourcegroup` コマンドのみです。 `scha_control(1HA)` と `scha_control(3HA)` の操作には影響を与えません。

リソースグループ(またはリソースグループ内のリソース)の制限操作を実行する前には、まず、リソースグループの `RG_system` プロパティを `FALSE` に設定しま

す。クラスタサービスをサポートするリソースグループ(または、リソースグループ内のリソース)を変更または削除するときには注意してください。

操作	例
リソースグループを削除する	<code>clresourcegroup delete RG1</code>
リソースグループプロパティを編集する (RG_systemを除く)	<code>clresourcegroup set -p RG_description=... +</code>
リソースグループへソースを追加する	<code>clresource create -g RG1 -t SUNW.nfs R1</code> リソースは作成後に有効な状態になり、リソース監視も有効になります。
リソースグループからリソースを削除する	<code>clresource delete R1</code>
リソースグループに属するリソースのプロパティを編集する	<code>clresource set -g RG1 -t SUNW.nfs -p r_description="HA-NFS res" R1</code>
リソースグループをオフラインに切り替える	<code>clresourcegroup offline RG1</code>
リソースグループを管理する	<code>clresourcegroup manage RG1</code>
リソースグループを管理しない	<code>clresourcegroup unmanage RG1</code>
リソースグループのリソースを有効にする	<code>clresource enable R1</code>
リソースグループのリソースに対する監視を有効にする	<code>clresource monitor R1</code>
リソースグループのリソースを無効にする	<code>clresource disable R1</code>
リソースの監視を無効にする	<code>clresource unmonitor R1</code>

リソースグループのRG\_systemプロパティの値がTRUEの場合、そのリソースグループで編集可能な唯一のプロパティはRG\_systemプロパティ自身です。つまり、RG\_systemプロパティの編集は無制限です。

カテゴリ: 任意  
 デフォルト: FALSE  
 調整: ANYTIME

# リソースプロパティの属性

この節では、システム定義プロパティの変更または拡張プロパティの作成に使用できるリソースプロパティの属性について説明します。



注意 - boolean、enum、int タイプのデフォルト値に、Null または空の文字列("") は指定できません。

以下にプロパティ名とその説明を示します。

## Array\_maxsize

stringarray タイプの場合、設定できる配列要素の最大数。

## Array\_minsize

stringarray タイプの場合、設定できる配列要素の最小数。

## Default

プロパティのデフォルト値を示します。

## Description

プロパティを簡潔に記述した注記(文字列)。RTR ファイル内でシステム定義プロパティに対する Description 属性を設定することはできません。

## Enumlist

enum タイプの場合、プロパティに設定できる文字列値のセット。

## Extension

リソースタイプの実装によって定義された拡張プロパティが RTR ファイルのエントリで宣言されていることを示します。拡張プロパティが使用されていない場合、そのエントリはシステム定義プロパティです。

## Max

int タイプの場合、プロパティに設定できる最大値。

## Maxlength

string および stringarray タイプの場合、設定できる文字列の長さの最大値。

## Min

int タイプの場合、プロパティに設定できる最小値。

## Minlength

string および stringarray タイプの場合、設定できる文字列の長さの最小値。

## Per\_node

使用された場合、拡張プロパティがノード単位またはゾーン単位で設定できることを示します。

Per\_node プロパティ属性をタイプ定義で指定する場合は、Default プロパティ属性でデフォルト値も指定してください。デフォルト値を指定すると、明示的な

値が割り当てられていないノードまたはゾーン上でノード単位またはゾーン単位のプロパティをユーザーが要求した場合に、値が返されることが保証されません。

タイプ `stringarray` のプロパティには `Per_node` プロパティ属性を指定できません。

#### Property

リソースプロパティの名前。

#### Tunable

クラスタ管理者がリソースのプロパティ値をいつ設定できるかを示します。クラスタ管理者にプロパティの設定を許可しない場合は、`NONE` または `FALSE` に設定します。クラスタ管理者にプロパティの調整を許可する値には、`TRUE` または `ANYTIME` (任意の時点)、`AT_CREATION` (リソースの作成時のみ)、または `WHEN_DISABLED` (リソースが無効のとき) があります。「いつ監視を無効にするか」や「いつオフラインにするか」などのそのほかの条件を確立するには、この属性を `ANYTIME` に設定して `validate` メソッドでリソースの状態を検証します。

デフォルトは、次のエントリに示すように、標準リソースプロパティごとに異なります。RTR ファイルで特に指定していない限り、拡張プロパティを調整する設定のデフォルトは `TRUE (ANYTIME)` です。

#### プロパティの型

指定可能な型は、`string`、`boolean`、`integer`、`enum`、`stringarray` です。RTR ファイル内で、システム定義プロパティの型の属性を設定することはできません。タイプは、RTR ファイルのエントリに登録できる、指定可能なプロパティ値とタイプ固有の属性を決定します。`enum` タイプは、文字列値のセットです。

## データサービスのコード例

---

この付録では、データサービスの各メソッドの完全なコード例を示します。また、リソースタイプ登録 (RTR) ファイルの内容も示します。

この付録の内容は次のとおりです。

- 303 ページの「リソースタイプ登録ファイルのリスト」
- 307 ページの「Start メソッドのコードリスト」
- 310 ページの「Stop メソッドのコードリスト」
- 312 ページの「gettime ユーティリティのコードリスト」
- 313 ページの「PROBE プログラムのコードリスト」
- 319 ページの「Monitor\_start メソッドのコードリスト」
- 321 ページの「Monitor\_stop メソッドのコードリスト」
- 323 ページの「Monitor\_check メソッドのコードリスト」
- 325 ページの「Validate メソッドのコードリスト」
- 329 ページの「Update メソッドのコードリスト」

### リソースタイプ登録ファイルのリスト

RTR ファイルには、クラスタ管理者がデータサービスを登録するとき、データサービスの初期構成を定義するリソースとリソースタイプのプロパティ宣言が含まれています。

例 B-1 SUNW.Sample RTR ファイル

```
#  
# Copyright (c) 1998-2006 by Sun Microsystems, Inc.  
# All rights reserved.  
#  
# ドメインネームサービス (DNS) の登録情報  
#
```

## 例 B-1 SUNW.Sample RTR ファイル (続き)

```
#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;

# リソースタイプ宣言のあとに、中括弧に囲まれたリソースプロパティ宣言
# のリストが続く。プロパティ名宣言は、各エントリの左中括弧の直後にある
# 最初の属性である必要がある。
#
# <method>_timeout プロパティは、RGM がメソッド呼び出しが失敗
# したという結論を下すまでの時間 (秒) を設定する。

# すべてのメソッドタイムアウトの MIN 値は、60 秒に設定されている。
# これは、管理者が短すぎる時間を設定することを防ぐためである。短すぎる
# 時間を設定するとスイッチオーバーやフェイルオーバーの性能が上がらず、
# さらには、予期せぬ RGM アクションが発生する可能性がある(間違った
# フェイルオーバー、ノードの再起動、リソースグループの
# ERROR_STOP_FAILED 状態への移行、オペレータの介入の必要性など)。
# メソッドタイムアウトに短すぎる時間を設定すると、データサービス全体
# の可用性を下げることになる。
{
    PROPERTY = Start_timeout;
    MIN=60;
    DEFAULT=300;
}
{
    PROPERTY = Stop_timeout;
```

## 例 B-1 SUNW.Sample RTR ファイル (続き)

```

        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;
    }

```

# 当該ノード上でアプリケーションを正常に起動できないと結論を下すまで  
# に、ある期間 (Retry\_Interval) に行う再試行の回数。

```

    {
        PROPERTY = Retry_count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
    }

```

# Retry\_interval には 60 の倍数を設定する。これは、秒から分に変換さ  
# れ、端数が切り上げられるためである。たとえば、50 (秒) という値を  
# 指定すると、1 分に変換される。

# このプロパティは再試行数 (Retry\_count) のタイミングを決定する。

```

    {

```

## 例 B-1 SUNW.Sample RTR ファイル (続き)

```
    PROPERTY = Retry_interval;
    MIN=60;
    MAX=3600;
    DEFAULT=300;
    TUNABLE = ANYTIME;
}

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

#
# 拡張プロパティ
#

# クラスタ管理者はこのプロパティの値を設定して、アプリケーションが使用
# する構成ファイルが入っているディレクトリを示す必要がある。このアプリ
# ケーションの場合、DNS は PXFS (通常は named.conf) 上の DNS 構成ファ
# イルのパスを指定する。
{
    PROPERTY = Confdir;
    EXTENSION;
    STRING;
    TUNABLE = AT_CREATION;
    DESCRIPTION = "The Configuration Directory Path";
}

# 検証が失敗したと宣言するまでのタイムアウト値 (秒)
{
    PROPERTY = Probe_timeout;
    EXTENSION;
    INT;
    DEFAULT = 30;
    TUNABLE = ANYTIME;
    DESCRIPTION = "Time out value for the probe (seconds)";
}
```

## Start メソッドのコードリスト

RGM は、データサービスリソースのあるリソースグループがクラスタノードまたはゾーン上でオンラインになると、そのノードまたはゾーン上で Start メソッドを実行します。また、リソースが有効になったときも、RGM は同じ動作をします。サンプルのアプリケーションでは、Start メソッドはそのノードまたは大域ゾーン上で `in.named` (DNS) デーモンを起動します。

### 例 B-2 dns\_svc\_start メソッド

```
#!/bin/ksh
#
# HA-DNS の Start メソッド
#
# このメソッドは PMF の制御下でデータサービスを起動する。DNS の
# in.named プロセスを起動する前に、いくつかの妥当性検査を実行する。
# データサービスの PMF タグは $RESOURCE_NAME.named である。
# PMF は、指定された回数 (Retry_count) だけ、サービスを起動しようとす
# る。そして、指定された期間 (Retry_interval) 内で試行回数がこの値を超えた
# 場合、PMF はサービスの起動に失敗したことを報告する。
# Retry_count と Retry_interval は両方とも RTR ファイルに設定されて
# いるリソースプロパティである。

#pragma ident "@(#)dns_svc_start 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソース名
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されているリソース
                # グループの名前
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプ名
```

## 例 B-2 dns\_svc\_start メソッド (続き)

```
        RESOURCETYPE_NAME=$OPTARG
        ;;

    *)
        logger -p ${SYSLOG_FACILITY}.err \
        -t [$RESOURCECETYPE_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
```

```
# メッセージの記録に使用する syslog 機能番号を取得する。
```

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

```
# このメソッドに渡された引数を構文解析する。
```

```
parse_args "$@"
```

```
PMF_TAG=$RESOURCE_NAME.named
```

```
SYSLOG_TAG=$RESOURCECETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
```

```
# DNS を起動するため、リソースの Confdir プロパティの値を取得する。
```

```
# 入力されたリソース名とリソースグループを使用して、リソースを
```

```
# 追加するときにクラスタ管理者が設定した Confdir の値を見つける。
```

```
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
```

```
-G $RESOURCEGROUP_NAME Confdir`
```

```
# scha_resource_get は拡張プロパティの「タイプ」と「値」を戻す。
```

```
# 拡張プロパティの値だけを取得する。
```

```
CONFIG_DIR=`echo $config_info | awk '{print $2}``
```

```
# $CONFIG_DIR がアクセス可能であるかどうかを検査する。
```

```
if [ ! -d $CONFIG_DIR ]; then
```

```
    logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
```

## 例B-2 dns\_svc\_start メソッド (続き)

```

    “${ARGV0} Directory $CONFIG_DIR missing or not mounted”
    exit 1
fi

# データファイルへの相対パスが存在する場合、$CONFIG_DIR
# ディレクトリに移動する。
cd $CONFIG_DIR

# named.conf ファイルが $CONFIG_DIR ディレクトリ内に存在するか
# を検査する。
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

# RTR ファイルから Retry_count の値を取得する。
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# RTR ファイルから Retry_interval の値を取得する。この値の単位は秒
# であり、pmfadm に渡すときは分に変換する必要がある。変換時、端数は
# 切り上げられるので注意すること。たとえば、50 秒は 1 分に切り上げられる。
((RETRY_INTRVAL = `scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME 60`))

# PMF の制御下で in.named デーモンを起動する。$RETRY_INTERVAL の期
# 間、$RETRY_COUNT の回数だけ、クラッシュおよび再起動できる。どちら
# かの値以上クラッシュした場合、PMF は再起動を中止する。
# というタグですすでにプロセスが登録されて
# いる場合、PMF はすでにプロセスが動作していることを示す警告メッセ
# ージを送信する。
echo “Retry interval is “$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# HA-DNS が起動していることを示すメッセージを記録する。
if [ $? -eq 0 ]; then
    logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
        “${ARGV0} HA-DNS successfully started”
fi
exit 0

```

## Stop メソッドのコードリスト

RGM は、HA-DNS リソースのあるリソースグループがクラスタノードまたはゾーン上でオフラインになると、そのノードまたはゾーン上で Stop メソッドを実行します。また、リソースが無効になったときも、RGM は同じ動作をします。このメソッドは、そのノードまたは大域ゾーン上で `in.named` (DNS) デーモンを停止します。

### 例 B-3 dns\_svc\_stop メソッド

```
#!/bin/ksh
#
# HA-DNS の Stop メソッド
#
# このメソッドは、PMF を使用するデータサービスを停止する。サービス
# が動作していない場合、このメソッドは状態0 で終了する。その他の値
# は戻さない。リソースは STOP_FAILED 状態になる。

#pragma ident "@(#)dns_svc_stop 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソースの名前。
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されているリソース
                # グループの名前。
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプの名前。
                RESOURCETYPE_NAME=$OPTARG
                ;;
            *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [ $RESOURCETYPE_NAME, $RESOURCEGROUP_NAME, $RESOURCE_NAME ] \
```

## 例B-3 dns\_svc\_stopメソッド (続き)

```

        "ERROR: Option $OPTARG unknown"
        exit 1
        ;;
    esac
done

}

#####
# MAIN
#
#####

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# RTR ファイルから Stop_timeout 値を取得する。
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME`

# PMF 経由で SIGTERM シグナルを使用する規則正しい方法でデータサービ
# スを停止しようとする。SIGTERM がデータサービスを停止できるまで、
# Stop_timeout 値の 80% だけ待つ。停止できない場合、SIGKILL を送信
# して、データサービスを停止しようとする。SIGKILL がデータサービス
# を停止できるまで、Stop_timeout 値の 15% だけ待つ。停止できない場合、
# メソッドは何か異常があったと判断し、0 以外の状態で終了する。
# Stop_timeout の残りの 5% はほかの目的のために予約されている。
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

# in.named が動作しているかどうかを調べて、動作していれば停止する。
if pmfadm -q $PMF_TAG.named; then
    # シグナルをデータサービスに送信して、合計タイムアウト値
    # の80% だけ待つ。
    pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
    if [ $? -ne 0 ]; then
        logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \

```

## 例 B-3 dns\_svc\_stop メソッド (続き)

```

    "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
    SIGKILL"

# SIGTERM シグナルでデータサービスが停止しないので、今度は
# SIGKILL を使用して、合計タイムアウト値の 15% だけ待つ。
pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
if [ $? -ne 0 ]; then
    logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG] \
    "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESSFUL"

    exit 1
fi

else
# この時点でデータサービスは動作していない。メッセージを記録して、
# 成功で終了する。
logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
    "HA-DNS is not started"

# HA-DNS が動作していない場合でも、成功で終了し、データサービス
# リソースが STOP_FAILED 状態にならないようにする。
exit 0
fi

# DNS の停止に成功。メッセージを記録して、成功で終了する。
logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0

```

## gettime ユーティリティのコードリスト

gettime ユーティリティは、検証の再起動間の経過時間を PROBE プログラムが追跡するための C プログラムです。このプログラムは、コンパイル後、コールバックメソッドと同じディレクトリ (RT\_basedir プロパティが指すディレクトリ) に格納する必要があります。

## 例 B-4 gettime.c ユーティリティプログラム

```

# このユーティリティプログラムは、データサービスの検証メソッドによ
# って使用され、既知の参照ポイント (基準点) からの経過時間 (秒) を
# 追跡する。このプログラムは、コンパイル後、データサービスのコール
# バックメソッドと同じディレクトリ (RT_basedir) に格納しておくこと。

```

```

#pragma ident "@(#)gettime.c 1.1 00/05/24 SMI"

```

例 B-4 gettime.c ユーティリティープログラム (続き)

```
#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
{
    printf("%d\n", time(0));
    exit(0);
}
```

## PROBE プログラムのコードリスト

PROBE プログラムは、nslookup コマンドを使用して、データサービスの可用性を検査します (nslookup(1M) のマニュアルページを参照)。このプログラムは Monitor\_start コールバックメソッドによって起動され、Monitor\_stop コールバックメソッドによって停止されます。

例 B-5 dns\_probe プログラム

```
#!/bin/ksh
#pragma ident "@(#)dns_probe 1.1 00/04/19 SMI"
#
# HA-DNS の Probe メソッド
#
# このプログラムは、nslookup を使用して、データサービスの可用性を検査
# する。nslookup は DNS サーバーに照会することによって、DNS
# サーバー自身を探す。サーバーが応答しない場合、あるいは、別のサー
# バーが照会に応答した場合、probe メソッドはデータサービスまたはク
# ラスタ内の別のノードになんらかの問題が発生したという結論を下す。
# 検証は、RTR ファイルの THOROUGH_PROBE_INTERVAL で設定さ
# れた間隔で行われる。

#pragma ident "@(#)dns_probe 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
```

例 B-5 dns\_probe プログラム (続き)

```

R)
    # DNS リソースの名前。
    RESOURCE_NAME=$OPTARG
    ;;

G)
    # リソースが構成されているリソース
    # グループの名前。
    RESOURCEGROUP_NAME=$OPTARG
    ;;

T)
    # リソースタイプの名前。
    RESOURCETYPE_NAME=$OPTARG
    ;;

*)
    logger -p ${SYSLOG_FACILITY}.err \
        -t [ $RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME ] \
        "ERROR: Option $OPTARG unknown"
    exit 1
    ;;
esac
done
}

#####
# restart_service ()
#
# この関数は、まずデータサービスの Stop メソッドを呼び出し、
# 次に Start メソッドを呼び出すことによって、データサービスを再起動
# しようとする。データサービスがすでに起動しておらず、
# データサービスのタグが PMF に登録されていない場合、
# この関数はデータサービスをクラスタ内の
# 別のノードにフェイルオーバーする。
#
function restart_service
{
    # データサービスを再起動するには、まず、データサービス自身が
    # PMF 下に登録されているかどうかを確認する。
    pmfadm -q $PMF_TAG
    if [[ $? -eq 0 ]]; then
        # データサービスの TAG が PMF に登録されている場合、
        # データサービスを停止し、起動し直す。
        # 当該リソースの Stop メソッド名と STOP_TIMEOUT 値を取得する。
        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`
    fi
}

```

## 例B-5 dns\_probe プログラム (続き)

```

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

# 当該リソースの Start メソッド名と START_TIMEOUT 値を取得する。
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
    # データサービスの TAG が PMF に登録されていない場合、
    # データサービスが PMF 下で許可されている再試行最大回数を
    # 超えていることを示す。したがって、データサービスを再起動
    # してはならない。その代わりに、同じクラスタ内にある別のノード
    # にフェイルオーバーを試みる。
    scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
        -R $RESOURCE_NAME
fi

return 0
}

#####
# decide_restart_or_failover ()
#
# この関数は、検証が失敗したときに行うべきアクション、つまり、デー
# タサービスをローカルで再起動するか、クラスタ内の別のノードに
# フェイルオーバーするかを決定する。
#
function decide_restart_or_failover

```

## 例 B-5 dns\_probe プログラム (続き)

```
{  
  
# 最初の再起動の試行であるかどうかを検査する。  
if [ $retries -eq 0 ]; then  
    # 最初の失敗である。  
    # 最初の試行の時刻を記録する。  
    start_time=`$RT_BASEDIR/gettimè`  
    retries=`expr $retries + 1`  
    # 最初の失敗であるので、データサービスを  
    # 再起動しようと試行する。  
    restart_service  
    if [ $? -ne 0 ]; then  
        logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG] \  
            "${ARGV0} Failed to restart data service."  
        exit 1  
    fi  
else  
    # 最初の失敗ではない。  
    current_time=`$RT_BASEDIR/gettimè`  
    time_diff=`expr $current_time - $start_time`  
    if [ $time_diff -ge $RETRY_INTERVAL ]; then  
        # この失敗は再試行最大期間後に発生した。  
        # したがって、再試行カウンタをリセットし、  
        # 再試行時間をリセットし、さらに再試行する。  
        retries=1  
        start_time=$current_time  
        # 前回の失敗が Retry_interval よりも以前に発生しているので、  
        # データサービスを再起動しようと試行する。  
        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  
        # 再試行最大期間内であり、再試行カウンタは満了  
        # している。したがって、フェイルオーバーする。  
        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
```

## 例B-5 dns\_probe プログラム (続き)

```

# 再試行最大期間内であり、再試行カウンタは満了
# していない。したがって、さらに再試行する。
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
}

#####
# MAIN
#####

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# 証が行われる間隔はシステム定義プロパティー THOROUGH_PROBE_INTERVAL
# に設定されている。scha_resource_get でこのプロパティーの値を取得する。
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME

# 検証用のタイムアウト値を取得する。この値は RTR ファイルの
# PROBE_TIMEOUT 拡張プロパティーに設定されている。nslookup のデフォル
# トのタイムアウトは 1.5 分。
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}'`

# リソースの NETWORK_RESOURCES_USED プロパティーの値を取得して、
# DNS がサービスを提供するサーバーを見つける。
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME

# システム定義プロパティー Retry_count から再試行最大回数を取得する。

```

例 B-5 dns\_probe プログラム (続き)

```

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

# システム定義プロパティ Retry_interval から再試行最大期間を取得する。
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# リソースタイプの RT_basedir プロパティから gettime ユーティリティの
# 完全パスを取得する。
RT_BASEDIR=scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# 検証は無限ループで動作し、nslookup コマンドを実行し続ける。
# nslookup 応答用の一時ファイルを設定する。
DNSPROBEFILE=/tmp/.${RESOURCE_NAME}.probe
probfail=0
retries=0

while :
do
# 検証が動作すべき期間は <THOROUGH_PROBE_INTERVAL> プロパティに指
# 定されている。したがって、THOROUGH_PROBE_INTERVAL の間、検証
# プログラムが休眠するように設定する。
sleep $PROBE_INTERVAL

# DNS がサービスを提供しているIP アドレス上で nslookup コマンド
# を実行する。
hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
> $DNSPROBEFILE 2>&1

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

# nslookup への応答が HA-DNS サーバーから来ており、
# /etc/resolv.conf ファイル内に指定されているほかのネームサーバー
# から来ていないことを確認する。
if [ $probfail -eq 0 ]; then
# nslookup 照会に応答するサーバーの名前を取得する。
SERVER=`awk ' $1=="Server:" {print $2 }' \
$DNSPROBEFILE | awk -F. ' { print $1 } '`
if [ -z "$SERVER" ];
then
    probfail=1

```

## 例B-5 dns\_probe プログラム (続き)

```

        else
            if [ $SERVER != $DNS_HOST ]; then
                probefail=1
            fi
        fi
    fi

# probefail 変数が 0 以外である場合、nslookup コマンドがタイム
# アウトしたか、あるいは、別のサーバー (/etc/resolv.conf ファイ
# ルに指定されている) から照会への応答が来ていることを示す。
# どちらの場合でも、DNS サーバーは応答していないので、
# このメソッドは decide_restart_or_failover を呼び出して、
# データサービスをローカルで起動するか、あるいは、別のノードに
# フェイルオーバーするかを評価する。

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 メソッドのコードリスト

このメソッドは、データサービスの PROBE プログラムを起動します。

## 例B-6 dns\_monitor\_start メソッド

```

#!/bin/ksh
#
# Monitor start Method for HA-DNS.
#
# このメソッドは、PMF の制御下でデータサービスのモニター (検証) を
# 起動する。モニターは一定の間隔でデータサービスを検証するプロセス
# で、問題が発生すると、データサービスを同じノード上で再起動するか、
# クラスタ内の別のノードにフェイルオーバーする。モニター用の PMF
# タグは $RESOURCE_NAME.monitor。

#pragma ident "@(#)dns_monitor_start 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#

```

## 例 B-6 dns\_monitor\_start メソッド (続き)

```
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソースの名前。
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されているリソース
                # グループの名前
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプの名前。
                RESOURCETYPE_NAME=$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

# メッセージの記録に使用する syslog 機能を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=${RESOURCE_NAME}.monitor
SYSLOG_TAG=${RESOURCETYPE_NAME},${RESOURCEGROUP_NAME},${RESOURCE_NAME}
```

例B-6 dns\_monitor\_startメソッド (続き)

```
# データサービスの RT_BASEDIR プロパティを取得することによって、検
# 証メソッドが存在する場所を見つける。
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# PMF の制御下でデータサービスの検証を開始する。無限再試行オプション
# ンを使用して検証メソッドを起動する。リソースの名前、タイプ、および
# グループを検証メソッドに渡す。
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
    -T $RESOURCECETYPE_NAME

# HA-DNS のモニターが起動されたことを示すメッセージを記録する。
if [ $? -eq 0 ]; then
    logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
        "${ARGV0} Monitor for HA-DNS successfully started"
fi
exit 0
```

## Monitor\_stopメソッドのコードリスト

このメソッドは、データサービスの PROBE プログラムを停止します。

例B-7 dns\_monitor\_stopメソッド

```
#!/bin/ksh
# HA-DNS の Monitor_stop メソッド
# PMF を使用して動作しているモニターを停止する。

#pragma ident "@(#)dns_monitor_stop 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソースの名前。

```

## 例 B-7 dns\_monitor\_stop メソッド (続き)

```

        RESOURCE_NAME=$OPTARG
        ;;
    G)
        # リソースが構成されているリソース
        # グループの名前
        RESOURCEGROUP_NAME=$OPTARG
        ;;
    T)
        # リソースタイプの名前
        RESOURCETYPE_NAME=$OPTARG
        ;;
    *)
        logger -p ${SYSLOG_FACILITY}.err \
        -t [${RESOURCETYPE_NAME},${RESOURCEGROUP_NAME},${RESOURCE_NAME}] \
        "ERROR: Option $OPTARG unknown"
        exit 1
        ;;
done
done
}

#####
# MAIN
#
#####

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

# メッセージの記録に使用する syslog 機能を取得する。
SYSLOG_FACILITY=`scha_cluster_get -0 SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# モニターが動作しているかどうかを調べて、動作していれば停止する。
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

```

## 例B-7 dns\_monitor\_stop メソッド (続き)

```

# モニターは正常に停止している。メッセージを記録する。
logger -p ${SYSLOG_FACILITY}.info -t [${SYSLOG_TAG} \
    "${ARGV0} Monitor for resource " $RESOURCE_NAME \
    " successfully stopped"
fi
fi
exit 0

```

## Monitor\_check メソッドのコードリスト

このメソッドは、Confdir プロパティが示すディレクトリの存在を確認します。PROBE メソッドがデータサービスを新しいノードまたはゾーンにフェイルオーバーしたとき、RGM は Monitor\_check を呼び出します。また、潜在的なマスターとなっているノードまたはゾーンを検査する場合にも、RGM は同じ動作をします。

## 例B-8 dns\_monitor\_check メソッド

```

#!/bin/ksh#
# DNS の Monitor_check メソッド
#
# 障害モニターがデータサービスを新しいノードにフェイルオーバー
# するとき、RGM はこのメソッドを呼び出す。Monitor_check は
# Validate メソッドを呼び出して、新しいノード上で構成ディレク
# トリおよびファイルが利用できるかどうかを確認する。

#pragma ident "@(#)dns_monitor_check 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in

            R)
                # DNS リソースの名前。
                RESOURCE_NAME=$OPTARG
                ;;

            G)
                # リソースが構成されているリソース

```

## 例 B-8 dns\_monitor\_check メソッド (続き)

```
# グループの名前。
RESOURCEGROUP_NAME=$OPTARG
;;

T)
# リソースタイプの名前。
RESOURCE_TYPE_NAME=$OPTARG
;;

*)
logger -p ${SYSLOG_FACILITY}.err \
-t [$RESOURCE_TYPE_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

# メッセージの記録に使用する syslog 機能を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# リソースタイプの RT_BASEDIR プロパティから validate メソッドの
# 完全パスを取得する。
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# 当該リソースの validate メソッド名を取得する。
VALIDATE_METHOD=`scha_resource_get -O VALIDATE -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# データサービスを起動するための Confdir プロパティの値を取得する。
# 入力されたリソース名とリソースグループを使用して、リソースを
```

例 B-8 dns\_monitor\_check メソッド (続き)

```
# 追加するときに設定した Confdir の値を取得する。
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get は、拡張プロパティの値とともにタイプも戻す。
# awk を使用して、拡張プロパティの値だけを取得する。
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Validate メソッドを呼び出して、データサービスを新しいノードに
# フェイルオーバーできるかどうかを確認する。
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
-T $RESOURCECTYPE_NAME -x Confdir=$CONFIG_DIR

# モニター検査が成功したことを示すメッセージを記録する。
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 メソッドのコードリスト

このメソッドは、Confdir プロパティが示すディレクトリの存在を確認します。RGMがこのメソッドを呼び出すのは、データサービスが作成されたときと、クラスタ管理者がデータサービスのプロパティを更新したときです。障害モニターがデータサービスを新しいノードまたはゾーンにフェイルオーバーしたときは、Monitor\_check メソッドは常にこのメソッドを呼び出します。

例 B-9 dns\_validate メソッド

```
#!/bin/ksh
# HA-DNS の Validate メソッド
# このメソッドは、リソースの Confdir プロパティを妥当性検査する。
# Validate メソッドが呼び出されるのは、リソースが作成されたときと、リソース
# プロパティが更新されたときの 2 つである。リソースが作成されたとき、
# Validate メソッドは -c フラグで呼び出され、すべてのシステム定義プ
# ロパティと拡張プロパティがコマンド行引数として渡される。リソースプロ
# パティが更新されたとき、Validate メソッドは -u フラグで呼び出され、
# 更新されるプロパティのプロパティ / 値のペアだけがコマンド行引数とし
# て渡される。
```

## 例 B-9 dns\_validate メソッド (続き)

```
#
# 例: リソースが作成されたとき、コマンド行引数は次のようになる。
#
# dns_validate -c -R <.> -G <.> -T <.> -r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# 例: リソースプロパティが更新されたとき、コマンド行引数は次のようになる。
#
# dns_validate -u -R <.> -G <.> -T <.> -r <sys-prop_being_updated=value>
#   または
# dns_validate -u -R <.> -G <.> -T <.> -x <extn-prop_being_updated=value>

#pragma ident    "@(#)dns_validate    1.1    00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'cur:x:g:R:T:G:' opt
    do
        case "$opt" in
            R)
                # DNS リソースの名前
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されているリソース
                # グループの名前
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプの名前
                RESOURCETYPE_NAME=$OPTARG
                ;;
            r)
                #メソッドはシステム定義プロパティにアクセスして
                #いない。したがって、このフラグは動作なし。
                ;;
            g)
                # メソッドはリソースグループプロパティにアクセスして
                # いない。したがって、このフラグは動作なし。
                ;;
            c)

```

## 例 B-9 dns\_validate メソッド (続き)

```

# Validate メソッドがリソースの作成中に呼び出されてい
# ることを示す。したがって、このフラグは動作なし。
;;

u)
# リソースがすでに存在しているときは、プロパティの更新
# を示す。Confdir プロパティを更新する場合、Confdir
# がコマンド行引数に現れるはずである。現れない場合、
# メソッドは scha_resource_get を使用して
# Confdir を探す必要がある。
UPDATE_PROPERTY=1
;;

x)
# 拡張プロパティのリスト。プロパティと値のペア。
# 区切り文字は「=」。
PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
VAL=`echo $OPTARG | awk -F= '{print $2}'`

# Confdir 拡張プロパティがコマンド行上に存在する場合、
# その値を記録する。
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

# メッセージの記録に使用する syslog 機能を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# CONFDIR の値を NULL に設定する。この後、このメソッドは Confdir

```

## 例 B-9 dns\_validate メソッド (続き)

```
# プロパティの値を、コマンド行から取得するか、scha_resource_get を
# 使って取得する。
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

# プロパティの更新の結果として呼び出されている場合、Validate メソッ
# ドはコマンド行から Confdir 拡張プロパティの値を取得する。そうでな
# い場合、scha_resource_get を使用して Confdir の値を取得する。
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

# Confdir プロパティが値を持っているかどうかを確認する。持っていない
# い場合、状態 1 (失敗) で終了する。
if [[ -z $CONFDIR ]]; then
    logger -p ${SYSLOG_FACILITY}.err \
        "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
    exit 1
fi

# 実際の Confdir プロパティ値の妥当性検査はここから始まる。

# $CONFDIR がアクセス可能であるかどうかを検査する。
if [ ! -d $CONFDIR ]; then
    logger -p ${SYSLOG_FACILITY}.err -t [${SYSLOG_TAG}] \
        "${ARGV0} Directory $CONFDIR missing or not mounted"
    exit 1
fi

# named.conf ファイルが Confdir ディレクトリ内に存在するかどうかを
# 検査する。
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

# Validate メソッドが成功したことを示すメッセージを記録する。
logger -p ${SYSLOG_FACILITY}.info -t [${SYSLOG_TAG}] \
    "${ARGV0} Validate method for resource "$RESOURCE_NAME \
```

例 B-9 dns\_validate メソッド (続き)

```
    " completed successfully"

exit 0
```

## Update メソッドのコードリスト

プロパティが変更された場合、RGM は Update メソッドを呼び出して、そのことを動作中のリソースに通知します。

例 B-10 dns\_update メソッド

```
#!/bin/ksh
# HA-DNS の Update メソッド
#実際のプロパティの更新は RGM が行う。更新の影響を受けるのは障害モ
#ニターだけである。したがって、このメソッドは障害モニターを再起動
#する必要がある。

#pragma ident "@(#)dns_update 1.1 00/05/24 SMI"

#####
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
    typeset opt

    while getopts 'R:G:T:' opt
    do
        case "$opt" in
            R)
                # DNS リソースの名前
                RESOURCE_NAME=$OPTARG
                ;;
            G)
                # リソースが構成されているリソース
                # グループの名前。
                RESOURCEGROUP_NAME=$OPTARG
                ;;
            T)
                # リソースタイプの名前
                RESOURCETYPE_NAME=$OPTARG
                ;;
            *)
                logger -p ${SYSLOG_FACILITY}.err \
```

## 例 B-10 dns\_update メソッド (続き)

```

        -t [$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
        "ERROR: Option $OPTARG unknown"
        exit 1
    ;;
done
}
#####
# MAIN
#####
export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# メッセージの記録に使用する syslog 機能を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCE_TYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# リソースの RT_BASEDIR プロパティを取得することによって、
# 検証メソッドが存在する場所を見つける。
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`

# Update メソッドが呼び出されると、RGM は更新されるプロパティの値を
# 更新する。このメソッドは、障害モニター (検証メソッド) が動作し
# ているかどうかを検査し、動作している場合は強制終了し、再起動
# する必要がある。
if pmfadm -q $PMF_TAG.monitor; then

# すでに動作している障害モニターを強制終了する。
    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
        # DNS の停止に成功。メッセージを記録する。
        logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
            "Monitor for HA-DNS successfully stopped"
    fi
# モニターを再起動する。
    pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCE_TYPE_NAME

```

## 例B-10 dns\_updateメソッド (続き)

```
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
```



# サンプル DSDL リソースタイプのコード例

---

この付録では、SUNW.xfnts リソースタイプの各メソッドの完全なコード例を示します。また、コールバックメソッドが呼び出すサブルーチンのコードを含む、xfntc.c のコード例を示します。サンプルのリソースタイプ SUNW.xfnts の詳細については、[第8章](#)を参照してください。

この付録の内容は次のとおりです。

- 333 ページの「xfnts.c File Listing」
- 347 ページの「xfnts\_monitor\_check メソッドのコードリスト」
- 348 ページの「xfnts\_monitor\_start メソッドのコードリスト」
- 349 ページの「xfnts\_monitor\_stop メソッドのコードリスト」
- 350 ページの「xfnts\_probe メソッドのコードリスト」
- 353 ページの「xfnts\_start メソッドのコードリスト」
- 354 ページの「xfnts\_stop メソッドのコードリスト」
- 356 ページの「xfnts\_update メソッドのコードリスト」
- 357 ページの「xfnts\_validate メソッドのコードリスト」

## xfnts.c File Listing

このファイルは、SUNW.xfnts メソッドが呼び出すサブルーチンを実装します。

例C-1 xfnts.c

```
/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * HA-XFS 用の一般的なユーティリティ
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
```

## 例C-1 xfnts.c (続き)

```
* 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
*
*/

#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
```

## 例C-1 xfnts.c (続き)

```

* time left from the probe_timeout.
*/
#define SVC_DISCONNECT_TIMEOUT_SECONDS 2

/*
* svc_validate():
*
* Do HA-XFS specific validation of the resource configuration.
*
* svc_validate will check for the following
* 1. Confdir_list extension property
* 2. fontserver.cfg file
* 3. xfs binary
* 4. port_list property
* 5. network resources
* 6. other extension properties
*
* If any of the above validation fails then, Return > 0 otherwise return 0 for
* success
*/

int
svc_validate(scds_handle_t scds_handle)
{
    char    xfnts_conf[SCDS_ARRAY_SIZE];
    scha_str_array_t *confdirs;
    scds_net_resource_list_t *snrlp;
    int rc;
    struct stat statbuf;
    scds_port_list_t *portlist;
    scha_err_t err;

    /*
     * Get the configuration directory for the XFS dataservice from the
     * confdir_list extension property.
     */
    confdirs = scds_get_ext_confdir_list(scds_handle);

    /* Return an error if there is no confdir_list extension property */
    if (confdirs == NULL || confdirs->array_cnt != 1) {
        scds_syslog(LOG_ERR,
            "Property Confdir_list is not set properly.");
        return (1); /* Validation failure */
    }
}

```

## 例C-1 xfnts.c (続き)

```
/*
 * 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
     */
    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,
```

## 例C-1 xfnts.c (続き)

```

        "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;
    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:

```

## 例C-1 xfnts.c (続き)

```
    scds_free_net_list(snrlp);
    scds_free_port_list(portlist);

    return (rc); /* return result of validation */
}

/*
 * svc_start():
 *
 * Start up the X font server
 * Return 0 on success, > 0 on failures.
 *
 * The XFS service will be started by running the command
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS will be started under PMF. XFS will be started as a single instance
 * service. The PMF tag for the data service will be of the form
 * <resourcegroupname,resourcename,instance_number.svc>. In case of XFS, since
 * there will be only one instance the instance_number in the tag will be 0.
 */

int
svc_start(scds_handle_t scds_handle)
{
    char    xfnts_conf[SCDS_ARRAY_SIZE];
    char    cmd[SCDS_ARRAY_SIZE];
    scha_str_array_t *confdirs;
    scds_port_list_t    *portlist;
    scha_err_t    err;

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

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

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

    /*
     * Construct the command to start HA-XFS.
     * NOTE: XFS daemon prints the following message while stopping the XFS
     * "/usr/openwin/bin/xfs notice: terminating"
     */
}
```

## 例C-1 xfnts.c (続き)

```

    * In order to suppress the daemon message,
    * the output is redirected to /dev/null.
    */
(void) sprintf(cmd,
    "/usr/openwin/bin/xfns -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.
 *
 * 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;

    /*

```

## 例C-1 xfnts.c (続き)

```
* 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);
    }
}
```

## 例C-1 xfnts.c (続き)

```

/*
 * 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) {

```

## 例C-1 xfnts.c (続き)

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

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

}

/*
 * This function starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as <RG-name,RS-name,instance_number.mon>. The restart option
 * of PMF is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

int
mon_start(scds_handle_t scds_handle)
{
    scha_err_t    err;

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

    /*
     * The probe xfnts_probe is assumed to be available in the same
     * subdirectory where the other callback methods for the RT are
     * installed. The last parameter to scds_pmf_start denotes the
     * child monitor level. Since we are starting the probe under PMF
     * we need to monitor the probe process only and hence we are using
     * a value of 0.
     */
    err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
        SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

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

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

## 例C-1 xfnts.c (続き)

```

    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)
{
    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.
 */

```

## 例C-1 xfnts.c (続き)

```
int
svc_probe(scds_handle_t scds_handle, char *hostname, int port, int
timeout)
{
    int rc;
    hrtime_t t1, t2;
    int sock;
    char testcmd[2048];
    int time_used, time_remaining;
    time_t connect_timeout;

    /*
     * probe the dataservice by doing a socket connection to the port
     * specified in the port_list property to the host that is
     * serving the XFS dataservice. If the XFS service which is configured
     * to listen on the specified port, replies to the connection, then
     * the probe is successful. Else we will wait for a time period set
     * 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
```

## 例C-1 xfnts.c (続き)

```

* 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);

```

## 例C-1 xfnts.c (続き)

```
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 (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 メソッドのコードリスト

このメソッドは、基本的なリソースタイプ構成が有効であることを確認します。

例C-2 xfnts\_monitor\_check.c

```
/*
 * Copyright (c) 1998-2006 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);
    }

    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 メソッドのコードリスト

このメソッドは、xfnts\_probe メソッドを起動します。

例C-3 xfnts\_monitor\_start.c

```
/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_start.c - HA-XFS のモニター起動メソッド
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * このメソッドは、HA-XFS リソース用の障害モニターを起動する。
 * そのためには、検証機能を PMF の制御下で起動する。PMF タグの形式は
 * <RG-name, RS-name.mon> である。PMF の再起動オプションを
 * 使用するが、無限に再起動しない。その代わりに、
 * interval/retry_time を RTR ファイルから取得する。
 */

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

    /* RGM から渡された引数を処理して、syslog を初期化する。 */
    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);

    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);

    /* monitor_start メソッドの結果を戻す。 */
    return (rc);
}
```

## xfnts\_monitor\_stopメソッドのコードリスト

このメソッドは、xfnts\_probeメソッドを停止します。

例C-4 xfnts\_monitor\_stop.c

```
/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_stop.c - HA-XFS のモニター停止メソッド
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * このメソッドは、HA-XFS リソース用の障害モニターを停止する。
 * この処理は PMF 経由で行われる。障害モニター用の
 * PMF タグの形式は <RG-name_RS-name.mon> である。
 */

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

    /* RGM から渡された引数を処理して、syslog を初期化する。 */
    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);

    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);

    /* Return the result of monitor stop method */
    return (rc);
}
```

## xfnts\_probe メソッドのコードリスト

xfnts\_probe メソッドは、アプリケーションの可用性を検査して、データサービスをフェイルオーバーするか、再起動するかを決定します。xfnts\_monitor\_start コールバックメソッドがこのプログラムを起動し、xfnts\_monitor\_stop コールバックメソッドがそれを停止します。

例C-5 xfnts\_probe.c

```
/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_probe.c - HA-XFS の検査
 */

#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():
 * sleep() を実行して、PMF アクションスクリプトが sleep () に割り込むのを
 * 待機する無限ループ。sleep() への割り込みが発生すると、HA-XFS 用の
 * 起動メソッドを呼び出して、再起動する。
 */

int
main(int argc, char *argv[])
{
    int          timeout;
    int          port, ip, probe_result;
    scds_handle_t scds_handle;

    hrtime_t     ht1, ht2;
    unsigned long dt;
```

## 例C-5 xfnts\_probe.c (続き)

```

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);
}

/* 当該リソースに利用できるIP アドレスを取得する。 */
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);
}

/* ネットワークリソースが存在しない場合、エラーを戻す。 */
if (netaddr == NULL || netaddr->num_netaddrs == 0) {
    scds_syslog(LOG_ERR,
        "No network address resource in resource group.");
    return (1);
}

/*
 * X プロパティーからタイムアウト値を設定する。つまり、
 * 当該リソース用に構成されたすべてのネットワークリソース間で
 * タイムアウト値を分割するのではなく、検証を行うたびに、
 * 各ネットワークリソースに設定されているタイムアウト値を取得することを意味する。
 */
timeout = scds_get_ext_probe_timeout(scds_handle);

for (;;) {

    /*
     * 連続する検証の間、thorough_probe_interval
     * の期間、スリープ状態になる。
     */
    (void) scds_fm_sleep(scds_handle,
        scds_get_rs_thorough_probe_interval(scds_handle));

    /*
     * 使用するすべての IP アドレスを検証する。以下をループで検証する。
     * 1. 使用するすべてのネットワークリソース
     * 2. 指定されたリソースのすべての IP アドレス
     * 検証するIP アドレスごとに、

```

## 例 C-5 xfnts\_probe.c (続き)

```
    * 障害履歴を計算する。
    */
probe_result = 0;
/*
 * すべてのリソースを繰り返し検証して、svc_probe() の
 * 呼び出しに使用する各 IP アドレスを取得する。
 */
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
    /*
     * 状態を監視するホスト名と
     * ポートを取得する。
     */
    hostname = netaddr->netaddrs[ip].hostname;
    port = netaddr->netaddrs[ip].port_proto.port;
    /*
     * HA-XFS がサポートするポートは 1 つだけなので、
     * ポート値はポートの配列の最初の
     * エントリから取得する。
     */
    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);

    /*
     * サービス検証履歴を更新し、
     * 必要に応じて、アクションを実行する。
     * 検証終了時間を取得する。
     */
    ht2 = gethrtime();

    /* ミリ秒に変換する。 */
    dt = (ulong_t)((ht2 - ht1) / 1e6);

    /*
     * 障害の履歴を計算し、
     * 必要に応じて、アクションを実行する。
     */
    (void) scds_fm_action(scds_handle,
        probe_result, (long)dt);
} /* ネットワークリソースごと */
} /* 検証を永続的に繰り返す。 */
}
```

## xfnts\_start メソッドのコードリスト

RGM は、データサービスリソースのあるリソースグループがクラスタノードまたはゾーン上でオンラインになると、そのノードまたはゾーン上で Start メソッドを実行します。また、リソースが有効になったときも、RGM は同じ動作をします。xfnts\_start メソッドは、そのノードまたは大域ゾーン上で xfs デーモンをアクティブにします。

例 C-6 xfnts\_start.c

```

/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_start.c - HA-XFS の起動メソッド
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * HA-XFS 用の起動メソッド。リソース設定に対していくつかの
 * 健全性検査を行なったあと、アクションスクリプトを使用して HA-XFS を
 * PMF の制御下で起動する。
 */

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

    /*
     * RGM から渡された引数を処理して、
     * syslog を初期化する。
     */

    if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
    {
        scds_syslog(LOG_ERR, "Failed to initialize the handle.");
        return (1);
    }

    /* 構成の妥当性を検査する。エラーがあれば戻る。 */
    rc = svc_validate(scds_handle);

```

## 例C-6 xfnts\_start.c (続き)

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

/* データサービスを起動する。失敗した場合、エラーで戻る。 */
rc = svc_start(scds_handle);
if (rc != 0) {
    goto finished;
}

/* サービスが完全に起動するまで待つ。 */
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:
/* 割り当てられた環境リソースを解放する。 */
scds_close(&scds_handle);

return (rc);
}
```

## xfnts\_stop メソッドのコードリスト

RGMは、HA-XFS リソースのあるリソースグループがクラスタノードまたはゾーン上でオフラインになると、そのノードまたはゾーン上でStopメソッドを実行します。また、リソースが無効になったときも、RGMは同じ動作をします。このメソッドは、そのノードまたは大域ゾーン上でxfsデーモンを停止します。

## 例C-7 xfnts\_stop.c

```
/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_stop.c - HA-XFS の停止メソッド
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * PMF を使用して HA-XFS プロセスを停止する。
 */

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

    /* RGM から渡された引数を処理して、syslog を初期化する。 */
    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);

    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);

    /* svc_stop メソッドの結果を戻す。 */
    return (rc);
}
```

## xfnts\_update メソッドのコードリスト

プロパティが変更された場合、RGM は Update メソッドを呼び出して、そのことを動作中のリソースに通知します。管理アクションがリソースまたはそのグループのプロパティの設定に成功したあとに、RGM は Update を実行します。

### 例 C-8 xfnts\_update.c

```
#pragma ident "@(#)xfnts_update.c 1.10      01/01/18 SMI"

/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_update.c - HA-XFS の更新メソッド
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * リソースのプロパティが更新された可能性がある。
 * このような更新可能なプロパティはすべて障害モニターに関連するもので
 * あるため、障害モニターを再起動する必要がある。
 */

int
main(int argc, char *argv[])
{
    scds_handle_t    scds_handle;
    scha_err_t      result;

    /* RGM から渡された引数を処理して、syslog を初期化する。 */
    if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
    {
        scds_syslog(LOG_ERR, "Failed to initialize the handle.");
        return (1);
    }

    /*
     * 障害モニターがすでに動作していることを検査し、
     * 動作している場合、障害モニターを停止および再起動する。
     * scds_pmf_restart_fm() への 2 番目のパラメータは、再起動する
     * 必要がある障害モニターのインスタンスを一意に識別する。
     */
}
```

## 例C-8 xfnts\_update.c (続き)

```

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.");
    /* scds_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);
    return (1);
}

scds_syslog(LOG_INFO,
    "Completed successfully.");

/* scds_initialize が割り当てたすべてのメモリーを解放する。 */
scds_close(&scds_handle);

return (0);
}

```

## xfnts\_validate メソッドのコードリスト

xfnts\_validate メソッドは、Confdir\_list プロパティーが示すディレクトリの存在を確認します。RGMがこのメソッドを呼び出すのは、クラスタ管理者がデータサービスを作成したときと、データサービスのプロパティーを更新したときです。障害モニターがデータサービスを新しいノードまたはゾーンにフェイルオーバーしたときは、Monitor\_check メソッドは常にこのメソッドを呼び出します。

## 例C-9 xfnts\_validate.c

```

/*
 * Copyright (c) 1998-2006 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_validate.c - HA-XFS の検証メソッド
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * プロパティーが正しく設定されていることを確認する。
 */

```

## 例 C-9 xfnts\_validate.c (続き)

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

    /* RGM から渡された引数を処理して、syslog を初期化する。 */
    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_initialize が割り当てたすべてのメモリーを解放する。 */
    scds_close(&scds_handle);

    /* 検証メソッドの結果を戻す。 */
    return (rc);
}
```

## 有効な RGM 名と値

---

この付録では、リソースグループマネージャー (RGM) の名前と値に指定できる文字の条件について説明します。

この付録の内容は次のとおりです。

- 359 ページの「有効な RGM 名」
- 361 ページの「RGM の値」

### 有効な RGM 名

RGM 名は、次のカテゴリに分類されます。

- リソースグループ名
- リソースタイプ名
- リソース名
- プロパティ名
- 列挙型リテラル名

### 命名規則 (リソースタイプ名を除く)

リソースタイプ名を除き、すべての名前は次の規則に従う必要があります。

- 名前は ASCII である。
- 名前の先頭は文字である。
- 名前に使用できる文字は、英字の大文字と小文字、数字、ハイフン (-)、下線 ( )。
- 名前に使用できる最大文字数は 255 である。

## リソースタイプ名の形式

リソースタイプの完全な名前の書式は、次のように、リソースタイプによって異なります。

- リソースタイプのリソースタイプ登録 (RTR) ファイルに `#$upgrade` 指令が含まれる場合、書式は次のようになります。

```
vendor-id.base-rt-name:rt-version
```

- リソースタイプの RTR ファイルに `#$upgrade` 指令が含まれない場合、書式は次のようになります。

```
vendor-id.base-rt-name
```

ピリオドは、*vendor-id* と *base-rt-name* を分離します。コロンは、*base-rt-name* と *rt-version* を分離します。

この書式における変数要素は次のようになります。

<i>vendor-id</i>	ベンダー ID 接頭辞を指定します。ベンダー ID 接頭辞は、RTR ファイル内の <code>Vendor_id</code> リソースタイププロパティの値です。リソースタイプを開発する場合、会社の略号など、ベンダーを一意に識別するベンダー ID 接頭辞を選択します。たとえば、Sun Microsystems, Inc. により開発されるリソースタイプのベンダー ID 接頭辞は <code>SUNW</code> です。
<i>base-rt-name</i>	ベースリソースタイプ名を指定します。ベースリソースタイプ名は、RTR ファイル内の <code>Resource_type</code> リソースタイププロパティの値です。
<i>rt-version</i>	バージョン接尾辞を指定します。バージョン接尾辞は、RTR ファイル内の <code>RT_version</code> リソースタイププロパティの値です。バージョン接尾辞は、RTR ファイルが <code>#\$upgrade</code> 指令を含む場合、完全なリソースタイプ名の部分だけを示します。 <code>#\$upgrade</code> 指令は、Sun Cluster 製品のリリース 3.1 から導入されました。

---

注 - ベースリソースタイプ名が1つのバージョンだけ登録されている場合、管理コマンドで完全な名前を使用する必要はありません。ベンダー ID 接頭辞、バージョン接尾辞、あるいはその両方は省略できます。

---

詳細は、253 ページの「資源タイプのプロパティ」を参照してください。

例 D-1 リソースタイプの完全な名前 (#\$supgrade ディレクティブが指定されている場合)

この例では、RTR ファイルで次のようなプロパティが設定されているリソースタイプの完全な名前を示します。

- Vendor\_id=SUNW
- Resource\_type=sample
- RT\_version=2.0

RTR ファイルによって定義される完全なリソースタイプ名は次のようになります。

```
SUNW.sample:2.0
```

例 D-2 リソースタイプの完全な名前 (#\$supgrade ディレクティブが指定されていない場合)

この例では、RTR ファイルで次のようなプロパティが設定されているリソースタイプの完全な名前を示します。

- Vendor\_id=SUNW
- Resource\_type=nfs

RTR ファイルによって定義される完全なリソースタイプ名は次のようになります。

```
SUNW.nfs
```

## RGMの値

RGMの値は、プロパティ値と記述値という2つのカテゴリに分類されます。どちらのカテゴリも規則は同じで、次のようになります。

- 値は ASCII であること。
- 値の最大長は 4M - 1 バイト (つまり、4,194,303 バイト) であること。
- 値に次の文字を含むことはできない。
  - NULL
  - 復帰改行
  - セミコロン (;)



# 非クラスタ対応のアプリケーションの要件

---

通常、非クラスタ対応のアプリケーションの高可用性(HA)を実現するには、特定の要件を満たす必要があります。このような要件のリストが、29ページの「アプリケーションの適合性の分析」に示されています。この付録では、それらの要件のうち、特定のものについて詳細に説明します。

アプリケーションの高可用性を実現するには、そのリソースをリソースグループで構成します。アプリケーションのデータは、高可用性のクラスタファイルシステムに格納されます。したがって、1つのサーバーが異常終了しても、正常に動作しているサーバーによりデータにアクセスできます。クラスタファイルシステムについては、『Sun Cluster の概念 (Solaris OS 版)』を参照してください。

ネットワーク上のクライアントがネットワークにアクセスする場合、論理ネットワーク IP アドレスは、データサービスリソースと同じリソースグループにある論理ホスト名リソースで構成されます。データサービスリソースとネットワークアドレスリソースは共にフェイルオーバーします。この場合、データサービスのネットワーククライアントは新しいホスト上のデータサービスリソースにアクセスします。

この付録の内容は次のとおりです。

- 364 ページの「多重ホストデータ」
- 365 ページの「ホスト名」
- 366 ページの「多重ホームホスト」
- 366 ページの「INADDR\_ANY へのバインドと特定の IP アドレスへのバインド」
- 367 ページの「クライアントの再試行」

## 多重ホストデータ

高可用性のクラスタファイルシステムのデバイスは多重ホスト化されているため、ある物理ホストがクラッシュしても、正常に動作している物理ホストの1つがデバイスにアクセスできます。アプリケーションの高可用性を実現するには、そのデータが高可用性であることが必要です。したがって、アプリケーションのデータは、複数のクラスタノードまたはゾーンからアクセス可能なファイルシステムに格納されている必要があります。Sun Cluster で高可用性にできるローカルファイルシステムには、UNIX File System (UFS)、Quick File System (QFS)、Veritas File System (VxFS)、および Solaris ZFS (Zettabyte File System) があります。

クラスタファイルシステムは、独立したものであるように作成されたデバイスグループにマウントされます。ユーザーは、あるデバイスグループをマウントされたクラスタファイルシステムとして使用し、別のデバイスグループをデータサービス (HA Oracle ソフトウェアなど) で使用する raw デバイスとして使用することもできます。

アプリケーションは、データファイルの位置を示すコマンド行スイッチまたは構成ファイルを持っていることがあります。アプリケーションが、固定されたパス名を使用する場合は、アプリケーションのコードを変更せずに、このパス名を、クラスタファイルシステム内のファイルを指すシンボリックリンクに変更できます。シンボリックリンクの使用法の詳細については、[364 ページの「多重ホストデータを配置するためのシンボリックリンクの使用」](#)を参照してください。

最悪の場合は、実際のデータの位置を示すための機構を提供するように、アプリケーションのソースコードを変更する必要があります。この機構は、追加のコマンド行引数を作成することにより実装できます。

Sun Cluster は、ボリュームマネージャーで構成されている UNIX UFS ファイルシステムと HA の raw デバイスの使用をサポートします。Sun Cluster ソフトウェアをインストールおよび構成するとき、クラスタ管理者は、どのディスクリソースを UFS ファイルシステムまたは raw デバイス用に使用するかを指定する必要があります。通常、raw デバイスを使用するのは、データベースサーバーとマルチメディアサーバーだけです。

## 多重ホストデータを配置するためのシンボリックリンクの使用

場合によっては、アプリケーションのデータファイルのパス名が固定されており、しかも、固定されたパス名を変更する機構がないものがあります。このような場合に、シンボリックリンクを使用すればアプリケーションのコードを変更せずに、済ませられる場合もあります。

たとえば、アプリケーションがそのデータファイルに固定されたパス名 `/etc/mydatafile` を指定すると仮定します。このパスは、論理ホストのファイルシステムの1つにあるファイルを示す値を持つシンボリックリンクに変更できます。たとえば、パスを `/global/phys-schost-2/mydatafile` へのシンボリックリンクにできます。

ただし、データファイルの名前を内容とともに変更するアプリケーション(または、その管理手順)の場合には、シンボリックリンクをこのように使用すると問題が生じる可能性があります。たとえば、まず新しい一時ファイル `/etc/mydatafile.new` を作成することで、アプリケーションが更新を実行するとします。次に、このアプリケーションは `rename()` システムコール(または `mv` コマンド)を使用して、この一時ファイルの名前を実際のファイルの名前に変更します。一時ファイルを作成し、その名前を実際のファイル名に変更することで、データサービスは、そのデータファイルの内容が常に適切であるようにします。

このとき、`rename()` アクションはこのシンボリックリンクを破壊します。このため、`/etc/mydatafile` という名前は通常ファイルとなり、クラスタのクラスタファイルシステムの中ではなく、`/etc` ディレクトリと同じファイルシステムの中に存在することになります。`/etc` ファイルシステムは各ホスト専用であるため、フェイルオーバーまたはスイッチオーバー後はデータが利用できなくなります。

根本的な問題点は、既存のアプリケーションはシンボリックリンクを認識せず、またシンボリックリンクを処理するようには作成されていないことにあります。シンボリックリンクを使用し、データアクセスを論理ホストのファイルシステムにリダイレクトするには、アプリケーション実装がシンボリックリンクを消去しないように動作する必要があります。したがって、シンボリックリンクは、クラスタのファイルシステムへのデータ配置に関する問題の完全な解決策ではありません。

## ホスト名

データサービス開発者は、データサービスが動作しているサーバーのホスト名を、データサービスが知る必要があるかどうかを判断する必要があります。知る必要があると判断した場合、物理ホスト名ではなく、論理ホスト名を使用するようデータサービスを変更する必要があります。この意味で、論理ホスト名とは、アプリケーションリソースと同じリソースグループ内にある論理ホスト名リソース内に構成されているホスト名です。

データサービスのクライアントサーバープロトコルでは、サーバーが自分のホスト名をクライアントへのメッセージの一部としてクライアントに戻すことがあります。このようなプロトコルでは、クライアントは戻されたホスト名をサーバーに接続するときのホスト名として使用できます。戻されたホスト名をフェイルオーバーやスイッチオーバーが発生した後も使用できるようにするには、物理ホストではなく、リソースグループの論理ホスト名を使用する必要があります。物理ホスト名を使用している場合は、論理ホスト名をクライアントに戻すようにデータサービスのコードを変更する必要があります。

## 多重ホームホスト

「多重ホームホスト」とは、複数のパブリックネットワーク上にあるホストのことです。このようなホストは複数（つまり、ネットワークごとに1つ）のホスト名/IPアドレスのペアを持ちます。Sun Cluster は、1つのホストが複数のネットワーク上に存在できるように設計されています。1つのホストが単一のネットワーク上に存在することも可能ですが、このような場合は「多重ホームホスト」とは呼びません。物理ホスト名が複数のホスト名/IPアドレスのペアを持つように、各リソースグループも複数（つまり、パブリックネットワークごとに1つ）のホスト名/IPアドレスのペアを持つことができます。Sun Cluster がリソースグループをある物理ホストから別の物理ホストに移動するとき、そのリソースグループに対するホスト名/IPアドレスのペアの完全なセットもすべて移動します。

リソースグループに対するホスト名/IPアドレスのペアのセットは、リソースグループに含まれる論理ホスト名リソースとして構成されます。このようなネットワークアドレスリソースは、クラスタ管理者がリソースグループを作成および構成するときに指定します。Sun Cluster データサービス API は、このようなホスト名/IPアドレスのペアを照会する機能を持っています。

Solaris オペレーティングシステム用に書かれているほとんどの市販のデータサービスデーモンは、多重ホームホストを適切に処理できます。ネットワーク通信を行うとき、多くのデータサービスは Solaris のワイルドカードアドレス `INADDR_ANY` にバインドします。すると、`INADDR_ANY` は、すべてのネットワークインタフェースのすべての IP アドレスを自動的に処理します。`INADDR_ANY` は、現在マシンに構成されているすべての IP アドレスに効率的にバインドします。一般的に、`INADDR_ANY` を使用するデータサービスデーモンは、変更しなくても、Sun Cluster 論理ネットワークアドレスを処理できます。

## INADDR\_ANY へのバインドと特定の IP アドレスへのバインド

Sun Cluster の論理ネットワークアドレスの概念では、多重ホーム化されていない環境でも、マシンは複数の IP アドレスを持つことができます。そのマシンは、独自の物理ホストの IP アドレスを1つだけ持ち、さらに、現在マスターしているネットワークアドレス（論理ホスト名）リソースごとに追加の IP アドレスを持ちます。ネットワークアドレスリソースのマスターになるとき、マシンは動的に追加の IP アドレスを獲得します。ネットワークアドレスリソースのマスターを終了するとき、マシンは動的に IP アドレスを放棄します。

データサービスの中には、`INADDR_ANY` にバインドしていると、Sun Cluster 環境で適切に動作しないもあります。このようなデータサービスは、リソースグループのマスターになるとき、またマスターをやめるときに、バインドしている IP アドレスのセットを動的に変更する必要があります。このようなデータサービスが再バインド

する方法の1つが、起動メソッドと停止メソッドを使用し、データサービスのデーモンを強制終了および再起動するという方法です。

`Network_resources_used` リソースプロパティを使用すると、エンドユーザーは、アプリケーションリソースをバインドすべきネットワークアドレスリソースを構成できます。この機能が必要なリソースタイプの場合、そのリソースタイプのRTRファイルで`Network_resources_used` プロパティを宣言する必要があります。

リソースグループをオンラインまたはオフラインにするとき、RGMは、データサービスリソースメソッドを呼び出す順番に従って、ネットワークアドレスの取り付け(plumb)、取り外し(unplumb)、「起動」または「停止」への構成を行います。47ページの「[Start および Stop メソッドを使用するかどうかの決定](#)」を参照してください。

データサービスは、データサービスのStopメソッドが戻るまでに、リソースグループのネットワークアドレスを使用して、終了している必要があります。同様に、データサービスは、Startメソッドが戻るまでに、リソースグループのネットワークアドレスの使用を開始している必要があります。

データサービスが、個々のIPアドレスではなく、`INADDR_ANY` にバインドする場合、データサービスリソースメソッドが呼び出される順番とネットワークアドレスメソッドが呼び出される順番には重要な関係があります。

データサービスの停止メソッドと起動メソッドが、データサービスのデーモンを終了および再起動することで作業を実行する場合、データサービスは適切な時間にネットワークアドレスの使用を停止および開始します。

## クライアントの再試行

ネットワーククライアントから見ると、フェイルオーバーやスイッチオーバーは、論理ホストに障害が発生し、高速再起動しているように見えます。したがって、クライアントアプリケーションとクライアントサーバプロトコルは、このような場合に何回か再試行するように構成されていることが理想的です。アプリケーションとプロトコルがすでに単一サーバーのクラッシュと再起動に対応できている場合、リソースグループのテイクオーバーまたはスイッチオーバーにも対応できます。無限に再試行するようなアプリケーションもあります。また、何回も再試行していることをユーザーに通知し、さらに継続するかどうかをユーザーにたずねるような、より洗練されたアプリケーションもあります。



## CRNP のドキュメントタイプ定義

---

この付録では、CRNP (Cluster Reconfiguration Notification Protocol) 用の以下のドキュメントタイプ定義 (DTD) を挙げます。

- 369 ページの「SC\_CALLBACK\_REG XML DTD」
- 371 ページの「NVPAIR XML DTD」
- 372 ページの「SC\_REPLY XML DTD」
- 373 ページの「SC\_EVENT XML DTD」

### SC\_CALLBACK\_REG XML DTD

---

注 - SC\_CALLBACK\_REG と SC\_EVENT の両方で使用される NVPAIR データ構造は、一度だけ定義されます。

---

<!-- SC\_CALLBACK\_REG XML format specification

Copyright 2001-2006 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 2001-2006 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)

-&gt;

&lt;!ELEMENT NVP AIR (NAME,VALUE+)&gt;

&lt;!-- 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.

-&gt;

&lt;!ELEMENT NAME (#PCDATA)&gt;

&lt;!-- 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.

-&gt;

&lt;!ELEMENT VALUE (#PCDATA)&gt;

## SC\_REPLYXMLDTD

&lt;!-- SC\_REPLY XML format specification

Copyright 2001-2006 Sun Microsystems, Inc. All rights reserved.  
Use is subject to license terms.

-&gt;

&lt;!-- SC\_REPLY definition

The root element of the XML document represents a reply to a message. The reply contains a status code and a status message.

## ATTRIBUTES:

VERSION: The CRNP protocol version of the message.  
STATUS\_CODE: The return code for the message. One of the following: OK, RETRY, LOW\_RESOURCES, SYSTEM\_ERROR, FAIL, MALFORMED, INVALID\_XML, VERSION\_TOO\_HIGH, or VERSION\_TOO\_LOW.

## CONTENTS:

```

                SUBELEMENTS: SC_STATUS_MSG(1)
->
<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
    VERSION                NMTOKEN                #FIXED    "1.0"
    STATUS_CODE            OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>
<!-- SC_STATUS_MSG definition
    The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
    code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.

    ATTRIBUTES:

    CONTENTS:
        Arbitrary string.
-->
<!ELEMENT SC_STATUS_MSG (#PCDATA)>

```

## SC\_EVENT XML DTD

---

注 - SC\_CALLBACK\_REG と SC\_EVENT の両方で使用される NVPAIR データ構造は、一度だけ定義されます。

---

```

<!-- SC_EVENT XML format specification

    Copyright 2001-2006 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
>
```

## CrnpClient.java アプリケーション

---

この付録では、CrnpClient.java アプリケーションの完全なコードを示します (詳細は第12章を参照)。

### CrnpClient.java のコンテンツ

```
/*
 * CrnpClient.java
 * =====
 *
 * 解析についての注意:
 *
 * このプログラムは、Sun Java Architecture for XML Processing (JAXP) API を
 * 使用しています。API ドキュメントや利用についての情報は、
 * http://java.sun.com/xml/jaxp/index.html を参照してください。
 *
 * このプログラムは、Java 1.3.1 以降を対象に作成されています。
 *
 * プログラムの概要:
 *
 * このプログラムのメインスレッドは、CrnpClient オブジェクトを作成し、
 * ユーザーがデモを終了するのを待機し、CrnpClient オブジェクトで
 * shutdown を呼び出し、最後にプログラムを終了します。
 *
 * CrnpClient コンストラクタは、EventReceptionThread オブジェクトを作成し、
 * (コマンド行で指定されたホストとポートを使用して) CRNP サーバーに対して
 * 接続を開き、(コマンド行の指定にもとづいて) 登録メッセージを作成し、登録
 * メッセージを送信し、応答の読み取りと解析を行います。
 *
 * EventReceptionThread は、このプログラムが動作するマシンのホスト名と
 * コマンド行に指定されるポートにバインドされる待機ソケットを作成します。
 * EventReceptionThread は、イベントコールバックの着信を待機し、受信した
 * ソケットストリームから XML ドキュメントを構築し、これを CrnpClient
```

```
* オブジェクトに戻して処理を行わせませす。
*
* CrnpClient 内のshutdown メソッドは、単に登録解除用の
* (REMOVE_CLIENT) SC_CALLBACK_REG メッセージを crnp サーバーへ送信
* するだけです。
*
* エラー処理についての注意: 説明を簡潔にするため、このプログラムはほとんどの
* エラーに対して単に終了するだけですが、実際のアプリケーションではさまざまな
* 方法でエラー処理がなされます (適宜再試行するなど)。
*/

// JAXP パッケージ
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.*;

// 標準パッケージ
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * クラス CrnpClient
 * -----
 * 上記のファイルヘッダーコメントを参照。
 */
class CrnpClient
{
    /*
     * main
     * ----
     * 実行のエントリーポイント main は、コマンド行引数の数を
     * 検証し、すべての作業を行う CrnpClient インスタンスを
     * 作成する。
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

        /* コマンド行引数の数を検証する */
        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);
}

/*
 * コマンド行には crnp サーバーの IP とポート、
 * 待機するローカルポート、登録タイプを示す
 * 引数が入る。
 */
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 を作成する。
CrnpClient client = new CrnpClient(regIp, regPort, localPort,
    args);

// ユーザーがプログラムを終了したくなるまで待機する。
System.out.println("Hit return to terminate demo...");

// ユーザーが何か入力するまで読み取りはブロックする。
try {
    System.in.read();
} catch (IOException e) {
    System.out.println(e.toString());
}

// クライアントを停止する。
client.shutdown();
System.exit(0);
}

/*
 * =====
 * public メソッド
 * =====
 */

/*
 * CrnpClient コンストラクタ
 * -----

```

```
* crnp サーバーとの通信方法を知るためにコマンド行引数を解析
* し、イベント受信スレッドを作成し、このスレッドの実行を開始
* し、XML DocumentBuilderFactory オブジェクトを作成し、
* 最後に crnp サーバーにコールバックの登録を行う。
*/
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
        localPort = localPortIn;
        regs = clArgs;

        /*
         * xml 処理用のドキュメントビルダー
         * ファクトリを設定する。
         */
        setupXmlProcessing();

        /*
         * ServerSocket を作成してこれをローカル IP と
         * ポートにバインドする EventReception を作成する。
         */
        createEvtRecepThr();

        /*
         * crnp サーバーに登録する。
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * -----
 * CrnpClient (イベントコールバックを受信する際に
 * EventReceptionThread によって使用される) にコールバックする。
 */
public void processEvent(Event event)
{
    /*
     * ここでは、説明の都合上、単純にイベントを System.out
```

```

    *  へ出力。実際のアプリケーションでは、通常、イベント
    *  をなんらかの方法で使用する。
    */
    event.print(System.out);
}

/*
 *  shutdown
 *  -----
 *  CRNP サーバーに対する登録を解除する。
 */
public void shutdown()
{
    try {
        /* サーバーに登録解除メッセージを送信する */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}

/*
 *  =====
 *  private ヘルパーメソッド
 *  =====
 */

/*
 *  setupXmlProcessing
 *  -----
 *  xml 応答と xml イベントを解析するためにドキュメント
 *  ビルダーファクトリを作成する。
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // わざわざ検証する必要はない。
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // コメントと空白文字は無視したい。
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // CDATA セクションを TEXT ノードに結合する。

```

```
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -----
 * 新しい EventReceptionThread オブジェクトを作成し、待機
 * ソケットがバインドされる IP とポートを保存し、スレッドの
 * 実行を開始する。
 */
private void createEvtRecepThr() throws Exception
{
    /* スレッドオブジェクトを作成する */
    evtThr = new EventReceptionThread(this);

    /*
     * イベント配信コールバックを待機し始めるために
     * スレッドの実行を開始する。
     */
    evtThr.start();
}

/*
 * registerCallbacks
 * -----
 * crnp サーバーに対するソケット接続を作成し、
 * イベント登録メッセージを送信する。
 */
private void registerCallbacks() throws Exception
{
    System.out.println("About to register");

    /*
     * crnp サーバーの登録 IP / ポートに接続されたソケット
     * を作成し、登録情報を送信する。
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * 応答を読み取る。
     */
    readRegistrationReply(sock.getInputStream());

    /*
```

```
        * ソケット接続を閉じる。
        */
sock.close();
}

/*
 * unregister
 * -----
 * registerCallbacks の場合と同様に、crnp サーバーに対する
 * ソケット接続を作成し、登録解除メッセージを送信し、
 * サーバーからの応答を待機し、ソケットを閉じる。
 */
private void unregister() throws Exception
{
    System.out.println("About to unregister");

    /*
     * crnp サーバーの登録 IP / ポートに接続された
     * ソケットを作成し、登録解除情報を送信する。
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * 応答を読み取る。
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * ソケット接続を閉じる。
     */
    sock.close();
}

/*
 * createRegistrationString
 * -----
 * このプログラムのコマンド行引数にもとづいて CallbackReg
 * オブジェクトを作成し、CallbackReg オブジェクトから XML
 * 文字列を取得する。
 */
private String createRegistrationString() throws Exception
{
    /*
     * 実際のCallbackReg クラスを作成し、ポートを設定する。
     */
}
```

```
CallbackReg cbReg = new CallbackReg();
cbReg.setPort("" + localPort);

// 登録タイプを設定する
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);
}

// イベントを追加する。
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
 * -----
 * クラス EC_Cluster を使用して (サブクラスは使用しない)
 * XML 登録イベントを作成する。
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
```

```
* -----
* クラス EC_Cluster、サブクラス ESC_cluster_membership を
* 使用して XML 登録イベントを作成する。
*/
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * -----
 * クラス EC_Cluster、サブクラス ESC_cluster_rg_state、
 * および "rg_name" nvpair (入力パラメタにもとづく) を
 * 1 つ使用して XML 登録イベントを作成する。
 */
private Event createRgEvent(String rgname)
{
    /*
     * rgname リソースグループ用の
     * リソースグループ状態変更イベントを作成する。
     * このイベントタイプには、どのリソースグループに興味
     * があるのかを示すため、名前 / 値ペア (nvpair) を指定
     * する。
     */
    /*
     * イベントオブジェクトを作成し、クラスとサブクラスを設定する。
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * nvpair オブジェクトを作成し、これをこのイベントに追加する。
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
}

/*
 * createREvent
```

```
* -----
* クラス EC_Cluster、サブクラス ESC_cluster_rg_state、
* および "rg_name" nvpair (入力パラメタにもとづく) を
* 1 つ使用して XML 登録イベントを作成する。
*/
private Event createREvent(String rname)
{
    /*
    * rgname リソースグループ用の
    * リソースグループ状態変更イベントを作成する。
    * があるのかを示すため、名前 / 値ペア (nvpair) を指定
    * する。
    */
    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
* -----
* REMOVE_CLIENT CallbackReg オブジェクトを作成し、
* CallbackReg オブジェクトから XML 文字列を取得する。
*/
private String createUnregistrationString() throws Exception
{
    /*
    * CallbackReg オブジェクトを作成する。
    */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
    * 登録を OutputStream に整形化する。
    */
    String xmlStr = cbReg.convertToXml();

    // デバッグのために文字列を出力する。
    System.out.println(xmlStr);
    return (xmlStr);
}
```

```

}

/*
 * readRegistrationReply
 * -----
 * xml を解析してドキュメントにし、このドキュメントから
 * RegReply オブジェクトを構築し、RegReply オブジェクトを
 * 出力する。実際のアプリケーションでは、通常、RegReply
 * オブジェクトの status_code にもとづいて処理をする。
 */
private void readRegistrationReply(InputStream stream)
    throws Exception
{
    // ドキュメントビルダーを作成する。
    DocumentBuilder db = dbf.newDocumentBuilder();

    //
    // 解析前に ErrorHandler を設定する。
    // ここではデフォルトハンドラを使用。
    //
    db.setErrorHandler(new DefaultHandler());

    // 入力ファイルを解析する。
    Document doc = db.parse(stream);

    RegReply reply = new RegReply(doc);
    reply.print(System.out);
}

/* private 指定のメンバー変数 */
private InetAddress regIp;
private int regPort;
private EventReceptionThread evtThr;
private String regs[];

/* public 指定のメンバー変数 */
public int localPort;
public DocumentBuilderFactory dbf;
}

/*
 * クラス EventReceptionThread
 * -----
 * 上記のファイルヘッダーコメントを参照。
 */
class EventReceptionThread extends Thread
{

```

```
/*
 * EventReceptionThread コンストラクタ
 * -----
 * ローカルホスト名とワイルドカードポートにバインドされる
 * 新しい ServerSocket を作成する。
 */
public EventReceptionThread(CrnpClient clientIn) throws IOException
{
    /*
     * イベントの取得時に再度呼び返すことができるように、
     * クライアントに対する参照を保持する。
     */
    client = clientIn;

    /*
     * バインドする IP を指定する。これは、ローカル
     * ホスト IP である。このマシンに複数のパブリック
     * インタフェースが構成されている場合は、
     * InetAddress.getLocalHost によって検出される
     * ものをどれでも使用する。
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
    System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Thread.Start メソッドによって呼び出される。
 *
 * ServerSocket で着信接続を待機し、永続的にループする。
 *
 * 各着信接続が受け入れられる際に xml ストリームから
 * Event オブジェクトが作成される。続いてこのオブジェクト
 * が CrnpClient オブジェクトに返されて処理される。
 */
public void run()
{
    /*
     * Loop forever.
     */
    try {
        //
        // CrnpClient 内のドキュメントビルダーファクトリを
        // 使用してドキュメントビルダーを作成する。
        //
    }
}
```

```
DocumentBuilder db = client.dbf.newDocumentBuilder();

//
// 解析前に ErrorHandler を設定する。
// ここではデフォルトハンドラを使用。
//
db.setErrorHandler(new DefaultHandler());

while(true) {
    /* サーバーからのコールバックを待機 */
    Socket sock = listeningSock.accept();

    // 入力ファイルを解析する。
    Document doc = db.parse(sock.getInputStream());

    Event event = new Event(doc);
    client.processEvent(event);

    /* ソケットを閉じる */
    sock.close();
}
// 到達不能

} catch (Exception e) {
    System.out.println(e);
    System.exit(1);
}
}

/* private 指定のメンバー変数 */
private ServerSocket listeningSock;
private CrnpClient client;
}

/*
 * クラス NVPair
 * -----
 * このクラスは名前 / 値ペア (両方とも文字列) を格納する。
 * このクラスは、そのメンバーから NVPAIR XML メッセージを構築し、
 * NVPAIR XML 要素を解析してそのメンバーにすることができる。
 *
 * NVPAIR の形式仕様では複数の値が許可されているが、ここでは
 * 単純に値は 1 つだけという前提を下す。
 */
class NVPair
{
    /*
     * 2 つのコンストラクタ: 最初のコンストラクタは空の NVPair を
```

```
    * 作成する。2 つ目は NVPAIR XML 要素から NVPair を作成する。
    */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
    * Public 指定のセッター。
    */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }

    /*
    * 1 行で名前と値を出力する。
    */
    public void print(PrintStream out)
    {
        out.println("NAME=" + name + " VALUE=" + value);
    }

    /*
    * createXmlElement
    * -----
    * メンバー変数から NVPAIR XML 要素を作成する。
    * この要素を作成できるように、ドキュメントをパラメタとして
    * 受け付ける。
    */
    public Element createXmlElement(Document doc)
    {
        // 要素を作成する。
        Element nvpair = (Element)
            doc.createElement("NVPAIR");
        //
        // 名前を追加する。実際の名前は別の
        // CDATA セクションであることに注意。
    }
}
```

```
//
Element eName = doc.createElement("NAME");
Node nameData = doc.createCDATASection(name);
eName.appendChild(nameData);
nvpair.appendChild(eName);
//
// 値を追加する。実際の値は別の
// CDATA セクションであることに注意。
//
Element eValue = doc.createElement("VALUE");
Node valueData = doc.createCDATASection(value);
eValue.appendChild(valueData);
nvpair.appendChild(eValue);

return (nvpair);
}

/*
 * retrieveValues
 * -----
 * XML 要素を解析して名前と値を取得する。
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // NAME 要素を検出する。
    //
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }

    //
    // TEXT セクションを取得する。
    //
    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;
    }

    // 値を取得する。

```

```
name = n.getNodeValue();

//
// ここで値要素を取得する。
//
nl = elem.getElementsByTagName("VALUE");
if (nl.getLength() != 1) {
    System.out.println("Error in parsing: can't find "
        + "VALUE node.");
    return;
}

//
// TEXT セクションを取得する。
//
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;
}

// 値を取得する。
value = n.getNodeValue();
}

/*
 * Public 指定のアクセッサ
 */
public String getName()
{
    return (name);
}

public String getValue()
{
    return (value);
}

// Private 指定のメンバー変数
private String name, value;
}

/*
 * クラス Event
 * -----

```

```
* このクラスは、クラス、サブクラス、ベンダー、パブリッシャー、名前 /
* 値ペアのリストから成るイベントを格納する。このクラスでは、そのメンバー
* から SC_EVENT_REG XML 要素を作成し、この要素を解析してそのメンバーに
* することができる。次の非対称性に注意: SC_EVENT 要素を解析するが、
* 作成するのは SC_EVENT_REG 要素である。これは、SC_EVENT_REG 要素が
* 登録メッセージ (これは作成の必要がある) 内で使用され、SC_EVENT 要素
* がイベント配信 (これは解析の必要がある) 内で使用されるためである。
* 違いは、SC_EVENT_REG 要素にはベンダーとパブリッシャーがないことだけ
* である。
```

```
*/
class Event
{

    /*
    * 2 つのコンストラクタ: 最初のコンストラクタは空のイベントを
    * 作成し、2 つ目は SC_EVENT XML ドキュメントからイベントを
    * 作成する。
    */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // デバッグで使用できるようにドキュメントを文字列に
        // 変換して出力する。
        //
        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());

        // 実際に解析する。
        retrieveValues(doc);
    }
}
```

```
    }

    /*
     * Public 指定のセッター。
     */
    public void setClass(String classIn)
    {
        regClass = classIn;
    }

    public void setSubclass(String subclassIn)
    {
        regSubclass = subclassIn;
    }

    public void addNvpair(NVPair nvpair)
    {
        nvpairs.add(nvpair);
    }

    /*
     * createElement
     * -----
     * メンバー変数から SC_EVENT_REG XML 要素を作成する。
     * この要素を作成できるように、ドキュメントをパラメタとして
     * 受け付ける。NVPair createElement 機能を使用。
     */
    public Element createElement(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.createElement(doc));
        }
        return (event);
    }

    /*
     * メンバー変数を複数行に出力する。
     */
    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
 * -----
 * XML ドキュメントを解析し、クラス、サブクラス、ベンダー、
 * パブリッシャー、および nvpair を取得する。
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // SC_EVENT 要素を検出する。
    //
    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);

    //
    // CLASS、SUBCLASS、VENDOR、および PUBLISHER
    // 属性の値を取得する。
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // すべての nv ペアを取得する。
    //
    for (Node child = n.getFirstChild(); child != null;

```

```
        child = child.getNextSibling()
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 * Public 指定のアクセッサメソッド。
 */
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 指定のメンバー変数
private String regClass, regSubclass;
private Vector nvpairs;
private String vendor, publisher;
}

/*
 * クラス CallbackReg
 * -----
 * このクラスは、ポートと登録タイプ (どちらも文字列)、およびイベントリストを
 * 格納する。このクラスは、そのメンバーから SC_CALLBACK_REG XML メッセージ
 * を作成できる。
 *
 */
```

```
* SC_CALLBACK_REG メッセージを解析する必要があるのは CRNP サーバー
* だけであるため、このクラスで SC_CALLBACK_REG メッセージを解析でき
* なくてもよい。
*/
```

```
class CallbackReg
{
    // setRegType メソッドに便利な定義
    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 指定のセッター。
     */
    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
     * -----
     * メンバー変数から SC_CALLBACK_REG XML ドキュメントを構築する。
     * Event createXmlElement 機能を使用。
     */
    public String convertToXml()
    {
        Document document = null;
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.newDocument();
        } catch (ParserConfigurationException pce) {
            // 指定したオプションを持つパーサーを構築できない。
            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);

        //
        // ここでドキュメントを文字列に変換する。
        //
        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 指定のメンバー変数
private String port;
private String regType;
private Vector regEvents;
}

/*
 * クラス RegReply
 * -----
 * このクラスは、status_code と status_msg (どちらも文字列) を格納する。
 * このクラスは、SC_REPLY XML 要素を解析し、そのメンバーにできる。
 */
class RegReply
{
    /*
     * 1 つのコンストラクタが XML ドキュメントを受け入れて解析を行う。
     */
    public RegReply(Document doc)
    {
        //
        // ここでドキュメントを文字列に変換する。
        //
        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 指定のアクセッサ

```

```
    */
    public String getStatusCode()
    {
        return (statusCode);
    }

    public String getStatusMsg()
    {
        return (statusMsg);
    }

    /*
     * 1 行で情報を出力する。
     */
    public void print(PrintStream out)
    {
        out.println(statusCode + ": " +
            (statusMsg != null ? statusMsg : ""));
    }

    /*
     * retrieveValues
     * -----
     * XML ドキュメントを解析し、statusCode と statusMsg を取得する。
     */
    private void retrieveValues(Document doc)
    {
        Node n;
        NodeList nl;

        //
        // SC_REPLY 要素を検出する。
        //
        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);

        // STATUS_CODE 属性の値を取得する。
        statusCode = ((Element)n).getAttribute("STATUS_CODE");

        //
        // SC_STATUS_MSG 要素を検出する。
        //
```

```
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;
}

//
// TEXT セクションが存在する場合は、それを取得する。
//
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;
}

// 値を取得する。
statusMsg = n.getNodeValue();
}

// private 指定のメンバー変数。
private String statusCode;
private String statusMsg;
}
```



# 索引

---

## 数字・記号

- #\$upgrade\_from ディレクティブ, 81, 83
  - ANYTIME, 82
  - AT\_CREATION, 82
  - Tunable 属性の値, 81
  - WHEN\_DISABLED, 82
  - WHEN\_OFFLINE, 82
  - WHEN\_UNMANAGED, 82
  - WHEN\_UNMONITORED, 82
- #\$upgrade ディレクティブ, 81

## A

- Affinity\_timeout, リソースプロパティ, 264
- Agent Builder
  - Cluster Agent モジュール, 191
    - 違い, 196
  - rtconfig ファイル, 190
  - アプリケーションの分析, 169
  - インストール, 170
  - 既存のリソースタイプのクローン作成, 183
  - 起動, 171, 204
  - 構成, 170
    - 「構成」画面, 178
  - コードの再利用, 183
  - コマンド行バージョン, 185
    - 「作成」画面, 175
  - サポートファイル, 190
  - 出力, 209
  - 使用, 169
  - スクリプト, 188

## Agent Builder (続き)

- 生成されたソースコードの編集, 184
- 説明, 20, 26
- ソースファイル, 187
- で GDS を作成, 199
- ディレクトリ構造, 186
- ナビゲーション, 172
  - 「ファイル」メニュー, 174
  - 「ブラウザ」, 173
  - 「編集」メニュー, 175
  - メニュー, 174
- のコマンド行バージョンを使って GDS を使用するサービスを作成, 212
- バイナリファイル, 187
- パッケージディレクトリ, 190
- マニュアルページ, 188
- を使って、GDS を使用するサービスを作成, 204
- Agent Builder のインストール, 170
- ANYTIME, #upgrade\_from ディレクティブ, 82
- API\_version, リソースタイププロパティ, 254
- API、Resource Management, 「RMAPI」を参照
- Array\_maxsize, リソースプロパティ属性, 301
- Array\_minsize, リソースプロパティ属性, 301
- arraymax, リソースタイプのアップグレード, 80
- arraymin, リソースタイプのアップグレード, 80
- AT\_CREATION, #upgrade\_from ディレクティブ, 82
- Auto\_start\_on\_new\_cluster, リソースグループプロパティ, 286

**B**

- Boot, リソースタイププロパティ, 255
- Boot\_timeout, リソースプロパティ, 264
- Boot メソッド、使用, 51, 75

**C**

- Cheap\_probe\_interval, リソースプロパティ, 265
- clsetup, 説明, 27
- Cluster Reconfiguration Notification Protocol, 「CRNP」を参照
- Cluster Agent モジュール
  - Agent Builder との違い, 196
  - インストール, 191
  - 起動, 192
  - 使用, 194
  - 設定, 191
  - 説明, 191
- CRNP (Cluster Reconfiguration Notification Protocol)
  - Java アプリケーション例, 233
  - SC\_CALLBACK\_REG メッセージ, 225
  - SC\_EVENT, 229, 230
  - SC\_REPLY, 227, 228
  - エラー状況, 228
  - 概念, 221
  - 機能, 222
  - クライアント, 225
  - クライアント識別プロセス, 225
  - クライアントとサーバーの登録, 225
  - サーバー, 225
  - サーバーイベントの配信, 229
  - サーバーの応答, 227
  - 説明, 222
  - 通信, 223
  - 認証, 232
  - プロトコルのセマンティクス, 223
  - メッセージのタイプ, 223
- CRNP クライアントとサーバーの登録, 225
- C プログラム関数, RMAPI, 67

**D**

- Data Service Development Library (DSDL), コンポーネント, 25
- Default, リソースプロパティ属性, 301
- Description, リソースプロパティ属性, 301
- Desired primaries, リソースグループプロパティ, 286
- DSDL (Data Service Development Library)
  - 高可用性ローカルファイルシステムの有効化, 131
  - 障害監視, 218
  - 障害監視関数, 220
  - 障害モニターの実装, 129
  - 説明, 127, 128
  - データサービスの起動, 129
  - データサービスの停止, 129
  - ネットワークアドレスのアクセス, 130
  - ネットワークリソースアクセス関数, 217
  - 汎用関数, 215
  - リソースタイプ実装のサンプル
    - scds\_initialize() 関数, 148
    - SUNW.xfnts RTR ファイル, 147
    - SUNW.xfnts 障害モニター, 157
    - svc\_probe() 関数, 159
    - svc\_start() からの復帰, 150
    - TCP ポート番号, 146
    - xfnts\_monitor\_check メソッド, 156
    - xfnts\_monitor\_start メソッド, 154
    - xfnts\_monitor\_stop メソッド, 155
    - xfnts\_probe のメインループ, 158
    - xfnts\_start メソッド, 148
    - xfnts\_stop メソッド, 153
    - xfnts\_update メソッド, 166
    - xfnts\_validate メソッド, 163
    - X フォントサーバー, 145
    - X フォントサーバー構成ファイル, 146
    - サービスの起動, 149
    - サービスの検証, 148
    - 障害モニターのアクションの決定, 162
  - リソースタイプのデバッグ, 130
- DSDL (データサービス開発ライブラリ)
  - libdsdev.so, 20
  - Process Monitor Facility (PMF) 関数, 219
  - 概要, 20

**DSDL (データサービス開発ライブラリ) (続き)**

- 実装場所, 20
- プロパティー関数, 217
- ユーティリティー関数, 220
- DSDLによる高可用性ローカルファイルシステムの有効化, 131
- DSDLによるデータサービスの起動, 129
- DSDLによるデータサービスの停止, 129
- DSDLによるリソースタイプのデバッグ, 130

**E**

- Enumlist, リソースプロパティー属性, 301

**F**

- Failback, リソースグループプロパティー, 286
- Failover, リソースタイププロパティー, 255
- Failover\_mode, リソースプロパティー, 265
- Fini, リソースタイププロパティー, 256
- Fini\_timeout, リソースプロパティー, 269
- Fini メソッド、実装のためのガイドライン, 51
- Fini メソッド、使用, 50-51, 74

**G****GDS (汎用データサービス)**

- Sun Cluster 管理コマンドで使用, 199
- Agent Builder のコマンド行バージョンを使ってサービスを作成, 212
- Child\_mon\_level プロパティー, 201
- Failover\_enabled プロパティー, 201
- Log\_level プロパティー, 202
- Network\_resources\_used プロパティー, 202
- Port\_list プロパティー, 200
- Probe\_command プロパティー, 202
- Probe\_timeout プロパティー, 203
- Start\_command 拡張プロパティー, 200
- Start\_timeout プロパティー, 203
- Stop\_command プロパティー, 203
- Stop\_signal プロパティー, 204
- Stop\_timeout プロパティー, 204

**GDS (汎用データサービス) (続き)**

- Sun Cluster Agent Builder で使用, 199
- Sun Cluster Agent Builder を使って、GDS を使用するサービスを作成, 204
- SUNW.gds リソースタイプ, 198
- Validate\_command プロパティー, 204
- Validate\_timeout プロパティー, 204
- コマンドを使って、GDS を使用するサービスを作成, 210
- 使用する場合, 198
- 使用方法, 199
- 使用理由, 198
- 説明, 197
- 必須プロパティー, 200
- Global\_resources\_used, リソースグループプロパティー, 286
- Global\_zone, リソースタイププロパティー, 257

**H**

- halockrun, 説明, 55
- hatimerun, 説明, 55
- HA データサービス, 検証, 61

**I**

- Implicit\_network\_dependencies, リソースグループプロパティー, 287
- Init, リソースタイププロパティー, 258
- Init\_nodes, リソースタイププロパティー, 258
- Init\_timeout, リソースプロパティー, 269
- Init メソッド、使用, 50, 74
- Installed\_nodes, リソースタイププロパティー, 258
- Is\_logical\_hostname, リソースタイププロパティー, 258
- Is\_shared\_address, リソースタイププロパティー, 259

**J**

- Java, CRNP を使用するアプリケーション例, 233

**L**

libdsdev.so, DSDL, 20  
libscha.so, RMAPI, 20  
Load\_balancing\_policy, リソースプロパティ  
ー, 269  
Load\_balancing\_weights, リソースプロパティ  
ー, 269

**M**

max, リソースタイプのアップグレード, 80  
Max, リソースプロパティ属性, 301  
Maximum primaries, リソースグループプロパ  
ティ, 287  
Maxlength, リソースプロパティ属性, 301  
min, リソースタイプのアップグレード, 80  
Min, リソースプロパティ属性, 301  
Minlength, リソースプロパティ属性, 301  
Monitor\_check, リソースタイププロパティ, 259  
Monitor\_check\_timeout, リソースプロパティ  
ー, 270  
Monitor\_check メソッド  
互換性, 82  
使用, 77  
Monitor\_start, リソースタイププロパティ, 259  
Monitor\_start\_timeout, リソースプロパティ  
ー, 270  
Monitor\_start メソッド, 使用, 77  
Monitor\_stop, リソースタイププロパティ, 259  
Monitor\_stop\_timeout, リソースプロパティ  
ー, 270  
Monitor\_stop メソッド, 使用, 77  
Monitored\_switch, リソースプロパティ, 270

**N**

Network\_resources\_used, リソースプロパティ  
ー, 271  
Nodelist, リソースグループプロパティ, 287  
Num\_resource\_restarts, リソースプロパティ  
ー, 271  
Num\_rg\_restarts, リソースプロパティ, 272

**O**

On\_off\_switch, リソースプロパティ, 272

**P**

Pathprefix, リソースグループプロパティ, 287  
Per\_node, リソースプロパティ属性, 301  
Pingpong\_interval, リソースグループプロパ  
ティ, 288  
Pkglist, リソースタイププロパティ, 259  
PMF (Process Monitor Facility)  
overview, 20  
関数, DSDL, 219  
Port\_list, リソースプロパティ, 272  
Postnet\_start メソッド, 使用, 77  
Postnet\_stop  
互換性, 82  
リソースタイププロパティ, 260  
Postnet\_stop\_timeout, リソースプロパティ  
ー, 273  
Prenet\_start, リソースタイププロパティ, 260  
Prenet\_start\_timeout, リソースプロパティ  
ー, 273  
Prenet\_start メソッド, 使用, 77  
Process Monitor Facility, 「PMF」を参照  
Property, リソースプロパティ属性, 302  
Proxy, リソースタイププロパティ, 260

**R**

R\_description, リソースプロパティ, 273  
Resource\_dependencies, リソースプロパティ  
ー, 273  
Resource\_dependencies\_offline\_restart, リソース  
プロパティ, 275  
Resource\_dependencies\_restart, リソースプロパ  
ティ, 276  
Resource\_dependencies\_weak, リソースプロパ  
ティ, 278  
Resource Group Manager, 「RGM」を参照  
Resource Group Manager (RGM)  
管理インタフェース, 26  
説明, 23

## Resource Group Manager (RGM) (続き)

- 目的, 20

- リソースグループの処理, 21

- リソースタイプの処理, 21

- リソースの処理, 21

## Resource\_list

- リソースグループプロパティ, 288

- リソースタイププロパティ, 261

## Resource Management API, 「RMAPI」を参照

Resource\_name, リソースプロパティ, 279

Resource\_project\_name, リソースプロパティ, 279

Resource\_state, リソースプロパティ, 280

resource-type, アップグレード, 80

Resource\_type, リソースタイププロパティ, 261

Retry\_count, リソースプロパティ, 280

Retry\_interval, リソースプロパティ, 281

RG\_affinities, リソースグループプロパティ, 288

RG\_dependencies, リソースグループプロパティ, 289

RG\_description, リソースグループプロパティ, 290

RG\_is\_frozen, リソースグループプロパティ, 290

RG\_mode, リソースグループプロパティ, 290

RG\_name, リソースグループプロパティ, 291

RG\_project\_name, リソースグループプロパティ, 291

RG\_slm\_cpu, リソースグループプロパティ, 291

RG\_slm\_cpu\_min, リソースグループプロパティ, 292

RG\_slm\_pset\_type, リソースグループプロパティ, 294

RG\_slm\_type, リソースグループプロパティ, 293

RG\_state, リソースグループプロパティ, 297

RG\_system, リソースグループプロパティ, 299

## RGM (Resource Group Manager)

- 値, 361

- 有効な名前, 359

## RMAPI (Resource Management API), 20

- C プログラム関数, 67

- libscha.so, 20

- クラスタ関数, 70

## RMAPI (Resource Management API) (続き)

- クラスタコマンド, 67

- コールバックメソッド, 71

- コンポーネント, 25

- シェルコマンド, 65

- 実装場所, 20

- 終了コード, 72

- メソッド引数, 71

- ユーティリティ関数, 71

- リソース関数, 67

- リソースグループ関数, 69

- リソースグループコマンド, 66

- リソースコマンド, 66

- リソースタイプ関数, 68

- リソースタイプコマンド, 66

RT\_basedir, リソースタイププロパティ, 261

RT\_description, リソースタイププロパティ, 262

RT\_system, リソースタイププロパティ, 262  
rt-version, アップグレード, 80

## RT\_version

- 変更するとき, 83

- 目的, 83

- リソースタイププロパティ, 262

rtconfig ファイル, 190

## RTR (Resource Type Registration)

- ファイル

- SUNW.xfnts, 147

## RTR (リソースタイプ登録)

- ファイル

- 変更, 86

## RTR (リソースタイプ登録ファイル)

- ファイル

- 説明, 134

**S**

SC\_CALLBACK\_REG, 内容, 226-227

SC\_EVENT, 内容, 230

SC\_REPLY, 内容, 228

Scalable, リソースプロパティ, 281

scds\_initialize() 関数, 148

Single\_instance, リソースタイププロパティ, 262

Start, リソースタイププロパティ, 262  
Start\_timeout, リソースプロパティ, 282  
Start メソッド, 使用, 47, 73  
Status, リソースプロパティ, 282  
Status\_msg, リソースプロパティ, 282  
Stop, リソースタイププロパティ, 263  
Stop\_timeout, リソースプロパティ, 283  
Stop メソッド  
    互換性, 82  
    使用, 73  
Stopメソッド, 使用, 47  
Sun Cluster  
    GDSによる使用, 198  
    アプリケーション環境, 19  
    コマンド, 27  
Sun Cluster Agent Builder, 「Agent Builder」を参照  
Sun Cluster Manager, 説明, 26  
SunPlex Manager, 「Sun Cluster Manager」を参照  
SUNW.xfnts  
    RTR ファイル, 147  
    障害モニター, 157  
Suspend\_automatic\_recovery, リソースグループプロパティ, 299  
svc\_probe() 関数, 159

## T

TCP 接続, DSDL 障害監視の使用, 218  
Thorough\_probe\_interval, リソースプロパティ, 283  
Tunable, リソースプロパティ属性, 302  
Tunable 属性のオプション, 81  
    ANYTIME, 82  
    AT\_CREATION, 82  
    WHEN\_DISABLED, 82  
    WHEN\_OFFLINE, 82  
    WHEN\_UNMANAGED, 82  
    WHEN\_UNMONITORED, 82  
Tunable 属性の制約, 文書の要件, 88  
Type, リソースプロパティ, 283  
Type\_version, リソースプロパティ, 283

## U

UDP\_affinity, リソースプロパティ, 284  
Update, リソースタイププロパティ, 263  
Update\_timeout, リソースプロパティ, 284  
Update メソッド  
    互換性, 82  
    使用, 56, 76  
upgrade 指令, 360

## V

Validate, リソースタイププロパティ, 263  
Validate\_timeout, リソースプロパティ, 284  
Validate メソッド  
    使用, 56, 76  
vendor-id  
    アップグレード, 80  
    区別, 80  
Vendor\_ID, リソースタイププロパティ, 263

## W

Weak\_affinity, リソースプロパティ, 285  
WHEN\_DISABLED, #upgrade\_from ディレクティブ, 82  
WHEN\_OFFLINE, #upgrade\_from ディレクティブ, 82  
WHEN\_UNMANAGED, #upgrade\_from ディレクティブ, 82  
WHEN\_UNMONITORED, #upgrade\_from ディレクティブ, 82

## X

xfnts\_monitor\_check, 156  
xfnts\_monitor\_start, 154  
xfnts\_monitor\_stop, 155  
xfnts\_start, 148  
xfnts\_stop, 153  
xfnts\_update, 166  
xfnts\_validate, 163  
xfs サーバー, ポート番号, 146

X フォントサーバー  
構成ファイル, 146  
定義, 145

## あ 値

RGM (Resource Group Manager), 361  
デフォルトプロパティ, 84  
アップグレード, 文書の要件, 88-90  
アップグレード対応, 定義済み, 80  
アプリケーション環境, Sun Cluster, 19

## い

依存関係, リソース間の調節, 62  
イベント, 保証された配信, 230  
インストール要件, リソースタイプパッケージ, 85  
インタフェース  
Resource Group Manager (RGM), 26  
コマンド行, 27  
プログラミング, 25

## え

エラー状況, CRNP, 228

## お

オプション, Tunable 属性, 81

## か

概念, CRNP, 221  
拡張プロパティ  
宣言, 42  
リソースタイプ, 265  
リソースプロパティ属性, 301  
型, リソースプロパティの属性, 302

## 画面

「構成」, 178  
「作成」, 175

## 関数

DSDL Process Monitor Facility (PMF), 219  
DSDL 障害モニター, 220  
DSDL ネットワークリソースアクセス, 217  
DSDL プロパティ, 217  
DSDL ユーティリティ, 220  
RMAPI C プログラム, 67  
RMAPI クラスタ, 70  
RMAPI ユーティリティ, 71  
RMAPI リソース, 67  
RMAPI リソースグループ, 69  
RMAPI リソースタイプ, 68  
scds\_initialize(), 148  
svc\_probe(), 159  
汎用 DSDL, 215  
命名規則, 147  
完全修飾リソースタイプ名, 取得方法, 80  
管理インタフェース, Resource Group Manager (RGM), 26  
管理コマンド, を使って, GDS を使用するサービスを作成, 210

## き

キープアライブ, 使用, 60  
記述値, 規則, 361  
規則

関数名, 147  
記述値, 361  
コールバックメソッド名, 147  
プロパティ値, 361  
プロパティ名, 359  
リソースグループ名, 359  
リソース名, 359  
列挙型リテラル名, 359  
既存のリソースタイプのクローン作成, Agent Builder, 183

- く
  - クライアント, CRNP, 225
  - クラスタ関数, RMAPI, 70
  - クラスタコマンド, RMAPI, 67
- け
  - 検証
    - HA データサービス, 61
    - データサービス, 60
  - 検証チェック, スケーラブルサービス, 60
- こ
  - 構成, Agent Builder, 170
  - 「構成」画面, Agent Builder, 178
  - 構文
    - 記述値, 361
    - プロパティ値, 361
    - プロパティ名, 359
    - リソースグループ名, 359
    - リソースタイプ名, 360
    - リソース名, 359
    - 列挙型リテラル名, 359
  - コード
    - RMAPI 終了, 72
    - メソッドの変更, 86
    - モニターの変更, 86
  - コードの再利用, Agent Builder, 183
  - コールバックメソッド
    - Monitor\_check, 77
    - Monitor\_start, 77
    - Monitor\_stop, 77
    - Postnet\_start, 77
    - Prenet\_start, 77
    - RMAPI, 71
    - Update, 76
    - Validate, 76
    - 概要, 20
    - 使用, 56
    - 初期化, 72
    - 制御, 72
    - 説明, 24
  - コールバックメソッド (続き)
    - 命名規則, 147
  - コマンド
    - clsetup, 27
    - halockrun, 55
    - hatimerun, 55
    - RMAPI リソースタイプ, 66
    - Sun Cluster, 27
    - を使って, GDS を使用するサービスを作成, 210
  - コマンド行
    - Agent Builder, 185
    - コマンド, 27
  - コンポーネント, RMAPI, 25
- さ
  - サーバー
    - CRNP, 225
    - xfns
      - ポート番号, 146
    - X フォント
      - 構成ファイル, 146
      - 定義, 145
    - 「作成」画面, Agent Builder, 175
  - サポートファイル, Agent Builder, 190
  - サンプル, データサービス, 91
  - サンプル DSDL コード
    - scds\_initialize() 関数, 148
    - SUNW.xfnts RTR ファイル, 147
    - SUNW.xfnts 障害モニター, 157
    - svc\_probe() 関数, 159
    - svc\_start() からの復帰, 150
    - TCP ポート番号, 146
    - xfnts\_monitor\_check メソッド, 156
    - xfnts\_monitor\_start メソッド, 154
    - xfnts\_monitor\_stop メソッド, 155
    - xfnts\_probe のメインループ, 158
    - xfnts\_start メソッド, 148
    - xfnts\_stop メソッド, 153
    - xfnts\_update メソッド, 166
    - xfnts\_validate メソッド, 163
    - X フォントサーバー, 145
    - X フォントサーバー構成ファイル, 146

サンプル DSDL コード (続き)  
 サービスの起動, 149  
 サービスの検証, 148  
 障害モニターのアクションの決定, 162  
 サンプルデータサービス  
 Monitor\_check メソッド, 118  
 Monitor\_start メソッド, 116  
 Monitor\_stop メソッド, 117  
 RTR ファイル, 93  
 RTR ファイルの拡張プロパティ, 97  
 RTR ファイルのサンプルプロパティ, 95  
 Start メソッド, 103  
 Stop メソッド, 107  
 Update メソッド, 124  
 Validate メソッド, 119  
 エラーメッセージの生成, 102  
 共通の機能, 98  
 検証プログラム, 110  
 障害モニターの定義, 109  
 データサービスの制御, 103  
 プロパティ更新の処理, 119  
 プロパティ情報の取得, 102

## し

シェルコマンド, RMAPI, 65  
 実装  
 DSDL 障害モニター, 129  
 RMAPI, 20  
 リソースタイプ名, 84  
 リソースタイプモニター, 84  
 終了コード, RMAPI, 72  
 主ゾーン, 23  
 主ノード, 23  
 障害モニター  
 SUNW.xfnts, 157  
 関数, DSDL, 220  
 デーモン  
 設計, 142  
 書式, リソースタイプ名, 360  
 指令, #supgrade, 360

## す

スクリプト  
 Agent Builder, 188  
 構成, 207  
 作成, 204  
 スケーラブルサービス, 検証, 60  
 スケーラブルリソース, 実装, 57

## せ

生成された Agent Builder ソースコードの編集, 184

## そ

ソースコード, 生成された Agent Builder の編集, 183  
 ソースファイル, Agent Builder, 187  
 属性, リソースプロパティ, 301

## ち

チェック, スケーラブルサービスの検証, 60

## て

ディレクティブ  
 #supgrade, 81  
 #supgrade\_from, 81, 83  
 RT\_version, 81  
 RTR ファイル内の配置, 81  
 Tunable 属性の制約, 81  
 デフォルトの Tunable 属性, 82  
 ディレクトリ, Agent Builder, 190  
 ディレクトリ構造, Agent Builder, 186  
 データサービス  
 HA の検証, 61  
 開発環境の設定, 32  
 検証, 60  
 検証のためにクラスタに転送する, 34  
 作成, 60  
 インタフェースの決定, 31

## データサービス, 作成 (続き)

適合性の分析, 29

サンプル, 91

Monitor\_check メソッド, 118

Monitor\_start メソッド, 116

Monitor\_stop メソッド, 117

RTR ファイル, 93

RTR ファイルの拡張プロパティ, 97

RTR ファイルのリソースプロパティ, 95

Start メソッド, 103

Stop メソッド, 107

Update メソッド, 124

Validate メソッド, 119

エラーメッセージの生成, 102

共通の機能, 98

検証プログラム, 110

障害モニターの定義, 109

データサービスの制御, 103

プロパティ更新の処理, 119

プロパティ情報の取得, 102

データサービス開発ライブラリ, 「DSDL」を参照

データサービスの作成, 60

デーモン, 障害モニター的设计, 142

デフォルトのプロパティ値

Sun Cluster 3.0, 84

アップグレード用の新しい値, 84

継承される時, 84

## な

ナビゲーション, Agent Builder の, 172

## ね

ネットワークアドレスのアクセス, DSDL による, 130

ネットワークリソースアクセス関数, DSDL, 217

## は

バイナリファイル, Agent Builder, 187

パッケージディレクトリ, Agent Builder, 190

汎用データサービス

「GDS」を参照

汎用データサービス (GDS), 定義, 46

## ひ

引数, RMAPI メソッド, 71

## ふ

ファイル

Agent Builder におけるサポート, 190

Agent Builder のソース, 187

Agent Builder のバイナリ, 187

rtconfig, 190

フェイルオーバーリソース, 実装, 56

複数の登録バージョン間の区別, *rt-version*, 80

「ブラウズ」, Agent Builder, 173

プログラミングアーキテクチャー, 20

プログラミングインタフェース, 25

プロセス監視機能 (PMF), 目的, 55

プロセス管理, 55

プロパティ

Child\_mon\_level, 201

Failover\_enabled, 201

GDS, 必須, 201

Log\_level, 202

Network\_resources\_used, 202

Port\_list, 200

Probe\_command, 202

Probe\_timeout, 203

Start\_command 拡張, 200

Start\_timeout, 203

Stop\_command, 203

Stop\_signal, 204

Stop\_timeout, 204

Validate\_command, 204

Validate\_timeout, 204

拡張プロパティの宣言, 42

リソース, 264

リソースグループ, 285

リソースタイプ, 253

リソースタイプの設定, 34

## プロパティ (続き)

- リソースタイプの宣言, 35
- リソースの設定, 34, 56
- リソースの宣言, 38
- リソースの変更, 56

プロパティ関数, DSDL, 20

プロパティ属性, リソース, 301

## プロパティ値

- 規則, 361
- デフォルト, 84

プロパティ変数, 181

- そのタイプを Agent Builder がどのように置き換えるか, 183

の構文, 182

のリスト, 181

リソースグループのリスト, 182

リソースタイプのリスト, 182

リソースのリスト, 182

プロパティ名, 規則, 359

## 文書の要件

- Tunable 属性の制約, 88
- アップグレードの, 88-90

## へ

## 変数

- プロパティ, 181
- プロパティの構文, 182
- プロパティのタイプを Agent Builder がどのように置き換えるか, 183
- プロパティのリスト, 181
- リソースグループプロパティのリスト, 182
- リソースタイププロパティのリスト, 182
- リソースプロパティのリスト, 182

ベンダー間の区別, *vendor-id*, 80

## ま

## マスター

説明, 23

マニュアルページ, Agent Builder, 188

## め

## 命名規則

- 関数, 147
- コールバックメソッド, 147

## メソッド

- Boot, 51, 75, 141
- Fini, 50-51, 74, 141
- Fini, 実装のためのガイドライン, 51
- Init, 50, 74, 141
- Monitor\_check, 77, 140
- Monitor\_check コールバック, 77
- Monitor\_start, 77, 139
- Monitor\_start コールバック, 77
- Monitor\_stop, 77, 139
- Monitor\_stop コールバック, 77
- Postnet\_start, 77
- Postnet\_start コールバック, 77
- Prenet\_start, 77
- Prenet\_start コールバック, 77
- Start, 47, 73, 136
- Stop, 47, 73, 137
- Update, 56, 76, 140
- Update コールバック, 76
- Validate, 56, 76, 134
- Validate コールバック, 76
- xfnts\_monitor\_check, 156
- xfnts\_monitor\_start, 154
- xfnts\_monitor\_stop, 155
- xfnts\_start, 148
- xfnts\_stop, 153
- xfnts\_update, 166
- xfnts\_validate, 163
- コールバック, 56
  - 初期化, 72
  - 制御, 72
- 呼び出し回数への非依存性, 45
- メソッドコード, 変更, 86
- メソッド引数, RMAPI, 71
- メッセージ
  - SC\_CALLBACK\_REG CRNP, 225, 226-227
  - SC\_EVENT CRNP, 229, 230
  - SC\_REPLY CRNP, 227, 228
- メッセージログ, リソースへの追加, 55

## メニュー

- Agent Builder, 174
- Agent Builder の「ファイル」, 174
- Agent Builder の「編集」, 175

## も

- モニターコード, 変更, 86

## ゆ

- 有効な名前, RGM (Resource Group Manager), 359
- ユーティリティー関数
  - DSDL, 220
  - RMAPI, 71

## よ

- 呼び出し回数への非依存性, メソッド, 45

## り

## リソース

- 間の依存関係の調節, 62
- 監視, 52
- 起動, 46
- スケーラブルリソースの実装, 57
- 説明, 22
- 停止, 46
- フェイルオーバーの実装, 56
- メッセージログの追加, 55
- リソース関数, RMAPI, 67
- リソースグループ
  - スケーラブル, 23
  - 説明, 22
  - フェイルオーバー, 23
  - プロパティ, 23
- リソースグループ関数, RMAPI, 69
- リソースグループコマンド, RMAPI, 66
- リソースグループプロパティ, 285
  - Auto\_start\_on\_new\_cluster, 286

## リソースグループプロパティ (続き)

- Desired primaries, 286
- Failback, 286
- Global\_resources\_used, 286
- Implicit\_network\_dependencies, 287
- Maximum primaries, 287
- Nodelist, 287
- Pathprefix, 287
- Pingpong\_interval, 288
- Resource\_list, 288
- RG\_affinities, 288
- RG\_dependencies, 289
- RG\_description, 290
- RG\_is\_frozen, 290
- RG\_mode, 290
- RG\_name, 291
- RG\_project\_name, 291
- RG\_slm\_cpu, 291
- RG\_slm\_cpu\_min, 292
- RG\_slm\_pset\_type, 294
- RG\_slm\_type, 293
- RG\_state, 297
- RG\_system, 299
- Suspend\_automatic\_recovery, 299
- についての情報へのアクセス, 44
- リソースグループ名, 規則, 359
- リソースコマンド, RMAPI, 66
- リソースタイプ
  - DSDLによるデバッグ, 130
  - アップグレード時の処理, 84
  - アップグレードの要件, 79
  - 関数
    - RMAPI, 68
  - コマンド
    - RMAPI, 66
  - 説明, 21
  - 複数のバージョン, 79
  - 変更, 79
- リソースタイプ登録, 「RTR」を参照 (リソースタイプ登録)
  - ファイル
    - アップグレード, 80
  - リソースタイプ登録 (RTR), 説明, 24
  - リソースタイプのアップグレード, 79

- リソースタイプの変更, 79
- リソースタイプパッケージ, インストール要件, 85
- リソースタイププロパティ, 253
  - API\_version, 254
  - Boot, 255
  - Failover, 255
  - Fini, 256
  - Global\_zone, 257
  - Init, 258
  - Init\_nodes, 258
  - Installed\_nodes, 258
  - Is\_logical\_hostname, 258
  - Is\_shared\_address, 259
  - Monitor\_check, 259
  - Monitor\_start, 259
  - Monitor\_stop, 259
  - Pkglist, 259
  - Postnet\_stop, 260
  - Prenet\_start, 260
  - Proxy, 260
  - Resource\_list, 261
  - Resource\_type, 261
  - RT\_basedir, 261
  - RT\_description, 262
  - RT\_system, 262
  - RT\_version, 262
  - Single\_instance, 262
  - Start, 262
  - Stop, 263
  - Update, 263
  - Validate, 263
  - Vendor\_ID, 263
  - 設定, 34
  - 宣言, 35
- リソースタイプ名
  - Sun Cluster 3.0, 83
  - 完全修飾名の取得, 80
  - 規則, 360
  - 実装, 84
  - 制限, 83, 176
  - バージョン接尾辞, 80
  - バージョン接尾辞なし, 83
- リソースタイプモニター, 実装, 84
- リソースの依存関係, 調節, 62
- リソースプロパティ, 264
  - Affinity\_timeout, 264
  - Boot\_timeout, 264
  - Cheap\_probe\_interval, 265
  - Failover\_mode, 265
  - Fini\_timeout, 269
  - Init\_timeout, 269
  - Load\_balancing\_policy, 269
  - Load\_balancing\_weights, 269
  - Monitor\_check\_timeout, 270
  - Monitor\_start\_timeout, 270
  - Monitor\_stop\_timeout, 270
  - Monitored\_switch, 270
  - Network\_resources\_used, 271
  - Num\_resource\_restarts, 271
  - Num\_rg\_restarts, 272
  - On\_off\_switch, 272
  - Port\_list, 272
  - Postnet\_stop\_timeout, 273
  - Prenet\_start\_timeout, 273
  - R\_description, 273
  - Resource\_dependencies, 273
  - Resource\_dependencies\_offline\_restart, 275
  - Resource\_dependencies\_restart, 276
  - Resource\_dependencies\_weak, 278
  - Resource\_name, 279
  - Resource\_project\_name, 279
  - Resource\_state, 280
  - Retry\_count, 280
  - Retry\_interval, 281
  - Scalable, 281
  - Start\_timeout, 282
  - Status, 282
  - Status\_msg, 282
  - Stop\_timeout, 283
  - Thorough\_probe\_interval, 283
  - Type, 283
  - Type\_version, 283
  - UDP\_affinity, 284
  - Update\_timeout, 284
  - Validate\_timeout, 284
  - Weak\_affinity, 285
  - 拡張, 265
  - 設定, 34, 56

リソースプロパティ (続き)

宣言, 38

についての情報へのアクセス, 44

変更, 56

リソースプロパティの属性, 301

Array\_maxsize, 301

Array\_minsize, 301

Default, 301

Description, 301

Enumlist, 301

Extension, 301

Max, 301

Maxlength, 301

Min, 301

Minlength, 301

Per\_node, 301

Property, 302

Tunable, 302

型, 302

リソース名, 規則, 359

れ

例, CRNP を使用する Java アプリケーション, 233

列挙型リテラル名, 規則, 359

ろ

ログ, リソースへの追加, 55

コ

コマンド

で GDS を作成, 199