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

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

Part No: 820–6930–10
2009 年 1 月、 Revision A

090219@21990

目次

はじめに

『Sun Cluster データサービス開発ガイド (Solaris OS 版)』では、Sun™ Cluster データサービスを開発するための リソース管理 API の使用に関する情報が記載されていますSPARC® と x86 ベース両方のシステムの Sun Cluster 製品に関する概念的情報と参照情報が記載されています。

注 – この Sun Cluster 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、AMD64、および Intel 64) を使用するシステムをサポートします。このドキュメントでは、x86 とは 64 ビット x86 互換製品の広範囲なファミリーを指します。このドキュメントの情報では、特に明示されている場合以外はすべてのプラットフォームに関係します。

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。

内容の紹介

『Sun Cluster データサービス開発ガイド (Solaris OS 版)』は、次の章と付録で構成されています。

第 1 章「リソース管理の概要」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「データサービスの開発」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「リソース管理 API リファレンス」では、リソース管理 API (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「リソースタイプの変更」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「サンプルデータサービス」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紹介します。

第 6 章「データサービス開発ライブラリ」では、データサービス開発ライブラリ (Data Service 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「リソースタイプの設計」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「サンプル DSDL リソースタイプの実装」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「Sun Cluster Agent Builder」では、Sun Cluster Agent Builder について説明します。

第 10 章「汎用データサービス」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「DSDL API 関数」では、DSDL API 関数について説明します。

第 12 章「クラスタ再構成通知プロトコル」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スケーラブル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「標準プロパティー」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ーについて説明します。

付録 B「データサービスのコード例」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「サンプル DSDL リソースタイプのコード例」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「有効な RGM 名と値」では、リソースグループマネージャー (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「非クラスタ対応のアプリケーションの要件」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用させる要件を説明します。

付録 F「CRNP のドキュメントタイプ定義」では、CRNP のドキュメントタイプ定義を説明します。

付録 G 「CrnpClient.java アプリケーション」では、第 12 章「クラスタ再構成通知プロトコル」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

関連のある Sun Cluster のトピックについては、次の表に示したマニュアルを参照してください。Sun Cluster に関するマニュアルはすべて、http://docs.sun.com で参照できます。

項目	マニュアル
概要	『Sun Cluster の概要 (Solaris OS 版)』
	『Sun Cluster 3.2 1/09 Documentation Center 』
概念	『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)』
	『Sun Cluster クイックスタートガイド (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)』
	『Sun Cluster Quick Reference 』
ソフトウェアアップグレード	『Sun Cluster Upgrade Guide for Solaris OS 』
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 Data Services Reference Manual for Solaris OS 』
	『Sun Cluster Quorum Server Reference Manual for Solaris OS 』

Sun Cluster ドキュメントの完全なリストについては、http://wikis.sun.com/display/SunCluster/Home/ で Sun Cluster ソフトウェアの使用しているリリースのリリースノートを参照してください。

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 1/09)

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
`prtconf -v`	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
`psrinfo -v`	プロセッサの情報を表示する
`showrev -p`	インストールされているパッチを報告する
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%
AaBbCc123	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% **su** password:
AaBbCc123	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 *filename* 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。
「」	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% **grep '^#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。

◆ ◆ ◆ 第 1 章

1 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。

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ーを備えたリソースとして実行および管理できます。リソースグループマネージャー (Resource Group Manager、RGM) は、高可用性とスケーラビリティー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。
- リソース管理 API (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。
- プロセス監視機能 (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。

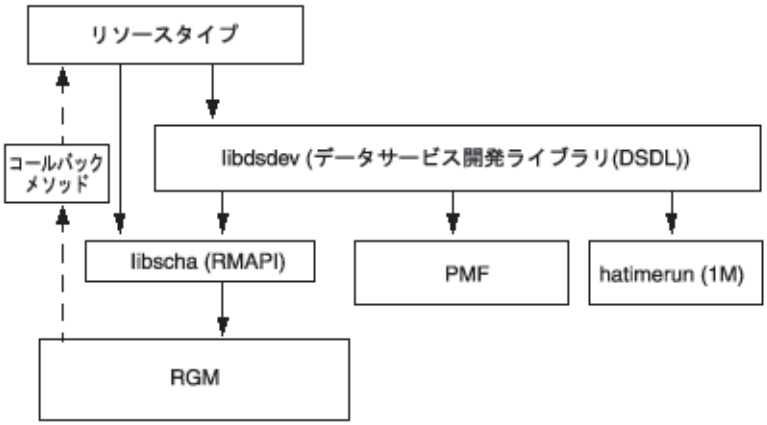


図 1–1　Sun Cluster アプリケーション環境のプログラミングアーキテクチャー

Sun Cluster Agent Builder (第 9 章「Sun Cluster Agent Builder」 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、事前に構成したポリシーに従って、選択した Solaris ホスト上のリソースを自動的に起動および停止します。RGM

は、ノードの障害やリブ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上でリソースを停止し、別のノード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て、障害が発生しているリソースを別のノードに再配置するなど、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、Solaris ホストあたり 1 つのノードでしかオンラインにできません。

リソースグループマネージャーモデル

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サーバーを実装します。クラスタ管理者は、財務カレンダ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ーセット

RGM は、リソースタイププロパティー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ーを指定し、プロパティーの値をリソースタイプ登録 (Resource Type Registration、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ーの設定」、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。

255 ページの「資源タイプのプロパティー」に、リソースタイププロパティー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ーと値を継承します。さらに、開発者は、RTR ファイルでリソースプロパティーを宣言できます。266 ページの「リソースのプロパティー」にはリソースプロパティー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ーがどのように指定されているかに応じて、特定のプロパティーの値を変更できます。たとえば、プロパティー定義は値の許容範囲を指定できます。プロパティー定義には、いつプロパティー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ーを変更できます。

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ー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、必ずリソースグループ内に構成されます。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す。

リソースグループがオンラインになっているノードを主ノードと呼びます。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` プロパティーを持っており、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` プロパティーを設定します。

リソースグループはプロパティーセットも持っています。このようなプロパティーには、クラスタ管理者が設定できる構成プロパティーや、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ーがあります。

RGM は、フェイルオーバーとスケーラブルという 2 種類のリソースグループを定義します。フェイルオーバーリソースグループは、常に単一のノード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グループは、同時に複数のノード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ーセットを提供します。このようなプロパティーの詳細については、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45 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。

288 ページの「 リソースグループのプロパティー」には、リソースグループプロパティーのリストがあります。

リソースグループマネージャー

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、クラスタ内のグローバルクラスタ投票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。

- ノードで障害が発生した場合、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。個々のリソースでの状態変更を有効にするため、RGM は、各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ッソドのセットと、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す。
- 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に移動される前に、ノードの信頼性を査定します。

コールバックメソッドの詳細は、第 3 章「リソース管理 API リファレンス」、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「サンプルデータサービス」と第 8 章「サンプル DSDL リソースタイプの実装」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ーは、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。

リソース管理 API

RMAPI (リソース管理 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 関数の詳細は、`scha_calls(3HA)`のマニュアルページと、第 3 章「リソース管理 API リファレンス」を参照してください。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例については、第 5 章「サンプルデータサービス」も参照してください。

データサービス開発ライブラリ

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.pmfd(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 の詳細は、第 6 章「データサービス開発ライブラリ」、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。追加プロパティー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。

リソースグループマネージャー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 (Graphical User Interface、GUI) とコマンドセットの両方を提供します。

Sun Cluster Manager

Sun Cluster Manager は、次に示す作業を実行できる 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

ほとんどの Sun Cluster 管理作業は、 clsetup(1CL) ユーティリティーを使用して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ーの変更
- 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 の 各コマンドでは、RGM で使用されるリソースオブジェクトや、リソースタイプ、リソースグループの表示、作成、構成、および削除ができます。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。ただし、clresourcetype、clresourcegroup、および clresource の 各コマンドは、API が動作するクラスタ構成を構築するツールとしても使用できます。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については、clresourcetype(1CL)、clresourcegroup(1CL)、および clresource(1CL) の各マニュアルページを参照してください。

◆ ◆ ◆ 第 2 章

2 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ー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。

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ー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ー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ーを備えるための要件を要約します。詳細情報を確認する場合や、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 「データサービスのコード例」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ー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、367 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。366 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、366 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。 不便な点は、順次アップグレードを使用する場合、アプリケーションが RGM 制御のもとで動作している間はバイナリが使用されることです。

- 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できる必要があります。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、369 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なりません。

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、57 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。

- リソース管理 API (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、使用する方法に関わらず、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。

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

-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 Solaris ホスト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ーの設定

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ーおよびリソースプロパティーのセットを提供します。リソースタイププロパティーでは、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。すべてのリソースタイププロパティーのリストについては、255 ページの「資源タイプのプロパティー」を参照してください。

リソースプロパティー (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ー (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ーについては、266 ページの「リソースのプロパティー」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ーは、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 (Resource Type Registration、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、一連のプロパティーを宣言します。たとえば、特定のプロパティー (`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ーも RTR ファイルで宣言されていなければ、クラスタ管理者はそれらのプロパティーを利用できません。いくつかのプロパティー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 がそのプロパティーを定義し、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ーの宣言

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ー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ーは、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ー `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ー名は大文字と小文字が区別されません。プロパティー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 = "smpl";
Vendor_id = SUNW;
RT_description = "Sample Service on Sun Cluster";

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

Init_nodes = RG_PRIMARIES;

RT_basedir=/opt/SUNWsmpl/bin;

Start           =    smpl_svc_start;
Stop            =    smpl_svc_stop;

Validate        =    smpl_validate;
Update          =    smpl_update;

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

ヒント – RTR ファイルの最初のエントリには、`Resource_type` プロパティー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` プロパティー (この例では「`smpl`」) 単独で指定できます。`Vendor_id` プロパティーを接頭辞として使用し、リソースタイプ (この例では「`SUNW.smpl`」) 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バージョンの Sun Cluster には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。このプロパティーについては、255 ページの「資源タイプのプロパティー」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。このプロパティーについては、255 ページの「資源タイプのプロパティー」の `Failover` のエントリで詳しく説明しています。

`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_PRIMARIES`
データサービスをマスターできるノード上でのみ、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_PRIMARIES` で指定されたノードは、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のサブセットです。この値に `RT_INSTALLED_NODES` を設定した場合、RGM 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上でこれらのメソッドを呼び出します。

`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。

ゾーンクラスタのリソースタイププロパティーの宣言

ユーザー (およびクラスタ管理者) は、ゾーンルートパス以下に RTR ファイルを作成することによって、特定のゾーンクラスタ内で使用するリソースタイプを登録できます。この RTR ファイルを正しく構成するために、次の条件を満たしていることを確認します。

- RTR ファイル内で `Global_zone` プロパティーが `FALSE` に設定されているか、またはまったく設定されません。ユーザーが `Global_zone` プロパティーを指定しない場合、このプロパティーはデフォルトでは `FALSE` に設定されます。
- RTR ファイルが、論理ホスト名または共有アドレスタイプではありません。

ゾーンクラスタのリソースタイプは、RTR ファイルを `/usr/cluster/lib/rgm/rtreg/` ディレクトリに置くことで登録することもできます。このディレクトリ内の RTR ファイルで宣言したリソースタイププロパティーをクラスタ管理者が構成することはできません。

`/opt/cluster/lib/rgm/rtreg/` ディレクトリ内の RTR ファイルで定義されたリソースタイプは、グローバルクラスタ専用として使用されます。

リソースプロパティーの宣言

リソースタイププロパティーと同様に、リソースプロパティーも RTR ファイルで宣言します。便宜上、リソースプロパティー宣言は RTR ファイルのリソースタイププロパティー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ー (つまり、「システム定義プロパティー」) 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ー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。その結果、プロセスグループのすべてのメンバーが、メソッドがタイムアウトを超過したノード上の /var/cluster/core ディレクトリまたは /var/cluster/core ディレクトリのサブ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ー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ー (拡張プロパティー) を定義することもできます。304 ページの「リソースプロパティーの属性」に、リソースプロパティーを変更および定義するための属性を示します。拡張プロパティー宣言は RTR ファイルのシステム定義プロパティー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ー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。

```
...

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

{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
```

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

プロパティー名 (PROPERTY = *value*) は、各リソースプロパティー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ーは、RTR ファイルのプロパティー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。304 ページの「リソースプロパティーの属性」にリソースプロパティー属性のリストを示します。

リソースプロパティーの次のセットは、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ーを定義します。

```
{
        PROPERTY = Failover_mode;
        DEFAULT=SOFT;
        TUNABLE = ANYTIME;
}
{
        PROPERTY = Thorough_Probe_Interval;
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
```

```
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

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

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

これらのリソースプロパティー宣言には、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ー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。

上記のプロパティーのほとんどは、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ーのあとには、次のような情報が続きます。詳細は、266 ページの「リソースのプロパティー」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`
: 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リソースのリスト。このリストには、プロパティー `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` プロパティーが宣言されている場合、RGM は自動的にこのプロパティーを作成します。`Scalable` プロパティーが 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。

  ユーザーが `Network_resources_used` プロパティーに値を割り当てていない場合、その値は、リソース依存関係のプロパティーの設定に基づいて、RGM によって自動的に更新されます。このプロパティーを直接設定する必要はありません。その代わりに
  、`Resource_dependencies`、`Resource_dependencies_offline_restart`、`Resource_dependencies_r`
  または `Resource_dependencies_weak` プロパティーを設定します。

  旧リリースの Sun Cluster ソフトウェアとの互換性を維持するために
  、`Network_resources_used` プロパティーの値を直接に設定することもできます。ユーザーが `Network_resources_used` プロパティーの値を直接設定した場合は
  、`Network_resources_used` プロパティーの値がリソース依存関係のプロパティーの設定に基づいて更新されることはありません。リソース名を `Network_resources_used` プロパティーに追加すると、そのリソース名は自動的に `Resource_dependencies` プロパティーにも追加されます。この依存関係を削除する唯一の方法は、`Network_resources_used` プロパティーからその依存関係を削除することです。ネットワークリソースの依存関係が元々 `Resource_dependencies` プロパティーに追加されたものなのか `Network_resources_used` プロパティーに追加されたものなのかがわからない場合は、両方のプロパティーから依存関係を削除します。

`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。このプロパティーを FALSE に設定した場合、リソースタイププロパティー 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。このプロパティーの詳しい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45 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ーを自動的に宣言します。ただし、これらのプロパティー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`
アプリケーションが待機するポートのリストです。Agent Builder がこのプロパティーを宣言するため、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。

拡張プロパティーの宣言

拡張プロパティーは、サンプル RTR ファイルの最後に出現します。

```
# Extension Properties
#

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

# The following two properties control restart of the fault monitor.
{
        PROPERTY = Monitor_retry_count;
        EXTENSION;
        INT;
        DEFAULT = 4;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Number of PMF restarts allowed for fault monitor.";
}
{
```

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

# Child process monitoring level for PMF (-C option of pmfadm).
# Default of -1 means to not use the -C option of pmfadm.
# A value of 0 or greater indicates the desired level of child-process.
# monitoring.
{
        PROPERTY = Child_mon_level;
        EXTENSION;
        INT;
        DEFAULT = -1;
        TUNABLE = ANYTIME;
        DESCRIPTION = “Child monitoring level for PMF";
}
# User added code -- BEGIN VVVVVVVVVVV
# User added code -- 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ーを作成します。

`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。このプロパティー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ーを作成できます。

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ー情報へのアクセス

一般に、コールバックメソッドはリソースのプロパティー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ーと拡張プロパティー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ーにアクセスするための C 関数セット (プロパティーごとに 1 つ) と、拡張プロパティー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ーを設定する API 関数が存在しないため、プロパティー機構を使用して、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 コマンドを使用するか、または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って、特定のリソースプロパティーを設定できます。ただし、 clresource はコールバックメソッドから呼び出さないでください。 これは、クラスタの再構成時、つまり GRM がメソッドを呼び出すときに clresource がエラーになるためです。

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ーは、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 プロパティーが 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 -Z *zonename* 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非大域ゾーンの Solaris ゾーン名を表します。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、-Z *zonename* 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ーの詳細は、255 ページの「資源タイプのプロパティー」および rt_properties(5)のマニュアルページ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 (Generic Data Service、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ー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ー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「汎用データサービス」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に、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、クラスタノードのクラッシュ後、RGM は、そのノードがマスターしているリソースグループを新しいノードに移動します。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はありますが) 生き残ったホストノード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 プロパティー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、リソースがホスト 1 上のゾーン A 内でオフラインにされ、ホスト 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、ホスト 1 上のゾーン A 内のすべての活動を停止しようとします。ホスト 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、ホスト 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。

ローカルノード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 Stop メソッドは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 プロパティーは 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。その結果、プロセスグループのすべてのメンバーが、メソッドがタイムアウトを超過したノー

ド上の /var/cluster/core ディレクトリまたは /var/cluster/core ディレクトリのサブ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。

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際には、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ー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ー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ー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ー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 引数を持つ scha_control() または clnode evacuate コマンドによるスイッチオーバー) のあとで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、次の状態が発生した場合、リソースが管理されなくなったノード上で 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、RGM はノードリストのすべてのノード上で 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、RGM はノードリストのすべてのノード上で 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が削除されます。この場合、RGM は削除されたノード上でのみ 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 プロパティーの設定に依存します。Init_nodes プロパティーは RG_PRIMARIES または RT_INSTALLED_NODE 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_PRIMARIES 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。

特定のノード上でのみリソースが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判定することによって、これら 2 つの場合を区別できます。

ローカルのノードがリソースグループのノードリストに含まれている場合は、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。

ローカルのノードがリソースグループのノードリストに含まれていない場合は、Fini メソッドでそのローカルノード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 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を探します。すべてのノードで 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 プロパティーを設定します。これらのプロパティーを設定するには、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 プロパティー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ー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ノードでメソッドを実行します。ただし、リソースグループが非大域ゾーン内 (つまりゾーンクラスタノードまたはグローバルクラスタ非投票ノードのいずれか) で構成されている場合でも、大域ゾーン内ですべてのメソッドを実行することが必要なリソースタイプもいくつかあります。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 (Resource Type Registration、RTR) ファイルの中で Global_zone プロパティーを 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 プロパティーに 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。このプロパティーに 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=TRUE を宣言するリソースタイプは、Global_zone_override リソースプロパティーを宣言する場合もあります。この場合、Global_zone_override プロパティーの値は、そのリソースの Global_zone プロパティーの値より優先されます。Global_zone_override プロパティーの詳細は、266 ページの「リソースのプロパティー」 および r_properties(5)のマニュアルページを参照してください。

Global_zone リソースタイププロパティーが 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ノードで実行されます。

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ノードで暗黙的に動作します。Global_zone リソースタイププロパティーが 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ーが 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。

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ーおよびリソースグループプロパティーを取得します。そのほかの DSDL 問い合わせは、そのノード上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 Solaris ホスト上の複数のノードが含まれている。

  注 – 複数のノードを構成できるのは、グローバルクラスタノード上でのみです。各 Solaris ホストのゾーンクラスタでは 1 つのノードしか構成できません。

- その同じリソースグループに、Global_zone プロパティーが TRUE に設定されたリソースが 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、221 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ー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、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とき、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ー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ー値を `Validate` メソッドに渡します。RGM は、リソースのタイプの `Init_nodes` プロパティーが示すクラスタノードのセット上で `Validate` を呼び出します。`Init_nodes` については、255 ページの「資源タイプのプロ

パティー」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ーを変更したときだけです。RGM がプロパティーを設定したときや、モニターが Status と Status_msg リソースプロパティー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、プロパティー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ー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ー値を読み取り、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、依存しているデータサービスリソースとともに、クラスタノード間を移動します。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ーをいくつか提供します。

グローバルクラスタでは、フェイルオーバーリソースグループは別の Solaris ホストまたは同じ Solaris ホスト上のノードにフェイルオーバーできます。フェイルオーバーリソースグループは、この方法でゾーンクラスタにフェイルオーバーすることはできません。ただし、ホストで障害が発生すると、同一ホスト上のノードにこのリソースグループをフェイルオーバーした場合に高可用性は得られません。とはいえ、同一ホスト上のノード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ーを TRUE に設定し、同時に複数のノード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。このプロパティーのデフォルト値は 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ーは、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ーを使用すると、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 はグループの Maximum_primaries プロパティーを 1 に設定します。 さらに、RGM は、リソースグループが単独のノードでマスターされるように制限します。Failover プロパティーが 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ーは、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 プロパティーのデフォルトは 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でオンラインになることができます。スケーラブルリソース (ネットワーク負荷分散を使用) は、グローバルクラスタ非投票ノードでも動作するように構成できます。ただし、そのようなスケーラブルリソースを実行できるのは、Solaris ホストごとに 1 つのノード内だけです。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ー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ーを 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ーは、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にクライアントに見せるため)、Scalable には TRUE を設定します。

RG_mode プロパティーを使用すると、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 の場合は、RGM では Maximum_primaries を 1 より大きな値に設定できます。つまり、リソースグループを同時に複数のノードでマスターすることが可能になります。RGM は、Failover プロパティーが 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ーを使用して、あるノード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 プロパティーを設定して、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 プロパティーを宣言した場合、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ーのセットを自動的に作成します。

Network_resources_used
: このリソースが依存関係を有する共有アドレスリソースを特定します。このリストには、プロパティー 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 プロパティーが宣言されている場合、RGM は自動的にこのプロパティーを作成します。Scalable プロパティーが 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。

  ユーザーが Network_resources_used プロパティーに値を割り当てていない場合、その値は、リソース依存関係のプロパティーの設定に基づいて、RGM によって自動的に更新されます。このプロパティーを直接設定する必要はありません。その代わりに、Resource_dependencies、Resource_dependencies_offline_restart、Resource_dependenc または Resource_dependencies_weak プロパティーを設定します。

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 プロパティー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 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
アプリケーションが待機するポートを特定します。このプロパティー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ーと Scalable リソースプロパティーの両方を FALSE に宣言します。Scalable プロパティーは作成時に調整できるように指定します。

Failover プロパティーの値が 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 プロパティーの 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 プロパティーが 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ーの妥当性を検査します。プロパティー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。

- スケーラブルリソースは、1 つ以上の既存の共有アドレスリソースに対するリソースの依存関係を宣言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に関する Nodelist 内のすべてのノードは、SharedAddress リソースの NetIfList プロパティーに表示されている必要があります。

  スケーラブルリソースグループの Nodelist は、次に示す 2 つのノードリストのサブセット、組み合わせ、または結合したものであるか、あるいはこれらのノードリストと同一であることが必要です。

  - SharedAddress リソースを含むリソースグループの Nodelist。
  - SharedAddress リソースの AuxNodeList プロパティーに一覧表示されている Nodelist。

  注 – 共有アドレスのリソースグループのノードリストにスケーラブルリソースのリソースグループに関するノードリストのすべてのノードを含める場合は、AuxNodeList プロパティーを設定する必要はありません。

- スケーラブルリソースを含むリソースグループの RG_dependencies プロパティーは、スケーラブルリソースの Network_resources_used プロパティーに存在する、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 プロパティー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な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。

HA データサービスは、クラスタ内のすべての Solaris ホスト上ではなく、1 つの Solaris ホスト上のグローバルクラスタ非投票ノードで検証します。データサービスがグローバルクラスタ非投票ノード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。ホスト上のグローバルクラスタ非投票ノード内で動作している HA データサービスは、正常に動作していない場合でも、ほかのノード内またはほかのホスト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も検証します。

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。`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。依存しているリソースと依存されているリソースが異なるノード上で動作で

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リソースとは別のノード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 つの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ーを指定します。

  このアフィニティーは、`clresource` コマンドで `RG_affinities` プロパティーを使用して指定します コマンドでデバイス ID を更新して管理します。

カレンダ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間でやりとりを無限に繰り返す可能性があります。

◆ ◆ ◆ 第 3 章

3 リソース管理 API リファレンス

この章では、リソース管理 API (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で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ー、およびそのほか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` プロパティーと `Status_msg` プロパティーを設定する。
- 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、ノード名、ゾーン名、ID、状態、リソースグループなど) 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 プロパティーを設定する。
- 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()、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 プロパティーを設定します。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 プロパティーを設定します。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()`
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、ノード名、ゾーン名、ID、状態、リソースグループなど) 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、RGM 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。

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、API によって提供される、リソースタイプを実装するための主要な要素です。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 (ノードで障害が発生したときなど) 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、書き込み可能にしてはなりません。

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の詳細は、`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す。

- `-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す。

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ー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上でオンラインになったとき、RGM はそのクラスタノード上でこのメソッドを実行します。このメソッドは、そのノード上でリソースを起動します。

  ローカルノード上でリソースが起動され、利用可能になるまで、Start メソッドは終了できません。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 プロパティーの設定によって異なります。

リソースタイプ登録 (Resource Type Registration、RTR) ファイルの Start_timeout プロパティーが、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上でオフラインになったとき、RGM はそのクラスタノード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いけません。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。その結果、プロセスグループのすべてのメンバーが、メソッドがタイムアウトを超過したノード上の /var/cluster/core ディレクトリまたは /var/cluster/core ディレクトリのサブ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 プロパティーの設定によって異なります。266 ページの「リソースのプロパティー」を参照してください。

RTR ファイルの Stop_timeout プロパティーが、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ーにより特定されるノード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。

RGM は、次のが状態が発生した場合、リソースが管理されなくなったノード上で `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、RGM はノードリストのすべてのノード上で `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、RGM はノードリストのすべてのノード上で `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が削除されます。この場合、RGM は削除されたノード上でのみ `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` プロパティーの設定に依存します。`Init_nodes` プロパティーは `RG_PRIMARIES` 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_PRIMARIES` 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。

特定のノード上でのみリソースが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判定することによって、これら 2 つの場合を区別できます。

ローカルのノードがリソースグループのノードリストに含まれている場合は、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。

ローカルのノードがリソースグループのノードリストに含まれていない場合は、Fini メソッドでそのローカルノード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に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上のリソースを初期化します。RGM は、Init_nodes リソースプロパティーにより特定されるノード上でこのメソッドを実行します。Boot メソッドが呼び出されるのは、起動または再起動の結果としてノードがクラスタに結合または再結合したときです。

  Global_zone リソースタイププロパティーが TRUE に等しい場合、リソースを含むリソースグループがグローバルクラスタ非投票ノードで動作するように構成されているときでも、メソッドはグローバルクラスタ投票ノードで実行されます。

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ー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ーの更新時、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` プロパティーにより特定されるクラスタノード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。

`Validate` は、クラスタ管理者によってリソースプロパティーまたはリソースグループプロパティー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ー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ー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、プロパティー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ー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ー値を読み取り、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 `Prenet_start` と `Postnet_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。

`Prenet_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。

`Postnet_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に再配置する前に、そのノード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

4 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。

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ーの属性
- 標準プロパティー、拡張プロパティーを含む宣言済みリソースプロパティーセット

- default、 min、max、arraymin、arraymax 、tunability などのリソースプロパティーの属性
- 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。

この節の内容は、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ーです。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、362 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 #$upgrade_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 プロパティー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、このプロパティーの値を選択します。

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 ([)
- 右角括弧 (])

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ー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ーを使用し、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ー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ーは、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ーのリソースプロパティー属性を継承します。
- 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。プロパティーが `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ーも、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ー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ー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、プロパティーに値を指定し、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。Sun Cluster 3.0 と Sun Cluster 3.1 で正しく動作するためには、このリソースタイプのモニターコードが次に示す両方の命名規則を処理できることが必要です。

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

リソースタイプ名の形式は、362 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際には、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
- プロパティーのデフォルト値または tunable 属性が変更されたか
- プロパティーの 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ーが追加されたか、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ーのデフォルト値や 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ー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、既存のプロパティー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ー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ーの `min` 属性、`max` 属性、または `type` 属性が変更された場合は、クラスタ管理者がリソースタイプを更新したときに `clresourcetype(1CL)` コマンドによりこれらの制約が自動的に検査されます。

アップグレードにより新しいプロパティーが追加された場合や古いプロパティーが削除された場合、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バ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定します。

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。

プロパティー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 プロパティーの 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ーを変更します。	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がありません。	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ーを追加します。新しいメソッドには新しいプロパティーが必要です。 目的は、ノード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ーを追加します。新しいメソッドは新しいプロパティーを必要としません。	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す。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ーを説明する
- プロパティー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ー属性を述べる
- 必要に応じて、既存のリソースプロパティー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ラ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ー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ー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ーのデフォルト設定
- 導入されたリソースタイプの新しい拡張プロパティー
- 取り消されたリソースタイプの既存のプロパティー
- リソースタイプに対して宣言された標準プロパティーのセットに対する変更
- 変更されたリソースプロパティー (`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

◆ ◆ ◆ 第 5 章

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 デーモンは Solaris におけるドメインネームサービス (Domain Name Service、DNS) の実装です。サンプルのデータサービスでは、リソース管理 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。

リソース管理 API 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。

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、アプリケーションエラー、ノード障害など) 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 (Process Monitor Facility、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ーを定義します。
- `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ー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 (Resource Type Registration、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ーを継承します。

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リソースグループマネージャー (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/rtreg/` 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ー、拡張プロパティー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ーの設定」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ー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、307 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ー

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ーが続きます。

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ー名は大文字と小文字が区別されません。プロパティー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` プロパティー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ーについての情報を説明します。

- リソースタイプ名は、`Resource_type` プロパティーだけで指定できます (例: `sample`)。あるいは、接頭辞 *vendor-id* + ピリオド (.) + リソースタイププロパティー (例: `SUNW.sample`) という形式でも指定できます。

  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` プロパティーは、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` プロパティーは 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 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バージョンの Sun Cluster には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。このプロパティーについては、255 ページの「資源タイプのプロパティー」の `API_version` の項目で詳しく説明しています。
- `Failover = TRUE` は、複数のノード上で同時にオンラインにできるリソースグループでは、データサービスが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ー (`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ーの完全なリストとそのデフォルト値については、255 ページの「資源タイプのプロパティー」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ー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ー

慣習上、RTR ファイルでは、次のリソースプロパティーをリソースタイププロパティーのあとに宣言します。リソースプロパティーには、Sun Cluster ソフトウェアが提供するシステム定義プロパティーと、データサービス開発者が定義する拡張プロパティーが含まれます。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ー属性の数 (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ー

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ーを示しています。

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

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

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

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

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

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

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

{
   PROPERTY = Network_resources_used;
   TUNABLE = AT_CREATION;
   DEFAULT = “”;
}
```

Sun Cluster ソフトウェアはシステム定義プロパティーを提供しますが、リソースプロパティー属性を使用すると、異なるデフォルト値を設定できます。リソースプロ

パティー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、304 ページの「リソースプロパティーの属性」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ーについては、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。
- プロパティー `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ーの値を変更できることを意味します。上記のプロパティーは、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ー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ーが宣言されている場合にかぎりプロパティーを作成します。

  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ーを使用しているため、開発者はこれらのプロパティーを RTR ファイルで宣言する必要があります。プロパティー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266 ページの「リソースのプロパティー」を参照してください。

RTR ファイルの拡張プロパティー

次に、RTR ファイルの最後の例として、拡張プロパティーを示します。

```
# Extension Properties

# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, DNS, specify the path of the DNS configuration file on
# PXFS (typically named.conf).
```

```
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = “The Configuration Directory Path”;
}

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

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に使用する次の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

   # Send a SIGTERM signal to the data service and wait for 80% of the
```

```
   # total timeout value.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info \
          -t [$SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"
```

dns_monitor_start、dns_monitor_stop、dns_update の各メソッドは、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
...
# See if the monitor is running, and if so, kill it.
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 プロパティー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ー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「プロパティー更新の処理」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。

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

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

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

詳細は、scha_cluster_get(1HA)のマニュアルページを参照してください。

プロパティー情報の取得

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ー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ーと拡張プロパティーの両方が使用できます。システム定義プロパティーは事前に定義されています。拡張プロパティーは、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ーの値を取得するときは、-O オプションでプロパティーの名前を指定します。このコマンドは、プロパティー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 プロパティー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‘
```

拡張プロパティーの場合、データサービス開発者は、このプロパティーが拡張プロパティーであることを示す -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ーの名前を指定する必要があります。拡張プロパティーの場合、このコマンドは、プロパティー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 拡張プロパティー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、48 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を含むリソースグループがクラスタノード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でリソースが有効であるとき、RGM はそのノード上で 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 メソッドはそのホストのグローバルクラスタ投票ノードで 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、311 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ーで指定されます。

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 拡張プロパティーは、構成ディレクトリへのパスを提供します。プロパティー自身は 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 は拡張プロパティーであるため、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。

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

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

Start 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。

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

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。

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

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

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 つのプロパティーがあり、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。

```
# Get the value for retry count from the RTR file.
RETRY_CNT=‘scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘
# Get the value for retry interval from the RTR file. This value is in seconds
# and must be converted to minutes for passing to pmfadm. Note that the
```

```
# conversion rounds up; for example, 50 seconds rounds up to 1 minute.
((RETRY_INTRVAL=‘scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘ / 60))

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

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

Start 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 プロパティー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 プロパティーを検査し、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 プロパティーを設定していないため、このプロパティーはデフォルト値 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。同じノード上での再起動や、別のノード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 リソースを含むリソースグループがクラスタノード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であるとき、RGM はそのノード上で 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、314 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くても、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。

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

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

```
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           “HA-DNS is not started”

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

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

Stop 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 プロパティー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 プロパティーを検査し、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 プロパティーを設定していないため、このプロパティーはデフォルト値 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、317 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ー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に再配置することを要求します。

プロパティー値の取得

このプログラムには、次のプロパティーの値が必要です。

- 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 ユーティリティー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、上記プロパティーの値を取得し、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ー (Thorough_probe_intervalなど) の場合、scha_resource_get() 関数は値だけを戻します。拡張プロパティー (Probe_timeout など) の場合、scha_resource_get() 関数はタイプと値を戻します。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
probefail=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、probefail 変数を設定します。
- probefail 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るのでは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
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep
   # for a duration of THOROUGH_PROBE_INTERVAL.
   sleep $PROBE_INTERVAL

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

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

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

```
   fi
fi
```

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再起動するか、RGM がデータサービスを別のノード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、317 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 ユーティリティーを使用すると、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ーは、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ーのデフォルト値は、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え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にフェイルオーバーします。

```
function restart_service
{

        # To restart the data service, first verify that the
        # data service itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
```

```
                # Since the TAG for the data service is still registered under
                # PMF, first stop the data service and start it back up again.

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

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

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

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


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

        return 0
}
```

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 プログラムはエラー状態で終了します。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、323 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 プロパティーの値を取得し、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ます。

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

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

Monitor_stop 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、325 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します。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 がマスクできる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。

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

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

Monitor_stop 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

データサービスが含まれるリソースグループを PROBE メソッドが別のノード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。

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、327 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 拡張プロパティーが 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。

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

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

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

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

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

ノード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。

プロパティー更新の処理

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ー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

リソースの作成時や、管理アクションによるリソースまたはリソースグループのプロパティーの更新時に、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ーを変更したときだけです。RGM がプロパティーを設定したときや、モニターがリソースプロパティー 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ーと値が含まれます。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、単一のプロパティーである Confdir 拡張プロパティーを確認します。このプロパティーは、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 プロパティーは 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 プロパティー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 に渡すプロパティーの 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 プロパティー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ー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 メソッドやその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。

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

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Name of the DNS resource.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Name of the resource group in which the resource is
                  # configured.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
         T)
                  # Name of the resource type.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
         r)
                  # The method is not accessing any system defined
                  # properties so this is a no-op
                  ;;
         g)
                  # The method is not accessing any resource group
                  # properties, so this is a no-op
                  ;;
         c)
                  # Indicates the Validate method is being called while
                  # creating the resource, so this flag is a no-op.
                  ;;
         u)
                  # Indicates the updating of a property when the
                  # resource already exists. If the update is to the
                  # Confdir property then Confdir should appear in the
                  # command-line arguments. If it does not, the method must
                  # look for it specifically using scha_resource_get.
                  UPDATE_PROPERTY=1
                  ;;
         x)
                  # Extension property list. Separate the property and
                  # value pairs using "=" as the separator.
                  PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                  VAL=`echo $OPTARG | awk -F= '{print $2}'`
```

```
                # If the Confdir extension property is found on the
                # command line, note its value.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
   done
}
```

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ーを示す)、`g` フラグ (リソースグループプロパティーを示す)、`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ー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ーの名前と値を取得します。更新されているプロパティーの中に `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` がプロパティー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` 拡張プロパティー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 プロパティー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

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

注 – 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、プロパティーがコマンド行上で「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。

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

Confdir プロパティー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。

```
# Check that the named.conf file is present in the Confdir directory
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
```

```
            -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
         exit 1
fi
```

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。

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

exit 0
```

Validate 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ーが作成されず、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

プロパティーが変更された場合、RGM は Update メソッドを実行して、そのことを動作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ー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ーを更新しません。プロパティーの更新は 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、プロパティー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したがって、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。このメソッドは、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

# Kill the monitor that is running already
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
                 "${ARGV0} Could not stop the monitor"
       exit 1
    else
    # could successfully stop DNS. Log a message.
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                 "Monitor for HA-DNS successfully stopped"
    fi
```

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 プロパティー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。

◆ ◆ ◆ 第 6 章

6 データサービス開発ライブラリ

この章では、データサービス開発ライブラリ (Data Service 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。

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ーの問題に、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。

構成プロパティーの管理

すべてのコールバックメソッドは構成プロパティーにアクセスする必要があります。

DSDL は、次の手段により、プロパティーへのアクセスをサポートします。

- 環境の初期化
- プロパティー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 関数が使用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ー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ー値を解析します。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ー、および、よく使用される拡張プロパティー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、プロパティー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ーに対応します。関数の戻り値のタイプは取得するプロパティー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ーの取得、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ー用の 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ー (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 (Process Monitor Facility、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備えているため、障害モニターを実装する際の複雑さを大幅に軽減します。リソースがノード上で起動すると、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 プロパティーを使用して、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。この関数は 、Retry_interval プロパティーの期間中に、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 プロパティーを上回った場合、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() ユーティリティーは、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、各クラスタノード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「サンプル DSDL リソースタイプの実装」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

HAStoragePlus 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、アフィニティー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もあります。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。HAStoragePlus 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。

◆ ◆ ◆ 第 7 章

7 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データサービス開発ライブラリ (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ー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ー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ーを取り出し、これによって、そのリソースは、`scds_get`*name*`()` 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。

リソースタイプ登録ファイル

リソースタイプ登録 (Resource Type Registration、RTR) 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ー
- これらのプロパティー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ーのさまざまな設定

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。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ーが必要な場合は、そのプロパティーをリソースタイプ実装の RTR ファイルで拡張プロパティーとして宣言します。新しいプロパティーのアクセスには DSDL `scds_get_ext_property()` ユーティリティーを使用します。

`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ー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リソースグループマネージャー (Resource Group Manager、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ー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に対して呼び出されます。ノード群は、リソースタイププロパティー `Init_nodes` の値で定義されます。`Init_nodes` が `RG_PRIMARIES` 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 (通常は、クラスタのすべてのノード) に対して呼び出されます。

`Init_nodes` のデフォルト値は `RG_PRIMARIES` 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ー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ーを解析します。これによって、リソースプロパティーの新しい値を、そのリソースが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ーの新しい値は、コマンド行から読み込まれます。残りのプロパティー (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ー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ー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ーを検査します。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「サンプル DSDL リソースタイプの実装」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);   /* Invalid resource property setting */
   }
   return (0);   /* Acceptable setting */
}
```

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ーの取得、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()` ユーティリティー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);   /* Initialization Error */
   }
   if (svc_validate(handle) != 0) {
   return (1);   /* Invalid settings */
   }
   if (svc_start(handle) != 0) {
   return (1);   /* Start failed */
   }
   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のシステムに関してなん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() が十分に保守的であり、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() ユーティリティーを呼び出して、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ーは、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。詳細は、pmfadm(1M)のマニュアルページにある -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、クラスタノード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、そのノードで Start メソッドが正常に終了していなくても、RGM によって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 プロパティー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、221 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);   /* Initialization Error */
}
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、このノードでこのアプリケーションを停止すべきだからです。

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で使用可能かどうかが確認されます。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ー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ー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、どのプロパティー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ーに 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ーは、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ー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ー 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 管理コマンドやクラスタ管理 GUI で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す。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 (;;) {
   /* sleep for a duration of thorough_probe_interval between
   *  successive probes.
   */
   (void) scds_fm_sleep(scds_handle,
```

```
scds_get_rs_thorough_probe_interval(scds_handle));
/* Now probe all ipaddress we use. Loop over
* 1. All net resources we use.
* 2. All ipaddresses in a given resource.
* For each of the ipaddress that is probed,
* compute the failure history.
*/
probe_result = 0;
/* Iterate through the all resources to get each
* IP address to use for calling svc_probe()
*/
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
/* Grab the hostname and port on which the
* health has to be monitored.
*/
hostname = netaddr->netaddrs[ip].hostname;
port = netaddr->netaddrs[ip].port_proto.port;
/*
* HA-XFS supports only one port and
* hence obtaint the port value from the
* first entry in the array of ports.
*/
ht1 = gethrtime();
/* Latch probe start time */
probe_result = svc_probe(scds_handle, hostname, port, timeout);
/*
* Update service probe history,
* take action if necessary.
* Latch probe end time.
*/
ht2 = gethrtime();
/* Convert to milliseconds */
dt = (ulong_t)((ht2 - ht1) / 1e6);
/*
* Compute failure history and take
* action if needed
*/
(void) scds_fm_action(scds_handle,
probe_result, (long)dt);
}       /* Each net resource */
}       /* Keep probing forever */
```

◆ ◆ ◆ 第 8 章

サンプル DSDL リソースタイプの実装

この章では、データサービス開発ライブラリ (Data Service Development Library、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。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は、付録 C「サンプル DSDL リソースタイプのコード例」に記載されています。

この章の内容は次のとおりです。

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/xfs 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。xfs 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/xfs -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 プロパティーを使用して、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

xfs 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、xfs 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 プロパティーを使用すると、デフォルト値を設定したり、クラスタ管理者が xfs 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、このプロパティーのデフォルト値を 7100/tcp と定義します。SUNW.xfnts の Start メソッドで、Port_list を xfs 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 プロパティー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ーについて説明します。各プロパティーの目的については説明しません。プロパティーの詳細については、34 ページの「リソースとリソースタイププロパティーの設定」を参照してください。

次に示すように、Confdir_list 拡張プロパティー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 プロパティーには、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 プロパティーは、アプリケーションが待機するポートを特定します。

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

このプロパティー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 関数が使用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上でオンラインになったとき、またはリソースが有効になったとき、RGM はそのノード上で 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上で xfs 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、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。

```
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;

   /* 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);
   }
```

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。

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

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。

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

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

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

svc_wait() 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。

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

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

svc_wait() 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 プロパティーで指定された期限内にリソースの Retry_count プロパティーの値を超えた場合、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 {
      /*
       * probe the data service on the IP address of the
       * network resource and the portname
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Success. Free up resources and return */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

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

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

注 – 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); /* Successfully stopped */
```

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

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 プロパティーの値を示します。

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

リソースがノード上で起動したあと、RGM はそのノード上で 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));

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

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

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

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

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); /* Successfully stopped monitor */
}
```

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 プロパティーの値を示します。

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

リソースが属するリソースグループを障害モニターが別のノード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。

```
/* Process the arguments passed by RGM and initialize syslog */
    if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
```

```
        scds_syslog(LOG_ERR, "Failed to initialize the handle.");
        return (1);
    }

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

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

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

SUNW.xfnts 障害モニター

リソースがノード上で起動したあと、RGM は、PROBE メソッドを直接呼び出すのではなく、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ーを使用して、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 プロパティーを使用して) 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、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。

xfonts_probe のメインループ

xfonts_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、xfonts_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。

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

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

- 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 (;;) {
      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));
```

xfnts_probe 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++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS supports only one port and
```

```
             * hence obtain the port value from the
             * first entry in the array of ports.
             */
            ht1 = gethrtime(); /* Latch probe start time */
            scds_syslog(LOG_INFO, "Probing the service on port: %d.", port);

            probe_result =
            svc_probe(scds_handle, hostname, port, timeout);

            /*
             * Update service probe history,
             * take action if necessary.
             * Latch probe end time.
             */
            ht2 = gethrtime();

            /* Convert to milliseconds */
            dt = (ulong_t)((ht2 - ht1) / 1e6);

            /*
             * Compute failure history and take
             * action if needed
             */
            (void) scds_fm_action(scds_handle,
                probe_result, (long)dt);
        }   /* Each net resource */
    }    /* Keep probing forever */
```

svc_probe() 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 プロパティーの値の期間中に、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 プロパティーを上回った場合、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。

次回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ー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ーを変更したときや、モニターが Status と Status_msg リソースプロパティーを設定したときだけです。RGM がプロパティーを設定する場合、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ーと値が含まれます。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);

/* Return error if there is no confdir_list extension property */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }
```

- Confdir_list 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) {
      /*
       * 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);
   }
```

- サーバーデーモンバイナリがクラスタノード上でアクセスできるかどうか。

```
if (stat("/usr/openwin/bin/xfs", &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); /* Validation Failure */
   }
```

```
#ifdef TEST
   if (portlist->num_ports != 1) {
      scds_syslog(LOG_ERR,
          "Property Port_list must have only one value.");
      scds_free_port_list(portlist);
      return (1); /* Validation Failure */
   }
#endif
```

- 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); /* 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;
   }
```

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); /* return result of validation */
```

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ー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ーは、障害モニターに関連したものだけです。したがって、プロパティーが更新されたとき、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。

```
/* check if the Fault monitor is already running and if so stop
 * and restart it. The second parameter to scds_pmf_restart_fm()
 * uniquely identifies the instance of the fault monitor that needs
 * to be restarted.
 */

scds_syslog(LOG_INFO, "Restarting the fault monitor.");
result = scds_pmf_restart_fm(scds_handle, 0);
if (result != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "Failed to restart fault monitor.");
   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);
   return (1);
}

scds_syslog(LOG_INFO,
"Completed successfully.");
```

注 – 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。

◆ ◆ ◆ 第 9 章

9 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、リソースグループマネージャー (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。

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ー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ースクリプト。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ースクリプト。C または Korn シェルのソースソースを生成する場合は RTR ファイルを含む Solaris パッケージとユーティリティースクリプト。

Agent Builder を使って、プロセス監視機能 (Process Monitor Facility、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す。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。

この節の内容は、次のとおりです。

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す。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 がエクスポーズ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、176 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 cc から $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用ディレクトリなど) 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「汎用データサービス」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。この画面では、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します。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。

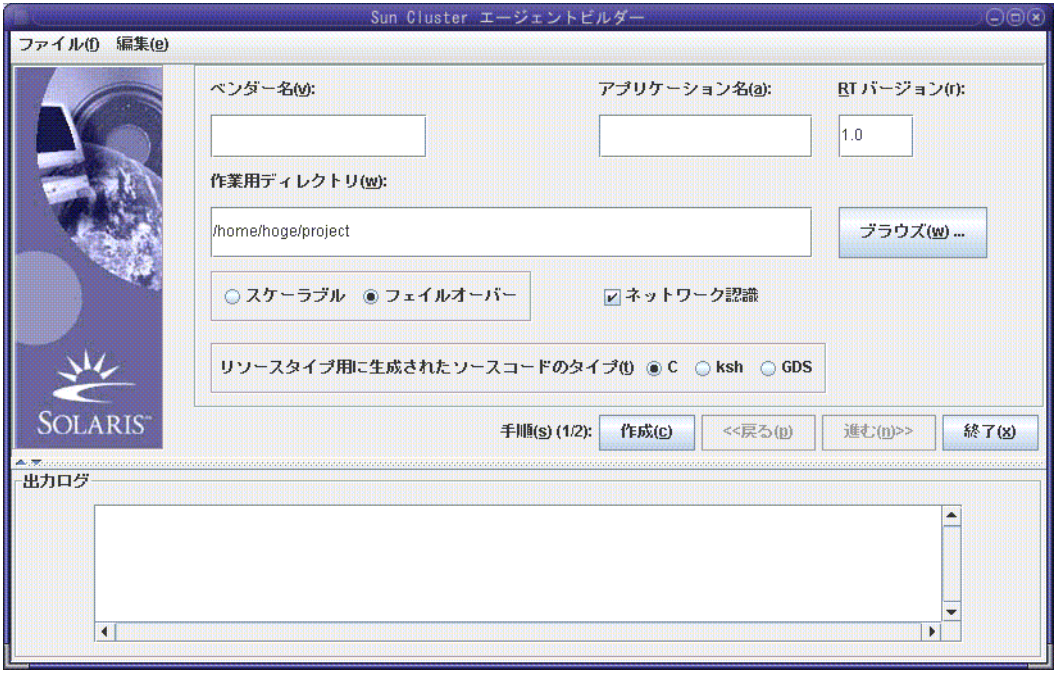


図 9–1 Agent Builder の「作成」画面

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、「スケーラブル」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ので、「進む」ボタンもグレー表示されます。

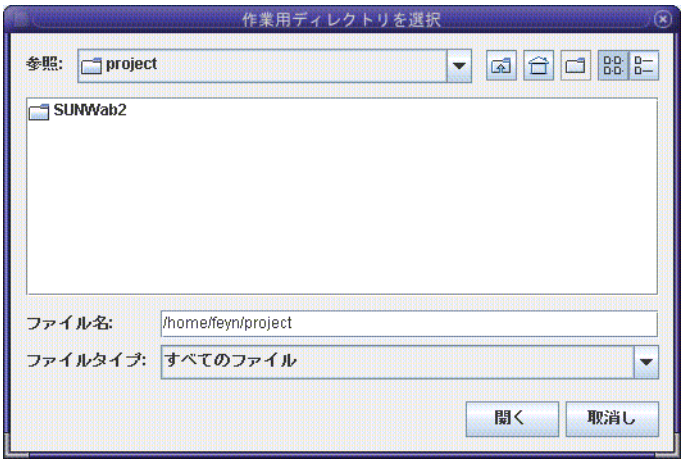
Agent Builder は、画面の下にある「出力ログ」領域に進捗メッセージを表示します。作業が終了したとき、Agent Builder は成功メッセージまたは警告メッセージを表示します。「進む」が強調表示されます。あるいは、これが最後の画面の場合は、「キャンセル」だけが強調表示されます。

「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。

アイコン	目的
	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。184 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 が提供するブラウズ画面を使用すると、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。

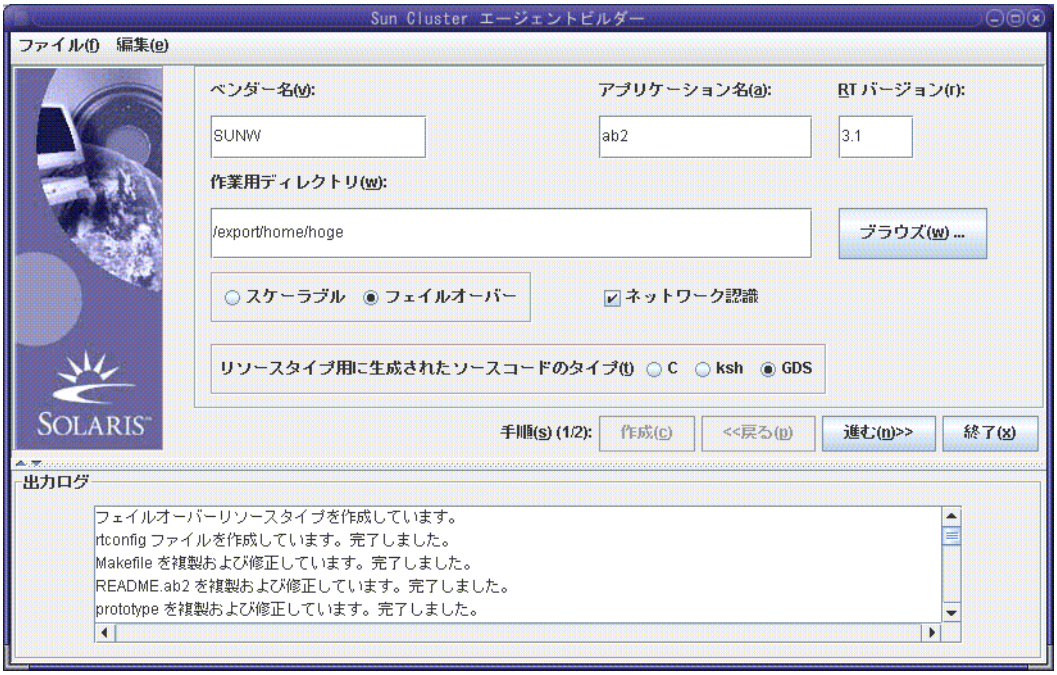


図 9–2 Agent Builder の「作成」画面 (情報の入力後)

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を識別する名前。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」。リソースタイプの名前。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。184 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「汎用データサービス」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結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。184 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。

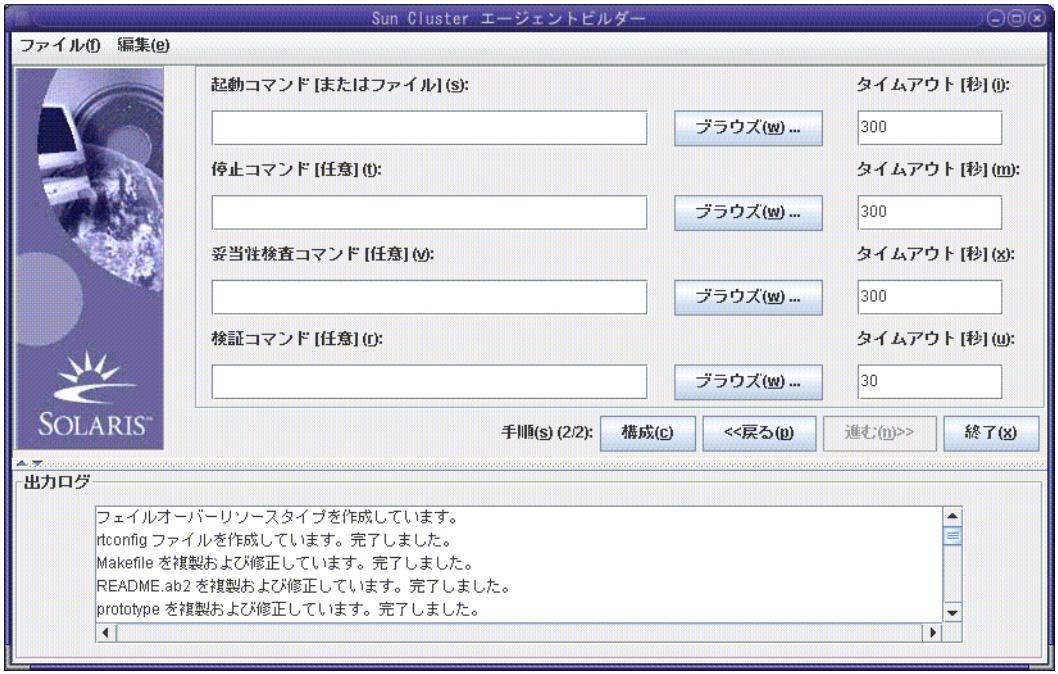


図 9–3 Agent Builder の「構成」画面

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、182 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 `$hostnames` 変数を使用できます。詳細は、181 ページの「Agent Builder の Korn シェルベース `$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 ("") で囲んではなりません。

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プロセス監視機能 (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、182 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 `$hostnames` 変数を使用できます。詳細は、181 ページの「Agent Builder の Korn シェルベース `$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」。定期的に実行され、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) の範囲の終了状態を戻すコマンド。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、「ブラウズ」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、プロパティー変数を指定することもできます。この変数については、182 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 `$hostnames` 変数を使用できます。詳細は、181 ページの「Agent Builder の Korn シェルベース `$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 ("") で囲んではなりません。

- 「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す。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 – `$hostnames` 変数は、Korn シェルベースのサービスでのみサポートされます。`$hostnames` 変数は、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ーで構成されます。

Network_resources_used プロパティー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。

プロパティー変数の使用

プロパティー変数を使用すれば、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ーの値を 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ー変数がないかをチェックし、プロパティー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 – プロパティー変数は、Korn シェルベースのサービスではサポートされません。

プロパティー変数のリスト

この節では、使用できるプロパティー変数を示します。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ーについては、付録 A 「標準プロパティー」を参照してください。

リソースプロパティー変数

- 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ー変数

- 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ー変数

- 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

プロパティー変数の構文

プロパティー変数を指定する場合は、プロパティー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ー変数を解釈し、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 によるプロパティー変数の置き換え

Agent Builder では、プロパティー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、コンマ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、プロパティーの妥当性検査を追加するコードや、Agent Builder がエクスポーズ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 vvvvvvvvvvvvvvv */
/* 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ー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ーの設定」を参照してください。プロパティーについては、付録 A「標準プロパティー」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ーを持たせるため、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。
   プロパティー変数を指定できます。プロパティー変数については、182 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. 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。

ディレクトリ名	内容
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には、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。189 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。

この節の内容は、次のとおりです。

ソースファイルとバイナリファイル

リソースグループマネージャー (Resource Group Manager、RGM) は、リソースグループと、最終的にはクラスタ上のリソースを管理します。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 (ノードの障害など) が発生したとき、RGM は、当該ノード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、当該ノード上で動作しているリソースを停止します。次に、Start メソッドを呼び出して、別のノード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「リソースグループマネージャーモデル」、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、次のファイルを生成します。

- 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ープログラム `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。185 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ー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ースクリプトに加え、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、各ユーティリティー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、ユーティリティー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、リソースタイプ登録 (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ー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と、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。184 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。このモジュールを使用することで、Sun Java Studio 製品で Sun Cluster ソフトウェアのリソースタイプを作成することができます。

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://wwws.sun.com/software/sundev/jde/documentation/index.html の Web サイトで参照できます。

▼ 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。

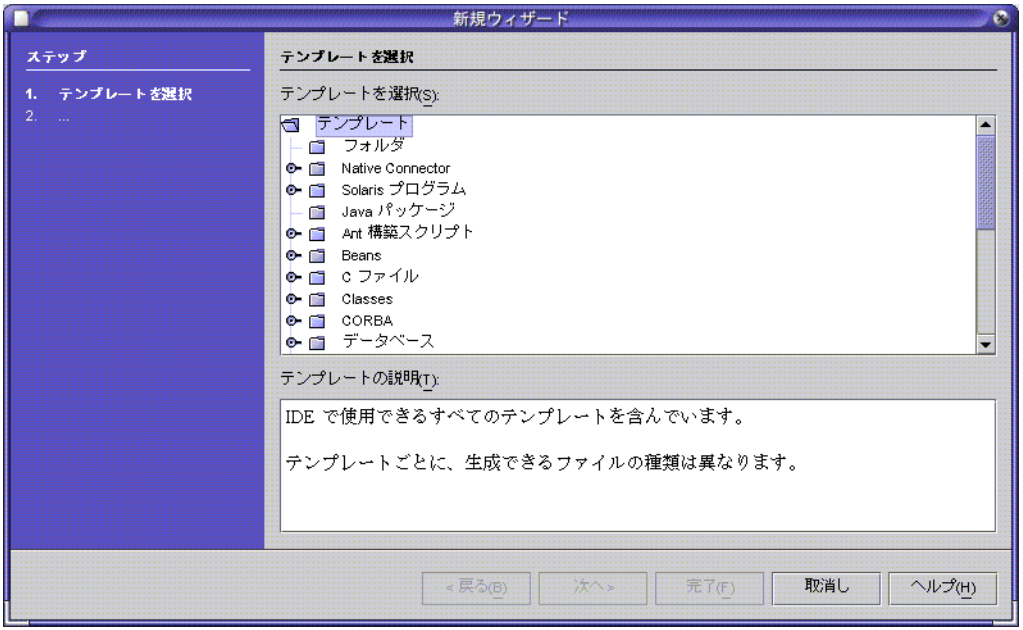
▼ Cluster Agent モジュールを起動する方法

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。

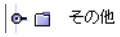
注 – Sun Java Studio 製品の設定、インストール、使用の詳細は、Sun Java Studio マニュアルに記載されています。このマニュアルは
、http://wwws.sun.com/software/sundev/jde/documentation/index.html の Web サイトで参照できます。

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。



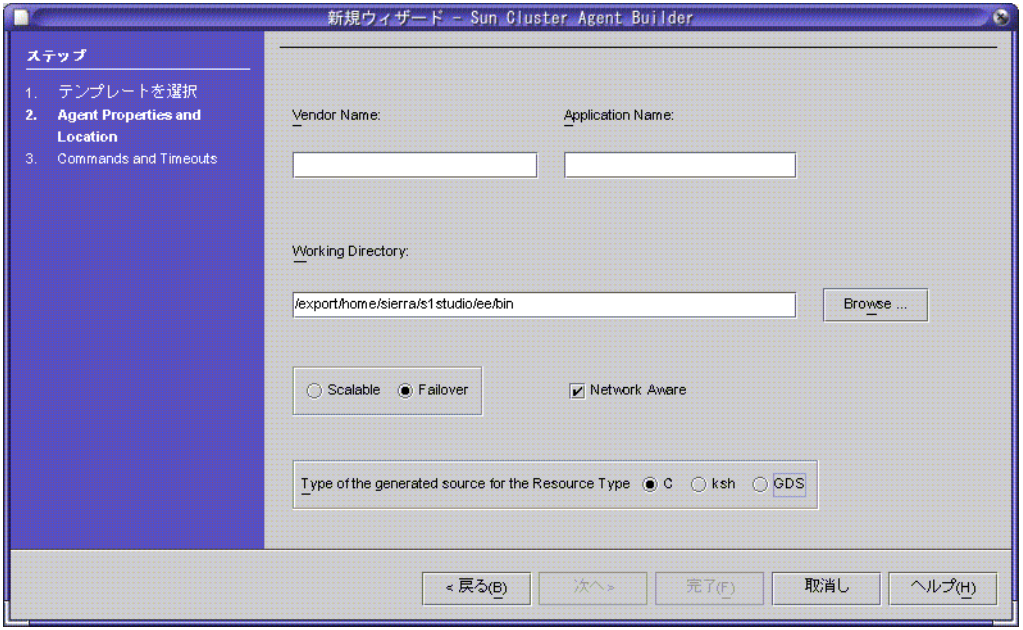
2 「テンプレートを選択」区画で、必要に応じて下方向へスクロールし、「その他」フォルダの横に表示されている鍵マークをクリックします。



「その他」フォルダが開きます。

3 「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。

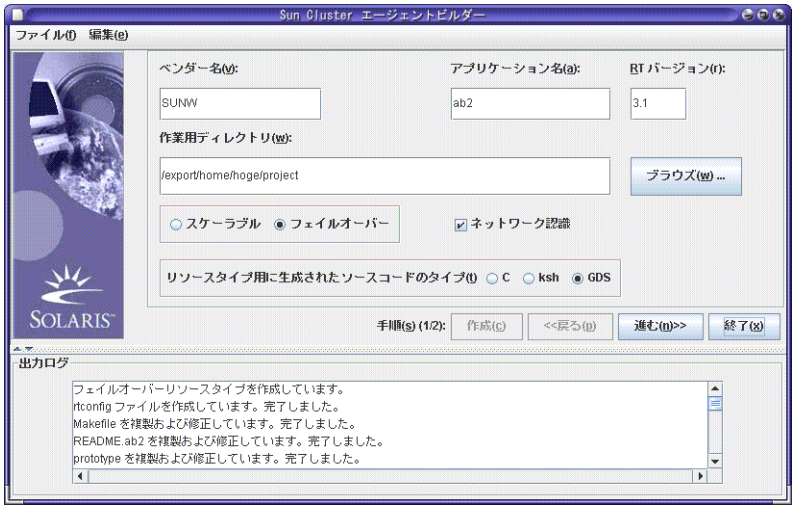


図 9–4 英語版 Agent Builder ソフトウェアの作成画面

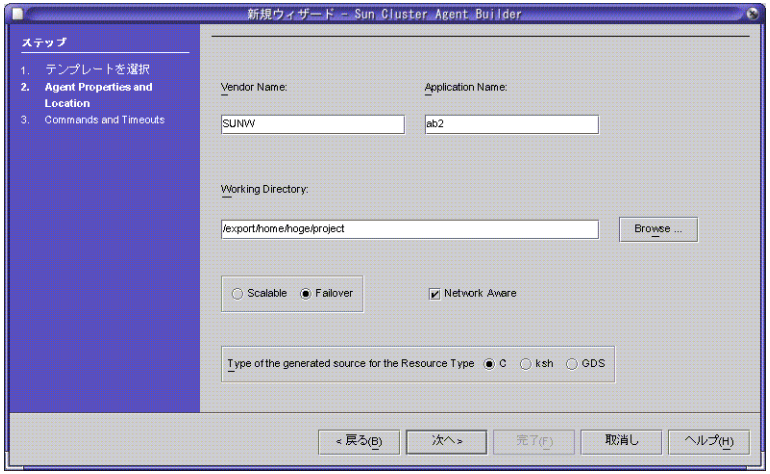


図 9–5 Cluster Agent モジュールの「新規ウィザード - Sun Cluster Agent Builder」画面

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。

◆ ◆ ◆ 第 10 章

10 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。

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、スケー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ーラビリティー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

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 ユ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ーを追加する場合や、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)`を参照してください。 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。例として、211 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212 ページの「Sun Cluster 管理コマンドを使って GDS ベースのスケー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ー変数が含まれます。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` プロパティーを使用します (詳細は202 ページの「`Log_level` プロパティー」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 プロパティー

この節では、必須 GDS プロパティーについて説明します。

Port_list プロパティー

Port_list プロパティーは、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 プロパティーは、Agent Builder によって作成される起動スクリプト内、または clresource コマンドで指定する必要があります。

このプロパティーの指定が必要かどうかは、アプリケーションがネットワーク対応かどうかによって決まります。アプリケーションがネットワーク対応であると指定した場合、つまり Network_aware プロパティーをデフォルトの TRUE に設定した場合は、Start_command 拡張プロパティーと Port_list プロパティー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対応ではないと指定した場合、つまり Network_aware プロパティーを FALSE に設定した場合は、Start_command 拡張プロパティーのみを指定します。Port_list プロパティーの指定は任意です。

Start_command プロパティー

Start_command 拡張プロパティー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。

アプリケーションがネットワーク対応の場合は、Start_command 拡張プロパティーと Port_list プロパティー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 拡張プロパティーだけを指定します。

任意の GDS プロパティー

任意の GDS プロパティーには、「システム定義プロパティー」と「拡張プロパティー」の両方が含まれます。システム定義プロパティーは、Sun Cluster により提供されるプロパティーの標準セットです。RTR ファイルで定義されているプロパティーは、拡張プロパティーと呼ばれます。

任意の GDS プロパティーには次のものがあります。

- Child_mon_level 拡張プロパティー (管理コマンドでのみ使用)
- Failover_enabled 拡張プロパティー
- Log_level 拡張プロパティー
- Network_aware 拡張プロパティー
- Network_resources_used プロパティー
- Probe_command 拡張プロパティー
- Probe_timeout 拡張プロパティー
- Start_timeout プロパティー
- Stop_command 拡張プロパティー
- Stop_signal 拡張プロパティー
- Stop_timeout プロパティー
- Validate_command 拡張プロパティー
- Validate_timeout プロパティー

Child_mon_level プロパティー

注 – Sun Cluster 管理コマンドを使用する場合は、Child_mon_level プロパティーを使用できます。Agent Builder を使用する場合、このプロパティーは使用できません。

このプロパティーは、プロセスモニター機能 (Process Monitor Facility、PMF) を通じて監視されるプロセスを制御します。このプロパティーは、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。このプロパティーは、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。

このプロパティーを省略するか、このプロパティーにデフォルト値の -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 プロパティー

この拡張プロパティーは、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ーに 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。

このプロパティーに 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へのフェイルオーバーは行われません。

このプロパティーを使用すると、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。このプロパティーのデフォルト値は 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 拡張プロパティーの代わりに Failover_mode プロパティーを使用します。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 プロパティー

このプロパティーは、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。このプロパティーには、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_aware プロパティー

このプロパティーでは、アプリケーションがネットワークを使用するかどうかを指定します。デフォルトでは、GDS はアプリケーションがネットワーク対応である、つまりネットワークを使用すると見なします (Network_aware の設定は TRUE)。

アプリケーションがネットワーク対応の場合は、Start_command 拡張プロパティーと Port_list プロパティー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 拡張プロパティーだけを指定します。

Network_resources_used プロパティー

このプロパティーは、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。このプロパティーのデフォルト値は、空のリストです。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、このプロパティーを指定してください。このプロパティーを省略するか 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 プロパティー

このプロパティーは、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 をとることがあります。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 プロパティーや 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 プロパティー

このプロパティーは、検証コマンドのタイムアウト値を指定します。詳細は、203 ページの「Probe_command プロパティー」を参照してください。Probe_timeout のデフォルトは 30 秒です。

Start_timeout プロパティー

このプロパティーは、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 プロパティー」を参照してください。Start_timeout のデフォルトは 300 秒です。

Stop_command プロパティー

このプロパティーは、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 拡張プロパティー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 プロパティー

このプロパティーは、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。指定できる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 プロパティー

このプロパティーは、停止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「Stop_command プロパティー」を参照してください。Stop_timeout のデフォルトは 300 秒です。

Validate_command プロパティー

このプロパティーは、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 プロパティー

このプロパティーは、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 プロパティー」 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 の詳細は、第 9 章「Sun Cluster Agent Builder」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

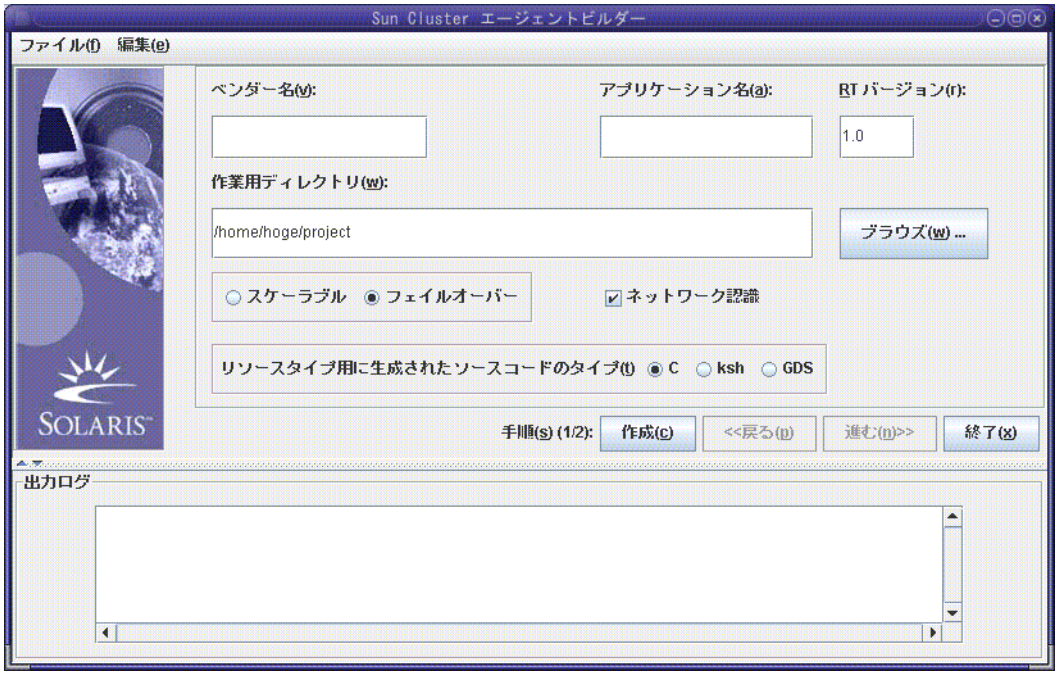
▼ Agent Builder を起動し、スクリプトを作成する

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ー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。

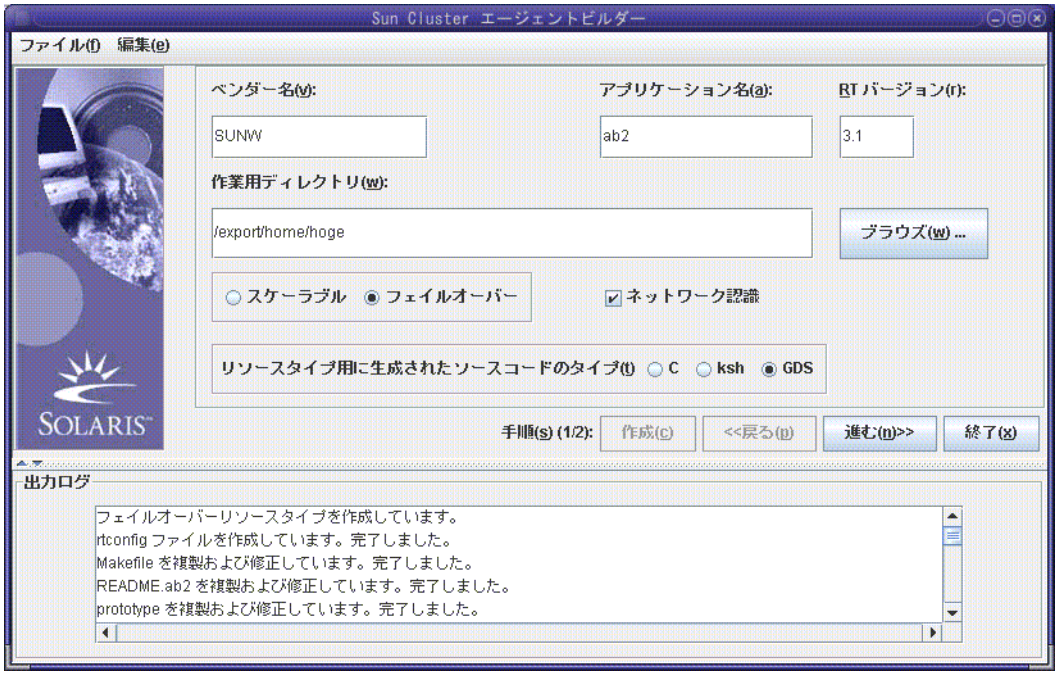
9 **(**省略可能**)** 表示されているデフォルト値から **RT** バージョンを変更します。

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。

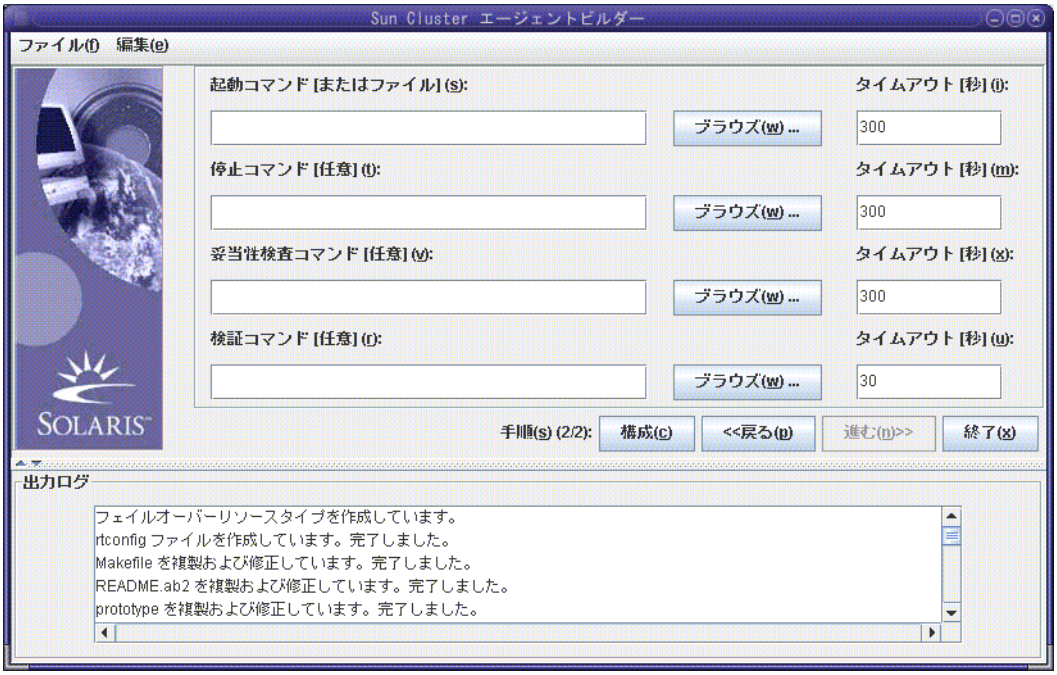
Agent Builder により、スクリプトが作成されます。結果が「出力ログ」領域に表示されます。



「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、182 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、182 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、182 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、182 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ー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ほか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 administration コマンドを使用して、たとえば、`clresourcetype` および、`clresourcegroupclresource`、GDS を維持および管理できます。

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 ベースのスケー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 スケー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、205 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、182 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]
```

◆ ◆ ◆ 第 11 章

11 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。

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、拡張プロパティーの値を取得します。
- 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ー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ーの値を取得します。
- scds_free_ext_property(3HA) – scds_get_ext_property() によって割り当てられたメモリーを解放します。

次の関数は、リソースによって使用される SUNW.HAStoragePlus 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.HAStoragePlus 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.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。詳細は、SUNW.HAStoragePlus(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_resource(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。

プロパティー関数

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ーも含む) に固有のプロパティー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。ライブラリは、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ー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 などのプロパティーを照会しなくても、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。100%!!!You typically use this function for debugging.
- 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) – ポートとプロトコルのリスト内容を 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、プロセス監視機能 (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定します。
- 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_start(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 プロパティー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) – デバッギングメッセージをシステムログに書き込みます。

◆ ◆ ◆ 第 12 章

12 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スケーラブル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。このリソースタイプの実装については、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。

CRNP の概念

CRNP は、標準の 7 層開放型システム間相互接続 (Open System Interconnect、OSI) 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、グローバルクラスタ投票ノード上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、`sysevntd` 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。

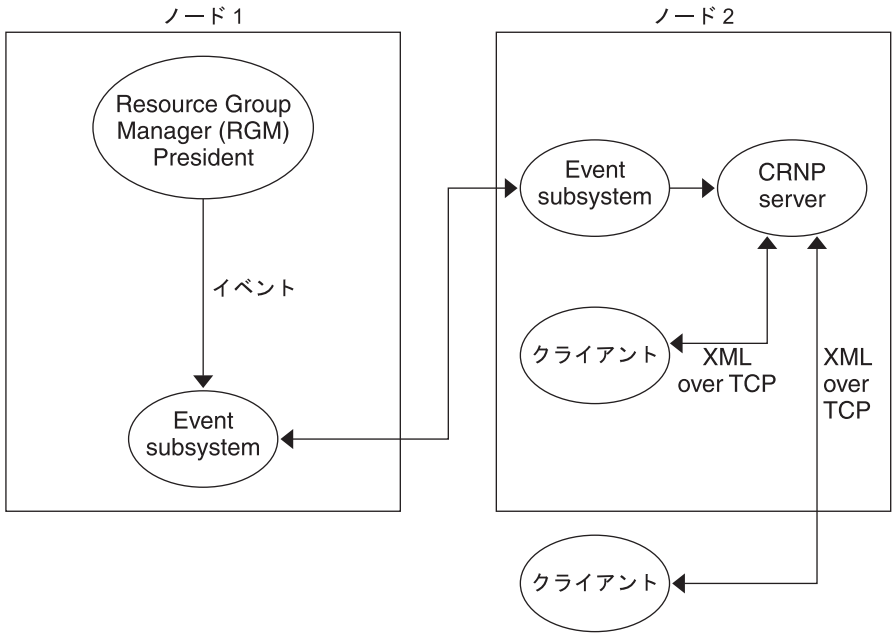


図 12–1 CRNP コンポーネント間のイベントの流れ

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。

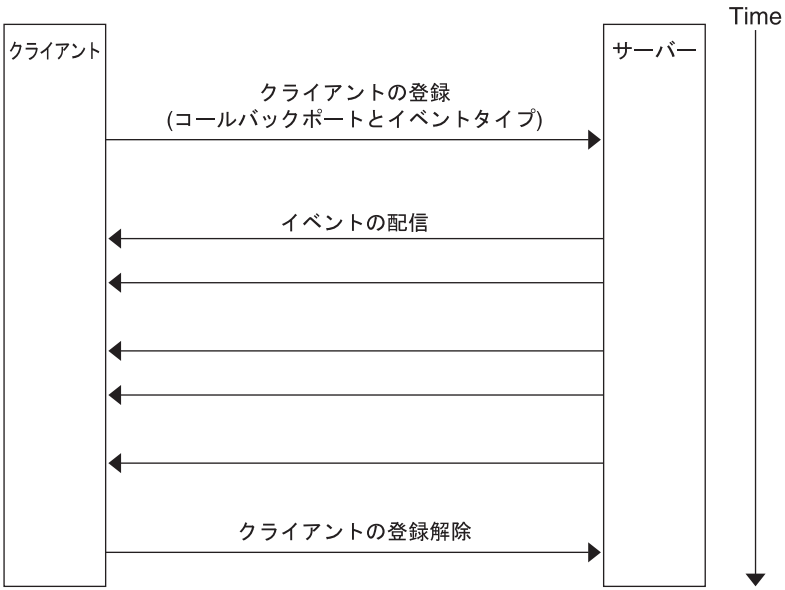


図 12–2 クライアントとサーバー間の通信の流れ

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	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。 ■ プロトコルバージョン ■ 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) 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。 ■ イベントクラス ■ イベントサブクラス (省略可能) ■ 名前と値がペアになったリスト (省略可能) 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ドキュメントタイプ定義 (Document Type Definition、DTD) は、SC_CALLBACK_REG です。この DTD の詳細は、付録 F 「CRNP のドキュメントタイプ定義」を参照してください。
SC_REPLY	このメッセージには次の情報が含まれます。 ■ プロトコルバージョン ■ エラーコード ■ エラーメッセージ: SC_REPLY のクラスを生成する DTD は SC_REPLY です。この DTD の詳細は、付録 F 「CRNP のドキュメントタイプ定義」を参照してください。
SC_EVENT	このメッセージには次の情報が含まれます。 ■ プロトコルバージョン ■ イベントクラス ■ イベントサブクラス ■ ベンダー ■ パブリッシャー ■ 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) ■ 名前 (文字列) ■ 値 (文字列または文字列配列) 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、付録 F 「CRNP のドキュメントタイプ定義」を参照してください。

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
EC_Cluster ESC_cluster_membership	必須: なし (省略可能) 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
EC_Cluster ESC_cluster_rg_state	次の条件で 1 つ必要: rg_*name* 値のタイプ: 文字列 (省略可能) なし	*rg_name* (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

クラスとサブクラス	名前と値のペア	説明
EC_Cluster ESC_cluster_r_state	次の条件で 1 つ必要: r_*name* 値のタイプ: 文字列 (省略可能) なし	*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
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、374 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も、登録が失敗することも、あるいは登録処理が行われない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る)。
- 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: node_list
ESC_cluster_membership	ベンダー: SUNW	値のタイプ: 文字配列
		名前: state_list
		state_list 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: rg_name
ESC_cluster_rg_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、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

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/webservices/jaxp/ で説明しています。
- この例は、付録 G 「CrnpClient.java アプリケーション」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 「CrnpClient.java アプリケーション」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 「CrnpClient.java アプリケーション」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/webservices/jaxp/ 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-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.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/xsltc.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 「CrnpClient.java アプリケーション」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();
                                // Construct event from the sock stream and process it
                                sock.close();
                        }
                        // UNREACHABLE

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* private member variables */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2 `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/webservices/jaxp/ 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) {
                // Parser with specified options can't be built
                pce.printStackTrace();
                System.exit(1);
        }

        // Create the root element
        Element root = (Element) document.createElement("SC_CALLBACK_REG");

        // Add the attributes
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Add the events
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(document));
        }
        document.appendChild(root);

        // Convert the whole thing to a string
        DOMSource domSource = new DOMSource(document);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
                Transformer transformer = tf.newTransformer();
                transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
                System.out.println(e.toString());
                return ("");
        }
        return (strWrite.toString());
}
```

```
        private String port;
        private String regType;
        private Vector regEvents;
}
```

2 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、240 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 「CrnpClient.java アプリケーション」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);

        // add the events
        for (int i = 4; i < regs.length; i++) {
                if (regs[i].equals("M")) {
                        cbReg.addRegEvent(createMembershipEvent());
                } else if (regs[i].equals("A")) {
                        cbReg.addRegEvent(createAllEvent());
                } else if (regs[i].substring(0,2).equals("RG")) {
                        cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
                } else if (regs[i].substring(0,1).equals("R")) {
                        cbReg.addRegEvent(createREvent(regs[i].substring(2)));
                }
        }

        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}

private Event createAllEvent()
{
        Event allEvent = new Event();
        allEvent.setClass("EC_Cluster");
        return (allEvent);
}

private Event createMembershipEvent()
```

```
{
        Event membershipEvent = new Event();
        membershipEvent.setClass("EC_Cluster");
        membershipEvent.setSubclass("ESC_cluster_membership");
        return (membershipEvent);
}

private Event createRgEvent(String rgname)
{
        Event rgStateEvent = new Event();
        rgStateEvent.setClass("EC_Cluster");
        rgStateEvent.setSubclass("ESC_cluster_rg_state");

        NVPair rgNvpair = new NVPair();
        rgNvpair.setName("rg_name");
        rgNvpair.setValue(rgname);
        rgStateEvent.addNvpair(rgNvpair);

        return (rgStateEvent);
}

private Event createREvent(String rname)
{
        Event rStateEvent = new Event();
        rStateEvent.setClass("EC_Cluster");
        rStateEvent.setSubclass("ESC_cluster_r_state");

        NVPair rNvpair = new NVPair();
        rNvpair.setName("r_name");
        rNvpair.setValue(rname);
        rStateEvent.addNvpair(rNvpair);

        return (rStateEvent);
}
```

2 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ーを設定します。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/webservices/jaxp/ 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();

        // We don't need to bother validating
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // We want to ignore comments and whitespace
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Coalesce CDATA sections into TEXT nodes.
        dbf.setCoalescing(true);
}
```

▼ 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応答して 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ッサを提供します。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/webservices/jaxp/ 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
{
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();
        db.setErrorHandler(new DefaultHandler());
```

```
        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
}
```

2 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/webservices/jaxp/ 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;

                // Find the SC_REPLY element.
                nl = doc.getElementsByTagName("SC_REPLY");
                if (nl.getLength() != 1) {
                        System.out.println("Error in parsing: can't find "
                            + "SC_REPLY node.");
                        return;
```

```
                }

                n = nl.item(0);

                // Retrieve the value of the statusCode attribute
                statusCode = ((Element)n).getAttribute("STATUS_CODE");

                // Find the SC_STATUS_MSG element
                nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
                if (nl.getLength() != 1) {
                        System.out.println("Error in parsing: can't find "
                            + "SC_STATUS_MSG node.");
                        return;
                }
                // Get the TEXT section, if there is one.
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                // Not an error if there isn't one, so we just silently return.
                        return;
                }

                // Retrieve the value
                statusMsg = n.getNodeValue();
        }

        private String statusCode;
        private String statusMsg;
}
```

▼ 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、241 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、248 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;

        // Find the SC_EVENT element.
        nl = doc.getElementsByTagName("SC_EVENT");
        if (nl.getLength() != 1) {
           System.out.println("Error in parsing: can't find "
               + "SC_EVENT node.");
           return;
        }

        n = nl.item(0);

        //
        // Retrieve the values of the CLASS, SUBCLASS,
        // VENDOR and PUBLISHER attributes.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Retrieve all the nv pairs
        for (Node child = n.getFirstChild(); child != null;
            child = child.getNextSibling())
        {
               nvpairs.add(new NVPair((Element)child));
        }
}
```

```
        public String getRegClass()
        {
                return (regClass);
        }

        public String getSubclass()
        {
                return (regSubclass);
        }

        public String getVendor()
        {
                return (vendor);
        }

        public String getPublisher()
        {
                return (publisher);
        }

        public Vector getNvpairs()
        {
                return (nvpairs);
        }

        private String vendor, publisher;
```

2 **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;

                // Find the NAME element
                nl = elem.getElementsByTagName("NAME");
```

```
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "NAME node.");
                   return;
                }
                // Get the TEXT section
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                   System.out.println("Error in parsing: can't find "
                       + "TEXT section.");
                   return;
                }

                // Retrieve the value
                name = n.getNodeValue();

                // Now get the value element
                nl = elem.getElementsByTagName("VALUE");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "VALUE node.");
                   return;
                }
                // Get the TEXT section
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                System.out.println("Error in parsing: can't find "
                            + "TEXT section.");
                        return;
                }

                // Retrieve the value
                value = n.getNodeValue();
                }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3 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、239 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、**RBAC** 承認 solaris.cluster.modify 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 「CrnpClient.java アプリケーション」に一覧表示されています。

◆ ◆ ◆ 付 録 A

標準プロパティー

この付録では、標準のリソースタイプ、リソース、リソースグループ、リソースグループの各プロパティーについて説明します。また、システム定義プロパティーの変更および拡張プロパティーの作成に使用するリソースプロパティー属性についても説明します。

注 – 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ー名は大文字と小文字が区別されません。プロパティー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。

資源タイプのプロパティー

次の情報は、Sun Cluster ソフトウェアによる定義されたリソースタイププロパティーを示します。

プロパティー値は以下のように分類されます。

- 必須。プロパティーはリソースタイプ登録 (Resource Type Registration、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、プロパティー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。
- 条件付き。RTR ファイル内に宣言を必要とするプロパティーです。宣言がない場合、RGM はこのプロパティーを作成しません。したがって、このプロパティーを管理ユーティリティー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。プロパティーが 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ーです。宣言がない場合、RGM はこのプロパティーを作成しません。したがって、このプロパティーを管理ユーティリティー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ーです。プロパティーが 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。プロパティーが 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、プロパティーが 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ーは、管理ユーティリティー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。

以下にプロパティー名とその説明を示します。

注 – `API_version` や `Boot` などのリソースタイププロパティー名では、大文字と小文字が区別されません。プロパティー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
3.2 1/09	9

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 – このプロパティーを宣言しないか、このプロパティーをデフォルト値 (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)

Boot メソッドプログラムのパスを指定する任意のコールバックメソッド。RGM は、クラスタを結合または再結合するノード上で、このタイプの各管理対象リソースに対して Boot メソッドを実行します。

Boot、Init、Fini、または Validate メソッドが実行されるノードセットは、リソースタイプの Init_nodes プロパティーの設定によって決まります。Init_nodes プロパティーは、リソースタイプの Installed_nodes プロパティーで指定されているノードを示す RG_PRIMARIES に設定することができます。

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)

このプロパティーを 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ーは、次のように Scalable リソースのプロパティー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ー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。 スケーラブルリソースは、グローバルクラスタ非投票ノードで動作するように構成できます。ただし、同じ Solaris ホストの複数のグローバルクラスタ非投票ノードで動作するようにスケーラブルリソースを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複数マスターサービスを選択する場合に使用します。 このタイプのスケーラブルサービスはゾーン内で使用可能です。

詳細は、r_properties(5)のマニュアルページにある Scalable の説明、および『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)
Fini メソッドプログラムのパスを指定する任意のコールバックメソッド。RGM は、このタイプのリソースが RGM の管理対象外になったときに Fini メソッドを実行します。

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。

Boot、Init、 Fini、または Validate メソッドが実行されるノードセットは、リソースタイプの Init_nodes プロパティーの設定によって決まります。Init_nodes プロパティーは、リソースタイプの Installed_nodes プロパティーで指定されているノードを示す RG_PRIMARIES に設定することができます。

RGM は、次の状態が発生した場合、リソースが管理されなくなったノード上で 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、RGM はノードリストのすべてのノード上で 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、RGM はノードリストのすべてのノード上で 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が削除されます。この場合、RGM は削除されたノード上でのみ 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 プロパティーの設定に依存します。Init_nodes プロパティーは RG_primaries 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_primaries 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 (ブール型)
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 (つまり、ゾーンクラスタノードまたはグローバルクラスタ非投票ノードのいずれか) で実行されるかどうかを示すブール値。このプロパティーに 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。このプロパティーに 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 プロパティーに 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。このプロパティーに 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)
Init メソッドプログラムのパスを指定する任意のコールバックメソッド。RGM は、このタイプのリソースが RGM の管理対象になったときに Init メソッドを実行します。

Boot、Init、 Fini、または Validate メソッドが実行されるノードセットは、リソースタイプの Init_nodes プロパティーの設定によって決まります。Init_nodes プロパティーは、リソースタイプの Installed_nodes プロパティーで指定されているノードを示す RG_PRIMARIES に設定することができます。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

Init_nodes (enum)
RGM が Init、Fini、Boot、Validate の各メソッドをコールするノードを示します。指定できる値は、リソースをマスターできるノードのみを指定する RG_PRIMARIES、またはこのリソースタイプがインストールされるすべてのノードを指定する RT_INSTALLED_NODES のいずれかです。

カテゴリ: 任意

デフォルト: RG_PRIMARIES

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の名前のリスト。すべてのクラスタノードを明示的に含めるには、アスタリスク (*) を指定します。これがデフォルトです。

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

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。ノードに対するモニターチェックプログラムがゼロ以外の値で終了した場合、`GIVEOVER` タグ付きで `scha_control` を呼び出した結果としてそのノードにフェイルオーバーしようとしても実行できません。

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

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 (ブール型)
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ー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 はこのプロパティーを更新します。

カテゴリ: 照会のみ

デフォルト: 空のリスト

調整: NONE

Resource_type (文字配列型)
リソース型の名前です。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 つのプロパティー *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 (:) の区切り文字を挿入します。リソースタイプの名前の最後の部分、*rt-version* には、RT_version プロパティー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の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 (ブール型)
リソースタイプの RT_system プロパティーが 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*) . このプロパティーは、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 プロパティーはどのリソース型にも適用できます。

RT_system プロパティーが TRUE に設定されたリソース型を削除するには、まず、このプロパティーを 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 (文字配列型)
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ーは任意でした。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 (文字配列型)
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 (文字配列型)
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 (文字配列型)
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ー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 (文字配列型)
Validate メソッドプログラムのパスを指定する任意のコールバックメソッド。RGM は、このタイプのリソースのプロパティーの値を確認するために Validate メソッドを実行します。

Boot、Init、 Fini、または Validate メソッドが実行されるノードセットは、リソースタイプの Init_nodes プロパティーの設定によって決まります。Init_nodes プロパティーは、リソースタイプの Installed_nodes プロパティーで指定されているノードを示す RG_PRIMARIES に設定することができます。

カテゴリ: 条件付/明示

デフォルト: デフォルトなし

調整: NONE

`Vendor_ID` (文字配列型)
`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ー

この節では、Sun Cluster ソフトウェアで定義されているリソースプロパティーについて説明します。

プロパティー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ーを使ってリソースを作成するとき、必ず値を指定しなければなりません。
- 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。
- 条件付き。RGM は、RTR ファイル内にプロパティーが宣言されている場合にかぎりプロパティーを作成します。宣言されていない場合プロパティーは存在せず、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ーは、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。詳細については、各条件付きプロパティー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。

304 ページの「リソースプロパティーの属性」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ー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

以下にプロパティー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指定されたクライアント 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に送信されます。

このプロパティーは、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。

このプロパティーは、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ーは 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)。このプロパティーは 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、このプロパティーは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティーの 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

拡張プロパティー
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ー。リソースタイプの実装によって、これらのプロパティーを定義します。拡張プロパティーに設定可能な各属性については、304 ページの「リソースプロパティーの属性」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: 特定のプロパティーに依存

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 Prenet_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に再配置することを示します。Start または Prenet_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 は別のノードでグループの起動を試みることが可能になります。

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 プロパティー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せん。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)

Prenet_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データサービス開発ライブラリ (Data Service Development Library、DSDL) に基づいているかどうかによって決まります。

- データサービスが監視の対象となるのは、そのサービスが Monitor_start メソッドを実装しており、かつリソースの監視が有効になっている場合です。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、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、また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る場合、検出される障害はリソースのプロセスツリーの終了のみです。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。ただし 、Retry_count の回数を超えると、リソースモニターは終了し、リソースの状態を FAULTED に設定して、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 が RESTART_ONLY に設定されている場合、リソースまたはリソースグループは、Retry_interval 内で Retry_count の回数 scha_control() 関数を呼び出すことで再起動できます。リソースグループが Retry_count を超過すると、再起動の試行が失敗します。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 が LOG_ONLY に設定されている場合、すべての scha_control() 要求はリソースまたはリソースグループを再起動するか、グループの障害をフェイルオーバーします。

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 Fini_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ーは 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

Global_zone_override (boolean)
このプロパティーは、RTR ファイル内で Global_zone=TRUE プロパティーを設定したリソースタイプについてのみ使用できます。Global_zone_override プロパティーの設定は、特定のリソースのリソースタイププロパティー Global_zone の値を上書きします。詳細は、rt_properties(5) のマニュアルページを参照してください。

Global_zone プロパティーを TRUE に設定すると、リソースのメソッドは常にグローバルクラスタ投票ノードで実行されます。

Global_zone プロパティーを TRUE に設定した場合リソースメソッドは常に大域ゾーンで実行されますが、Global_zone_override プロパティーを FALSE に設定すると

、リソースメソッドは強制的に非大域ゾーン (つまりリソースグループが構成されているゾーンクラスタノードまたはグローバルクラスタ非投票ノード) で実行されます。

RTR ファイル内でデフォルト値が指定されている場合、このプロパティーは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティーの Tunable 値は AT_CREATION になります。RTR ファイル内の Tunable 属性は 、AT_CREATION、WHEN_DISABLED、または ANYTIME に設定できます。

RTR ファイルで Tunable 属性を Anytime に設定する場合は注意が必要です 。Global_zone_override プロパティーの変更は、リソースがオンラインの場合でもただちに有効になります。たとえば、Global_zone_override の Tunable 属性を ANYTIME に設定し、非大域ゾーンに構成されているリソースについて Global_zone_override プロパティーを FALSE に設定してあるとします。このリソースがオンラインに切り替わるとき、開始メソッドは非大域ゾーンで実行されます 。その後、Global_zone_override プロパティーを TRUE に設定し、リソースがオフラインに切り替わった場合、停止メソッドは大域ゾーンで実行されます。実装するメソッドのコードは、この可能性に対応できることが必要です。対応できない場合は、Tunable 属性を WHEN_DISABLED または AT_CREATION に設定してください。

カテゴリ: 条件付き/任意

デフォルト: TRUE

調整: AT_CREATION

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ーは 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。このプロパティーは、スケーラブルサービス専用です。RTR ファイルに Scalable プロパティーが宣言されている場合、RGM は自動的にこのプロパティーを作成します。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 プロパティー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`。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`)
このプロパティーは、スケーラブルサービス専用です。RTR ファイルに `Scalable` プロパティーが宣言されている場合、RGM は自動的にこのプロパティーを作成します。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、このプロパティーの `Tunable` 値は `ANYTIME` になります。このプロパティーを変更すると、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ーは 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ーは 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ーは 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ィティー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、このプロパティーは存在しません。

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、プロパティー 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 プロパティーが宣言されている場合、RGM は自動的にこのプロパティーを作成します。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限り使用できません。

このプロパティーは、リソース依存関係プロパティーの設定に基づいて、RGM により自動的に更新されます。このプロパティーを直接設定する必要はありません。しかし、このプロパティー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 プロパティーに追加されます。また、このプロパテ

ィー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ーから削除されます。

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 プロパティーの値です。

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 プロパティーを宣言していない場合、この型のリソースに Num_resource_restarts プロパティーを使用できません。

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

各クラスタノードの 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リソースグループ再起動要求の数。*n* は、Retry_interval プロパティーの値です。

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 プロパティーを宣言していない場合、この型のリソースに Num_resource_restarts プロパティーを使用できません。

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`)
クラスタ管理者が管理ユーティリティー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` プロパティーが 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、このプロパティーは 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ーを設定する方法は、『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ーは 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ーは 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。詳細については、288 ページの「リソースグループのプロパティー」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、依存されてい

るリソースが Prenet_start と Start を完了するまで待機してから、依存しているリソースが 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、このプロパティーを指定するときに、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} 指定されている依存関係をホスト単位に限定します。依存関係の動作は、同じホスト上でのみ依存されているリソースに影響されます。依存しているリソースは、依存されているリソースが同じホスト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ーを持っていて、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ーが存在しない場合、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。

このプロパティーは、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。

ノード上にある「依存されている」リソースで障害が発生し、回復できない場合、RGM は該当ノード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、このプロパティーを指定するときに、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}`
: 指定されている依存関係をホスト単位に限定します。依存関係の動作は、同じホスト上でのみ依存されているリソースに影響されます。依存しているリソースは、依存されているリソースが同じホスト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ーを持っていて、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ーが

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。

このプロパティーの動作は 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、このプロパティーを指定するときに、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} 指定されている依存関係をホスト単位に限定します。依存関係の動作は、同じホスト上でのみ依存されているリソースに影響されます。依存しているリソースは、依存

されているリソースが同じホスト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ーを持っていて、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ーが存在しない場合、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_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。依存されているリソースが起動に失敗したり、オフラインのままであっても、依存しているリソースは依然として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、依存されているリソースが Prenet_start と Start を完了するまで待機してから、依存しているリソースが 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、このプロパティーを指定するときに、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}	指定されている依存関係をホスト単位に限定します。依存関係の動作は、同じホスト上でのみ依存されているリソースに影響されます。依存しているリソースは、依存されているリソースが同じホスト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ーを持っていて、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ーが存在しない場合、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。このプロパティーは、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。このプロパティー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` プロパティーから取得されます (`rg_properties(5)`のマニュアルページを参照)。どちらのプロパティー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。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内に存在している必要があります (`projects(1)`のマニュアルページ、および『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』を参照してください)。

このプロパティーは Solaris 9 OS からサポートされるようになりました。

注 – このプロパティーへの変更は、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`

各クラスタノードの `Resource_state` (`enum`)
RGM が判断した各クラスタノード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ーを構成できません。

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 プロパティーの設定に応じて、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上に保持ことを許可する
- 別のノードへのリソースグループのフェイルオーバーを要求する

このプロパティーは 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、クラスタ管理者のみ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、このプロパティーは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティーの Tunable 値は WHEN_DISABLED になります。

注 – このプロパティー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を再起動を試みます。

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、このプロパティーと Retry_count を組み合わせて使用します。このプロパティーは 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、このプロパティーは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティーの 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キング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、グローバルクラスタ非投票ノード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、Solaris ホストごとに 1 つのノード内だけです。

このプロパティーが 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ーを自動的に作成します
。Affinity_timeout、Load_balancing_policy、Load_balancing_weights、Network_resourc
これらのプロパティーは、RTR ファイル内で明示的に宣言されない限り、デフォルト値を持ちます。RTR ファイルで宣言されている場合、Scalable のデフォルトは TRUE です。

RTR ファイルにこのプロパティー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ーが宣言されていない場合、このリソースはスケーラブルではないため、このプロパティーを調整することはできません。RGM は、スケーラビブルサービスプロパティーをいっさい設定しません。ただし、Network_resources_used および Port_list プロパティーは、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ーは、スケーラブルサービスでも非スケーラブルサービスでも有用です。

このリソースプロパティーと Failover リソースタイププロパティーの併用については、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ーは 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

各クラスタノードの 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

各クラスタノード上の Status_msg (string)
リソースモニターによって、Status プロパティー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ーを空文字列でリセットします。ただし、このプロパティー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ーは 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)。このプロパティーは 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、このプロパティーは任意です。

RTR ファイル内に Tunable 属性が指定されていない場合、このプロパティーの 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。このプロパティーは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。このプロパティーの値は、リソースタイプの`RT_version`プロパティーと等しくなります。リソースの作成時、`Type_version`プロパティー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`プロパティーが新しい値に変更されることがあります。

このプロパティーの調整については、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`)
このプロパティーが`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。

このプロパティーは、`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。

このプロパティーは、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ーは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ーは 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)
このプロパティーが TRUE に設定されている場合、このプロパティーにより弱い形式のクライアントアフィニティーが有効になります。

弱い形式のクライアントアフィニティー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。

弱いアフィニティー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。

このプロパティーは、Load_balancing_policy が Lb_sticky または Lb_sticky_wild の場合にかぎり有効です。

このプロパティーは、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ー

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ーを示します。

プロパティー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ーを使ってリソースグループを作成するときに、必ず値を指定します。
- 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場合、システムのデフォルト値が使用されます。
- 照会のみ。管理ツールから直接設定できません。

以下にプロパティー名とその説明を示します。

`Auto_start_on_new_cluster` (`boolean`)
このプロパティーは、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 (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、クラスタの全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` に設定した場合、クラスタの再起動時にリソースグループが自動的に再起動することはありません。リソースグループは、`clresourcegroup online` コマンドまたは同等の GUI 指令を使用して、最初に手動でオンラインに切り替えられるまで、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` プロパティーの値は、`Maximum_primaries` プロパティー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をオフラインにし、優先度の高いノード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型があります。

スケーラブルリソースグループの場合、ネットワークアドレスリソースを含んでいないため、このプロパティーの影響はありません。

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の最大数です。

RG_mode プロパティーが Failover の場合、このプロパティーの値は 1 を超えてはいけません。RG_mode プロパティーが Scalable の場合は、1 より大きな値を設定できます。

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のリストです。これらのノードは、リソースグループの潜在的主ノードまたはマスターです。

カテゴリ: 任意

デフォルト: クラスタ内のすべての投票ノードの順不同リスト

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ーです。Pathprefix の値はリソースグループごとに固有の値を指定します。

カテゴリ: 任意

デフォルト: 空の文字列

調整: ANYTIME

Pingpong_interval (integer)

負数ではない整数値 (秒)。次のような状況において RGM は、この値を使って、リソースグループをどこでオンラインにするかを決めます。

- 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 -O GIVEOVER コマンドが特定のノード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) 最初のノードは、同じリソースによる別の scha_control -O 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ーを直接設定しません。このプロパティーは、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 によって更新されます。

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ホスト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ホスト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 に対して弱いポジティブアフィニティーを、RG3 に対して強いネガティブアフィニティーをもつことを表しています。

RG_affinities プロパティー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 (ポジティブおよびネガティブ) と RG_dependencies の関係図式の中に循環が含まれていてはなりません。

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。

- ノードがクラスタに結合されると、そのノードでは、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でオンラインに、RG2 がすべてのノード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 に対する Desired_primaries がゼロに設定されている場合は、RG1 に対する Desired_primaries プロパティーをゼロより大きい値に設定することはできません。
- 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 プロパティーを 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グローバルデバイスをスイッチオーバーするかどうかを表します。このプロパティーが TRUE に設定されている場合、大域デバイスはスイッチオーバーされます。このプロパティーが FALSE に設定されている場合、グローバル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大域デバイスに依存するかどうかは、Global_resources_used プロパティーの設定によります。

RG_is_frozen プロパティーをユーザーが直接設定することはありません。RG_is_frozen プロパティーは、大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、スケーラブルグループなのかを指定します。この値が Failover の場合、RGM はグループの Maximum_primaries プロパティーの値を 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に制限します。

このプロパティーの値が Scalable の場合は、Maximum_primaries プロパティーを 1 より大きい値に設定できます。その結果、グループを同時に複数のノードで同時にマスターすることができます。Failover プロパティーの値が 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 の場合、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ーです。この値は、クラスタ内で一意でなければなりません。

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)。このプロパティーは、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 プロパティー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で関連付けられた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内に存在している必要があります (projects(1)のマニュアルページ、および『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』を参照してください)。

このプロパティーは Solaris 9 OS からサポートされるようになりました。

注 – このプロパティーへの変更は、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 プロパティーが 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 プロパティーを使用できるのは、 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 プロパティー」を参照してください。

RG_slm_cpu プロパティーの最大値は 655 です。小数点以下 2 桁まで指定できます。RG_slm_cpu プロパティーには 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ル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 プロパティーに対して行う変更は、動的に考慮されます。

RG_slm_type プロパティーは 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 OS 以降、このプロジェクトはリソースグループのノードに作成されます。このノードはグローバルクラスタ投票ノードでもグローバルクラスタ非投票ノードでもかまいません。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 のプロパティー値の 100 倍です。RG_slm_cpu プロパティーが設定されていない場合、このプロジェクトは project.cpu-shares 値を 1 として作成されます。RG_slm_cpu プロパティーのデフォルト値は 0.01 です。

Solaris 10 OS から、RG_slm_pset_type プロパティーが 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 プロパティーが使用されます。また、RG_slm_cpu プロパティーは 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、『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』を参照してください。

注 – RG_slm_cpu プロパティーは、グローバルクラスタでのみ使用できます。このプロパティーはゾーンクラスタでは使用できません。

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。

このプロパティーは、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 プロパティーが AUTOMATED に設定されている
- RG_slm_pset_type プロパティーが DEDICATED_STRONG または DEDICATED_WEAK に設定されている
- RG_slm_cpu プロパティーが、RG_slm_cpu_min プロパティー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 プロパティーの最大値は 655 です。小数点以下 2 桁まで指定できます。RG_slm_cpu_min プロパティーには 0 を指定しないでください。RG_slm_cpu_min および RG_slm_cpu プロパティーは、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 プロパティーに対して行う変更は、動的に考慮されます。RG_slm_pset_type プロパティーが 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 プロパティー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。次回のスイッチオーバー時に、RG_slm_cpu_min プロパティーが使用できる 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、『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』を参照してください。

注 – RG_slm_cpu_min プロパティーは、グローバルクラスタでのみ使用できます。このプロパティーはゾーンクラスタでは使用できません。

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 プロパティーを 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 プロパティーの 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 プロパティーの合計の 100 倍に設定します。またこのプロパティーは、該当するノード内で RG_slm_type を AUTOMATED に設定します。このノードはグローバルクラスタ投票ノードでもグローバルクラスタ非投票ノードでもかまいません。グローバルクラスタ非投票ノードは、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 プロパティーが 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 プロパティーの説明を参照してください。RG_slm_type プロパティーを 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 プロパティーを MANUAL に設定した場合、RG_project_name プロパティー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す。

リソースプールとプロセッサセットについては、『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』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 プロパティーに 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です 。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。

注 – RG_slm_type プロパティーは、グローバルクラスタでのみ使用できます。このプロパティーはゾーンクラスタでは使用できません。

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。

このプロパティーは、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 プロパティーが 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グローバルクラスタ非投票ノードで実行される

RG_slm_pset_type の取り得る値は DEFAULT、DEDICATED_STRONG、および DEDICATED_WEAK です。

リソースグループを DEDICATED_STRONG または DEDICATED_WEAK として実行するには、そのリソースグループのノードリストにグローバルクラスタ非投票ノードだけが存在するようにリソースグループを構成してください。

グローバルクラスタ非投票ノードは、デフォルトプールである POOL_DEFAULT 以外のプールに対して決して構成しないでください。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。グローバルクラスタ非投票ノードは、デフォルトプール以外のプールに決して動的にバインドしてはいけません。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、確認されます。

DEDICATED_STRONG と DEDICATED_WEAK の値は、ノードリストに同じノードを持つリソースグループと相互に排他的です。同じノード内では、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については 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 プロパティーを DEDICATED_STRONG に設定した場合、Sun Cluster は、RG_slm_type プロパティーが AUTOMATED に設定されている場合に RG_slm_type プロパティー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グローバルクラスタ非投票ノード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。
  - 該当するリソースグループが起動するノード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 プロパティーの合計。
  - 該当するノードで実行中であるリソースグループ内の RG_slm_cpu プロパティーの合計。

  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ノード上でオンラインでなくなった場合、そのプールは破棄され、グローバルクラスタ非投票ノード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。
- ノードを実行しているすべてのリソースグループの RG_slm_cpu プロパティーの合計の 100 倍に zone.cpu_shares を設定します。

ユーザーが RG_slm_pset_type プロパティーを 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 プロパティーを 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該当するノード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 プロパティーの詳細は、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 プロパティーを 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 プロパティー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グローバルクラスタ非投票ノードでオンラインではない場合、CPU シェアの値はそのノードの 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 プロパティーを 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。ノードに割り当てられるシェアは、そのノード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、『Solaris のシステム管理 (Solaris コンテナ : 資源管理と Solaris ゾーン)』を参照してください。

注 – RG_slm_pset_type プロパティーは、グローバルクラスタでのみ使用できます。このプロパティーはゾーンクラスタでは使用できません。

カテゴリ: 任意

デフォルト: default

調整: ANYTIME

各クラスタノードの RG_state (enum)
RGM により
UNMANAGED、ONLINE、OFFLINE、PENDING_ONLINE、PENDING_OFFLINE、ERROR_STOP_FAILED、ONLINE_
または PENDING_ONLINE_BLOCKED に設定され、各クラスタノード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ーを構成できません。ただし、clresourcegroup コマンドを実行するか、同等の clsetup または Sun Cluster Manager コマンドを使用するこ

とによって、このプロパティーを間接的に設定することは可能です。RGM の制御下にないときは、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 – すべてのノードに適用される UNMANAGED 状態を除き、状態は個別のノード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 では OFFLINE であり、ノード B 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、過去には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 Prenet_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上ですでに停止されています。つまり、グループ内の有効なリソースすべてに対して、停止メソッド Monitor_stop、Stop、および Postnet_stop が (各リソースに合わせて) 正常に実行されました。この状態は、リソースグループがノードで最初に起動されるまでの状態としても適用されます。
PENDING_ONLINE	リソースグループはノード上で起動中です。グループ内の有効なリソースに対して、起動メソッド Prenet_start、Start、および Monitor_start が (各リソースに合わせて) 実行されようとしています。
PENDING_OFFLINE	リソースグループはノード上で停止中です。グループ内の有効なリソースに対し

て、停止メソッド Monitor_stop、Stop、および Postnet_stop が (各リソースに合わせて) 実行されようとしています。

ERROR_STOP_FAILED
: リソースグループ内の 1 つ以上のリソースが正常に停止できず、Stop_failed 状態にあります。グループ内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。ERROR_STOP_FAILED 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上での起動が許可されません。

  clresource clear などの管理コマンドを使用して、手動で Stop_failed リソースを終了し、その状態を OFFLINE にリセットする必要があります。

ONLINE_FAULTED
: リソースグループは PENDING_ONLINE で、このノード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 START_FAILED 状態または FAULTED 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。リソースの依存性が満たされると、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かにかかわらず、現在の状態のままとなります。指定されたノード上では、この状態でもリソース

グループを別の状態に手作業で切り替えられます。また、リソースグループ内の個々のリソースを有効または無効にすることもできます。

Suspend_automatic_recovery プロパティーに 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。このプロパティーが 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。

このプロパティー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 プロパティーの値を変更します。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 プロパティー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 プロパティーの値が 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っ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。clresourcegroup コマンドだけがこのプロパティーの影響を受けます。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 プロパティーを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	**clresourcegroup delete RG1**
リソースグループプロパティーを編集する (RG_system を除く)	**clresourcegroup set -p RG_desription=... +**
リソースグループへソースを追加する	**clresource create -g RG1 -t SUNW.nfs R1** 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	**clresource delete R1**
リソースグループに属するリソースのプロパティーを編集する	**clresource set -g RG1 -t SUNW.nfs -p r_description="HA-NFS res" R1**

操作	例
リソースグループをオフラインに切り替える	**clresourcegroup offline RG1**
リソースグループを管理する	**clresourcegroup manage RG1**
リソースグループを管理しない	**clresourcegroup unmanage RG1**
リソースグループのリソースを有効にする	**clresource enable R1**
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	**clresource monitor R1**
リソースグループのリソースを無効にする	**clresource disable R1**
リソースの監視を無効にする	**clresource unmonitor R1**

リソースグループの RG_system プロパティーの値が TRUE の場合、そのリソースグループで編集できるプロパティーは RG_system プロパティー自体だけです。つまり、RG_system プロパティー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ーの属性

この節では、システム定義プロパティーの変更または拡張プロパティーの作成に使用できるリソースプロパティー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。

以下にプロパティー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
: プロパティーのデフォルト値を示します。

`Description`
プロパティー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ーに対する `Description` 属性を設定することはできません。

`Enumlist`
`enum` タイプの場合、プロパティー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ーが 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。拡張プロパティー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ーです。

`Max`
`int` タイプの場合、プロパティー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、プロパティー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、拡張プロパティーがノード単位で設定できることを示します。

`Per_node` プロパティー属性をタイプ定義で指定する場合は、`Default` プロパティー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上でノード単位のプロパティー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す。

タイプ `stringarray` のプロパティーには `Per_node` プロパティー属性を指定できません。

`Property`
リソースプロパティーの名前。

`Tunable`
クラスタ管理者がリソースのプロパティー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ー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ー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。ほかの条件 (「監視をいつ無効にするか」や「いつオフラインにするか」など) を設定する場合は、この値を `ANYTIME` に設定し、`Validate` メソッドを使って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ーごとに異なります。RTR ファイルで特に指定していない限り、拡張プロパティーを調整する設定のデフォルトは `TRUE` (`ANYTIME`) です。

プロパティーの型
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ー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ー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。

◆ ◆ ◆ 付 録 B

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 (Resource Type Registration、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。

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ー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.
#
# Registration information for Domain Name Service (DNS)
#
```

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;

# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.
#
# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.

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

例 B–1 SUNW.Sample RTR ファイル (続き)

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

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

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

例 B–1 SUNW.Sample RTR ファイル (続き)

```
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

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

#
# Extension Properties
#

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

# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 30;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Start 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上でオンラインになると、そのノード上で 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
#
# Start Method for HA-DNS.
#
# This method starts the data service under the control of PMF. Before starting
# the in.named process for DNS, it performs some sanity checks. The PMF tag for
# the data service is $RESOURCE_NAME.named. PMF tries to start the service a
# specified number of times (Retry_count) and if the number of attempts exceeds
# this value within a specified interval (Retry_interval) PMF reports a failure
# to start the service. Retry_count and Retry_interval are both properties of the
# resource set in the RTR file.


#pragma ident   “@(#)dns_svc_start   1.1   00/05/24 SMI”

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

        while getopts ‘R:G:T:’ opt
        do
                case “$opt” in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
```

例 B–2 dns_svc_start メソッド (続き)

```
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    “ERROR: Option $OPTARG unknown”
                     exit 1
                     ;;

                esac
        done

}




#############################################################################
# MAIN
#
#############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args “$@”

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Get the value of the Confdir property of the resource in order to start
# DNS. Using the resource name and the resource group entered, find the value of
# Confdir value set by the cluster administrator when adding theresource.
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get returns the “type” as well as the “value” for the extension
# properties. Get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk ‘{print $2}’`

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

例 B–2 dns_svc_start メソッド (続き)

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

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

# Get the value for Retry_count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Get the value for Retry_interval from the RTR file. Convert this value, which is in
# seconds, to minutes for passing to pmfadm. Note that this is a conversion with
# round-up, for example, 50 seconds rounds up to one minute.
((RETRY_INTRVAL = `scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ 60))

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

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

Stop 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上でオフラインになると、そのノード上で 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
#
# Stop method for HA-DNS
#
# Stop the data service using PMF. If the service is not running the
# method exits with status 0 as returning any other value puts the resource
# in STOP_FAILED state.


#pragma ident   “@(#)dns_svc_stop   1.1   00/05/24 SMI”

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

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

例 B–3 dns_svc_stop メソッド (続き)

```
                    ;;
                esac
        done

}

###############################################################################
# MAIN
#
###############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args “$@”

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Obtain the Stop_timeout value from the RTR file.
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME`

# Attempt to stop the data service in an orderly manner using a SIGTERM
# signal through PMF. Wait for up to 80% of the Stop_timeout value to
# see if SIGTERM is successful in stopping the data service. If not, send SIGKILL
# to stop the data service. Use up to 15% of the Stop_timeout value to see
# if SIGKILL is successful. If not, there is a failure and the method exits with
# non-zero status. The remaining 5% of the Stop_timeout is for other uses.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

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

例 B–3 dns_svc_stop メソッド (続き)

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

          exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
           "HA-DNS is not started"

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

# Successfully stopped DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

gettime ユーティリティーのコードリスト

gettime ユーティリティーは、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 プロパティー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 ユーティリティープログラム

```
# This utility program, used by the probe method of the data service, tracks
# the elapsed time in seconds from a known reference point (epoch point). It
# must be compiled and placed in the same directory as the data service callback
# methods (RT_basedir).

#pragma ident   "@(#)gettime.c   1.1   00/05/24 SMI"

#include <stdio.h>
#include <sys/types.h>
```

例 B–4 gettime.c ユーティリティープログラム (続き)

```
#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"
#
# Probe method for HA-DNS.
#
# This program checks the availability of the data service using nslookup, which
# queries the DNS server to look for the DNS server itself. If the server
# does not respond or if the query is replied to by some other server,
# then the probe concludes that there is some problem with the data service
# and fails the service over to another node in the cluster. Probing is done
# at a specific interval set by THOROUGH_PROBE_INTERVAL in the RTR file.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"

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

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
```

例 B–5 dns_probe プログラム　(続き)

```
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    “ERROR: Option $OPTARG unknown”
                     exit 1
                     ;;
                esac
        done
}

#############################################################################
# restart_service ()
#
# This function tries to restart the data service by calling the Stop method
# followed by the Start method of the dataservice. If the dataservice has
# already died and no tag is registered for the dataservice under PMF,
# then this function fails the service over to another node in the cluster.
#
function restart_service
{
        # To restart the dataservice, first, verify that the
        # dataservice itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the dataservice is still registered under
                # PMF, first stop the dataservice and start it back up again.
                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                STOP_METHOD=`scha_resource_get -O STOP \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
```

例 B–5 dns_probe プログラム　(続き)

```
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                                "${ARGV0} Stop method failed."
                        return 1
                fi

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

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

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

        return 0
}

##############################################################################
# decide_restart_or_failover ()
#
# This function decides the action to be taken upon the failure of a probe:
# restart the data service locally or fail over to another node in the cluster.
#
function decide_restart_or_failover
{

   # Check if this is the first restart attempt.
   if [ $retries -eq 0 ]; then
```

例 B–5 dns_probe プログラム　(続き)

```
         # This is the first failure. Note the time of
         # this first attempt.
         start_time=`$RT_BASEDIR/gettimè
         retries=`expr $retries + 1`
         # Because this is the first failure, attempt to restart
         # the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart data service."
            exit 1
         fi
   else
      # This is not the first failure
      current_time=`$RT_BASEDIR/gettimè
      time_diff=`expr $current_time - $start_timè
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # This failure happened after the time window
         # elapsed, so reset the retries counter,
         # slide the window, and do a retry.
         retries=1
         start_time=$current_time
         # Because the previous failure occurred more than
         # Retry_interval ago, attempt to restart the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      elif [ $retries -ge $RETRY_COUNT ]; then
         # Still within the time window,
         # and the retry counter expired, so fail over.
         retries=0
         scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
             -R $RESOURCE_NAME
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failover attempt failed."
            exit 1
         fi
      else
         # Still within the time window,
         # and the retry counter has not expired,
         # so do another retry.
         retries=`expr $retries + 1`
```

例 B–5 dns_probe プログラム　(続き)

```
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      fi
fi
}

#############################################################################
# MAIN
#############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# The interval at which probing is to be done is set in the system defined
# property THOROUGH_PROBE_INTERVAL. Obtain the value of this property with
# scha_resource_get
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtain the timeout value allowed for the probe, which is set in the
# PROBE_TIMEOUT extension property in the RTR file. The default timeout for
# nslookup is 1.5 minutes.
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Probe_timeout`
PROBE_TIMEOUT=`echo $probe_timeout_info | awk '{print $2}'`

# Identify the server on which DNS is serving by obtaining the value
# of the NETWORK_RESOURCES_USED property of the resource.
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Get the retry count value from the system defined property Retry_count
RETRY_COUNT =`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ
```

例 B–5 dns_probe プログラム (続き)

```
# Get the retry interval value from the system defined property
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Obtain the full path for the gettime utility from the
# RT_basedir property of the resource type.
RT_BASEDIR=scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# The probe runs in an infinite loop, trying nslookup commands.
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

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

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

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

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

例 B–5 dns_probe プログラム　(続き)

```
                                probefail=1
                        fi
                fi
        fi

   # If the probefail variable is not set to 0, either the nslookup command
   # timed out or the reply to the query was came from another server
   # (specified in the /etc/resolv.conf file). In either case, the DNS server is
   # not responding and the method calls decide_restart_or_failover,
   # which evaluates whether to restart the data service or to fail it over
   # to another node.

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

Monitor_start 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.
#
# This method starts the monitor (probe) for the data service under the
# control of PMF. The monitor is a process that probes the data service
# at periodic intervals and if there is a problem restarts it on the same node
# or fails it over to another node in the cluster. The PMF tag for the
# monitor is $RESOURCE_NAME.monitor.

#pragma ident   “@(#)dns_monitor_start   1.1   00/05/24 SMI”

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

例 B–6 dns_monitor_start メソッド (続き)

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

}

##############################################################################
# MAIN
#
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the data service.
```

例 B–6 dns_monitor_start メソッド (続き)

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

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

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

Monitor_stop 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
# Monitor stop method for HA-DNS
# Stops the monitor that is running using PMF.

#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"

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

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
```

例 B–7 dns_monitor_stop メソッド (続き)

```
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    “ERROR: Option $OPTARG unknown”
                     exit 1
                     ;;
                esac
        done
}

##############################################################################
# MAIN
#
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args “$@”

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
          “${ARGV0} Could not stop monitor for resource “ \
          $RESOURCE_NAME
           exit 1
   else
      # Could successfully stop the monitor. Log a message.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
          “${ARGV0} Monitor for resource “ $RESOURCE_NAME \
```

例B–7 dns_monitor_stopメソッド　(続き)

```
          " 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プロパティー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#
# Monitor check  Method for DNS.
#
# The RGM calls this method whenever the fault monitor fails the data service
# over to a new node. Monitor_check calls the Validate method to verify
# that the configuration directory and files are available on the new node.

#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

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

   while getopts 'R:G:T:' opt
   do
      case "$opt" in

      R)
      # Name of the DNS resource.
      RESOURCE_NAME=$OPTARG
      ;;

      G)
      # Name of the resource group in which the resource is
      # configured.
      RESOURCEGROUP_NAME=$OPTARG
      ;;
```

例 B–8 dns_monitor_check メソッド (続き)

```
      T)
      # Name of the resource type.
      RESOURCETYPE_NAME=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
      “ERROR: Option $OPTARG unknown”
      exit 1
      ;;
      esac
   done

}

##############################################################################
# MAIN
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method.
parse_args “$@”

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

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

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

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

例 B–8 dns_monitor_check メソッド (続き)

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

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

# Log a message indicating that monitor check was successful.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS successful."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS not successful."
   exit 1
fi
```

Validate メソッドのコードリスト

このメソッドは、Confdir 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、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
# Validate method for HA-DNS.
# This method validates the Confdir property of the resource. The Validate
# method gets called in two scenarios. When the resource is being created and
# when a resource property is getting updated. When the resource is being
# created, this method gets called with the -c flag and all the system-defined
# and extension properties are passed as command-line arguments. When a resource
# property is being updated, the Validate method gets called with the -u flag,
# and only the property/value pair of the property being updated is passed as a
# command-line argument.
#
# ex: When the resource is being created command args will be
#
# dns_validate -c -R <..> -G <...> -T <..> -r <sysdef-prop=value>...
```

例 B–9 dns_validate メソッド (続き)

```
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# when the resource property is being updated
#
# dns_validate -u -R <..> -G <...> -T <..> -r <sys-prop_being_updated=value>
#   OR
# dns_validate -u -R <..> -G <...> -T <..> -x <extn-prop_being_updated=value>

#pragma ident   “@(#)dns_validate   1.1   00/05/24 SMI”

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

   while getopts ‘cur:x:g:R:T:G:’ opt
   do
                case “$opt” in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                r)
                        #The method is not accessing any system defined
                        #properties, so this is a no-op.
                        ;;
                g)
                        # The method is not accessing any resource group
                        # properties, so this is a no-op.
                        ;;
                c)
                        # Indicates the Validate method is being called while
                        # creating the resource, so this flag is a no-op.
                        ;;
                u)
```

例 B–9 dns_validate メソッド (続き)

```
                        # Indicates the updating of a property when the
                        # resource already exists. If the update is to the
                        # Confdir property then Confdir should appear in the
                        # command-line arguments. If it does not, the method must
                        # look for it specifically using scha_resource_get.
                        UPDATE_PROPERTY=1
                        ;;
                x)
                        # Extension property list. Separate the property and
                        # value pairs using “=” as the separator.
                        PROPERTY=`echo $OPTARG | awk -F= ‘{print $1}’`
                        VAL=echo $OPTARG | awk -F= ‘{print $2}’`

                        # If the Confdir extension property is found on the
                        # command line, note its value.
                        if [ $PROPERTY == “Confdir” ];
                        then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                        fi
                        ;;
                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        “ERROR: Option $OPTARG unknown”
                        exit 1
                        ;;
                esac
   done
}

##############################################################################
# MAIN
#
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Set the Value of CONFDIR to null. Later, this method retrieves the value
# of the Confdir property from the command line or using scha_resource_get.
CONFDIR=””
UPDATE_PROPERTY=0
CONFDIR_FOUND=0
```

例 B–9 dns_validate メソッド (続き)

```
# Parse the arguments that have been passed to this method.
parse_args “$@”

# If the validate method is being called due to the updating of properties
# try to retrieve the value of the Confdir extension property from the command
# line. Otherwise, obtain the value of Confdir using scha_resource_get.
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
   config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
       -G $RESOURCEGROUP_NAME Confdir`
   CONFDIR=`echo $config_info | awk ‘{print $2}’`
fi

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

# Now validate the actual Confdir property value.

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

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

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

exit 0
```

Update メソッドのコードリスト

プロパティーが変更された場合、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
# Update method for HA-DNS.
# The actual updates to properties are done by the RGM. Updates affect only
# the fault monitor so this method must restart the fault monitor.

#pragma ident   “@(#)dns_update   1.1   00/05/24 SMI”

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

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

例 B–10 dns_update メソッド (続き)

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args “$@”

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

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

# When the Update method is called, the RGM updates the value of the property
# being updated. This method must check if the fault monitor (probe)
# is running, and if so, kill it and then restart it.
if pmfadm -q $PMF_TAG.monitor; then

# Kill the monitor that is running already
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                    “${ARGV0} Could not stop the monitor”
                exit 1
        else
                # Could successfully stop DNS. Log a message.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    “Monitor for HA-DNS successfully stopped”
        fi
   # Restart the monitor.
   pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
      -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCETYPE_NAME
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                  “${ARGV0} Could not restart monitor for HA-DNS “
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    “Monitor for HA-DNS successfully restarted”
   fi
fi
exit 0
```

◆ ◆ ◆ 付 録 C

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 の詳細は、第 8 章「サンプル DSDL リソースタイプの実装」を参照してください。

この付録の内容は次のとおりです。

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.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
```

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/xfs -config %s -port %d 2>/dev/null”,
       xfnts_conf, portlist->ports[0].port);

   /*
    * Start HA-XFS under PMF. Note that HA-XFS is started as a single
    * instance service. The last argument to the scds_pmf_start function
    * denotes the level of children to be monitored. A value of -1 for
    * this parameter means that all the children along with the original
    * process are to be monitored.
    */
   scds_syslog(LOG_INFO, “Issuing a start request.”);
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

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

   scds_free_port_list(portlist);
   return (err); /* return Success/failure status */
}


/*
 * svc_stop():
 *
 * Stop the XFS server
 * Return 0 on success, > 0 on failures.
 *
 * 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 (scds_timerun(scds_handle, testcmd, time_remaining,
      SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

      scds_syslog(LOG_ERR,
         “Failed to check server status with command <%s>”,
         testcmd);
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }
   return (0);
}
```

xfnts_monitor_check 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 - Monitor Start method for HA-XFS
 */

#pragma ident “@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI”

#include <rgm/libdsdev.h>
#include “xfnts.h”

/*
 * This method starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as RG-name,RS-name.mon. The restart option of PMF
 * is used but not the “infinite restart”. Instead
 * interval/retry_time is obtained from the RTR file.
 */

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

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

   rc = mon_start(scds_handle);

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

   /* Return the result of monitor_start method */
   return (rc);
}
```

xfnts_monitor_stop 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 - Monitor Stop method for HA-XFS
 */

#pragma ident “@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI”

#include <rgm/libdsdev.h>
#include “xfnts.h”

/*
 * This method stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on RG-name_RS-name.mon.
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int    rc;

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

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

   /* Return the result of monitor stop method */
   return (rc);
}
```

xfnts_probe 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 - Probe for HA-XFS
 */

#pragma ident “@(#)xfnts_probe.c 1.26 01/01/18 SMI”

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include “xfnts.h”


/*
 * main():
 * Just an infinite loop which sleep()s for sometime, waiting for
 * the PMF action script to interrupt the sleep(). When interrupted
 * It calls the start method for HA-XFS to restart it.
 *
 */

int
main(int argc, char *argv[])
{
   int         timeout;
   int         port, ip, probe_result;
   scds_handle_t      scds_handle;

   hrtime_t      ht1, ht2;
   unsigned long      dt;
```

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);
   }

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

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

   /*
    * Set the timeout from the X props. This means that each probe
    * iteration will get a full timeout on each network resource
    * without chopping up the timeout between all of the network
    * resources configured for this resource.
    */
   timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {

      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));

      /*
       * Now probe all ipaddress we use. Loop over
       * 1. All net resources we use.
       * 2. All ipaddresses in a given resource.
       * For each of the ipaddress that is probed,
```

例 C–5 xfnts_probe.c (続き)

```
       * compute the failure history.
       */
      probe_result = 0;
      /*
       * Iterate through the all resources to get each
       * IP address to use for calling svc_probe()
       */
      for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS supports only one port and
          * hence obtain the port value from the
          * first entry in the array of ports.
          */
         ht1 = gethrtime(); /* Latch probe start time */
         scds_syslog(LOG_INFO, “Probing the service on “
             “port: %d.”, port);

         probe_result =
         svc_probe(scds_handle, hostname, port, timeout);

         /*
          * Update service probe history,
          * take action if necessary.
          * Latch probe end time.
          */
         ht2 = gethrtime();

         /* Convert to milliseconds */
         dt = (ulong_t)((ht2 - ht1) / 1e6);

         /*
          * Compute failure history and take
          * action if needed
          */
         (void) scds_fm_action(scds_handle,
             probe_result, (long)dt);
      }   /* Each net resource */
   }    /* Keep probing forever */
}
```

xfnts_start 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上でオンラインになると、そのノード上で 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 - Start method for HA-XFS
 */

#pragma ident “@(#)xfnts_svc_start.c 1.13 01/01/18 SMI”

#include <rgm/libdsdev.h>
#include “xfnts.h”

/*
 * The start method for HA-XFS. Does some sanity checks on
 * the resource settings then starts the HA-XFS under PMF with
 * an action script.
 */

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

   /*
    * Process all the arguments that have been passed to us from RGM
    * and do some initialization for syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, “Failed to initialize the handle.”);
      return (1);
   }

   /* Validate the configuration and if there is an error return back */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
```

例 C–6 xfnts_start.c (続き)

```
          “Failed to validate configuration.”);
      return (rc);
   }

   /* Start the data service, if it fails return with an error */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

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

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       “Returned from svc_wait”);

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


finished:
   /* Free up the Environment resources that were allocated */
   scds_close(&scds_handle);

   return (rc);
}
```

xfnts_stop 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上でオフラインになると、そのノード上で 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.
```

例 C–7 xfnts_stop.c (続き)

```
 *
 * xfnts_svc_stop.c - Stop method for HA-XFS
 */

#pragma ident “@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI”

#include <rgm/libdsdev.h>
#include “xfnts.h”

/*
 * Stops the HA-XFS process using PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

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

   rc = svc_stop(scds_handle);

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

   /* Return the result of svc_stop method */
   return (rc);
}
```

xfnts_update メソッドのコードリスト

プロパティーが変更された場合、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ー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 - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Some of the resource properties might have been updated. All such
 * updatable properties are related to the fault monitor. Hence, just
 * restarting the monitor should be enough.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

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

   /*
    * check if the Fault monitor is already running and if so stop and
    * restart it. The second parameter to scds_pmf_restart_fm() uniquely
    * identifies the instance of the fault monitor that needs to be
    * restarted.
    */

   scds_syslog(LOG_INFO, “Restarting the fault monitor.”);
   result = scds_pmf_restart_fm(scds_handle, 0);
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          “Failed to restart fault monitor.”);
      /* Free up all the memory allocated by scds_initialize */
```

例 C–8 xfnts_update.c (続き)

```
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       “Completed successfully.”);

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

   return (0);
}
```

xfnts_validate 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 - validate method for HA-XFS
 */

#pragma ident “@(#)xfnts_validate.c 1.9 01/01/18 SMI”

#include <rgm/libdsdev.h>
#include “xfnts.h”

/*
 * Check to make sure that the properties have been set properly.
 */

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

例 C–9 xfnts_validate.c (続き)

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

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

   /* Return the result of validate method */
   return (rc);

}
```

◆ ◆ ◆ 付 録 D

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。

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
- プロパティー名
- 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 (Resource Type Registration、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。

vendor-id 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 `Vendor_id` リソースタイププロパティーの値です。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 で開発されるリソースタイプのベンダー ID 接頭辞は `SUNW` です。

base-rt-name 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 `Resource_type` リソースタイププロパティーの値です。

rt-version 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 `RT_version` リソースタイププロパティーの値です。バージョン接尾辞は、RTR ファイルが `#$upgrade` 指令を含む場合、完全なリソースタイプ名の部分だけを示します。`#$upgrade` 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、255 ページの「資源タイプのプロパティー」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 (#$upgrade 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ー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 (#$upgrade 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ー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、プロパティー値と記述値という 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 (;)

◆ ◆ ◆ 付録 E

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。

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、366 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に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、このバインドは、すべてのネットワークインタフェースの IP アドレスをすべて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ーを使用すると、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 プロパティー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。48 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ープロトコルは、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。

◆ ◆ ◆ 付録 F

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。

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:
                       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)
–>
<!ELEMENT NVPAIR (NAME,VALUE+)>
<!– NAME definition

        The NAME is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
–>
<!ELEMENT NAME (#PCDATA)>

<!– VALUE definition
        The VALUE is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
–>
<!ELEMENT VALUE (#PCDATA)>
```

SC_REPLY XML DTD

```
<!– SC_REPLY XML format specification

        Copyright 2001-2006 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.
–>
<!– SC_REPLY definition

        The root element of the XML document represents a reply to a message.  The reply
        contains a status code and a status message.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                STATUS_CODE:    The return code for the message.  One of the
                                following: OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
                                MALFORMED, INVALID_XML, VERSION_TOO_HIGH, or
                                VERSION_TOO_LOW.

                CONTENTS:
```

```
                        SUBELEMENTS: SC_STATUS_MSG(1)
—>
<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
        VERSION         NMTOKEN                                         #FIXED   "1.0"
        STATUS_CODE     OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
                        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>
<!— SC_STATUS_MSG definition
        The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
        code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary string.
—>
<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

SC_EVENT XML DTD

注 – 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
>
```

◆ ◆ ◆ 付録 G

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「クラスタ再構成通知プロトコル」を参照)。

CrnpClient.javaのコンテンツ

```
/*
 * CrnpClient.java
 * ===============
 *
 * Note regarding XML parsing:
 *
 * This program uses the Sun Java Architecture for XML Processing (JAXP) API.
 * See http://java.sun.com/webservices/jaxp/ for API documentation and
 * availability information.
 *
 * This program was written for Java 1.3.1 or higher.
 *
 * Program overview:
 *
 * The main thread of the program creates a CrnpClient object, waits for the
 * user to terminate the demo, then calls shutdown on the CrnpClient object
 * and exits the program.
 *
 * The CrnpClient constructor creates an EventReceptionThread object,
 * opens a connection to the CRNP server (using the host and port specified
 * on the command line), constructs a registration message (based on the
 * command-line specifications), sends the registartion message, and reads
 * and parses the reply.
 *
 * The EventReceptionThread creates a listening socket bound to
 * the hostname of the machine on which this program runs, and the port
 * specified on the command line. It waits for an incoming event callback,
```

```
 * at which point it constructs an XML Document from the incoming socket
 * stream, which is then passed back to the CrnpClient object to process.
 *
 * The shutdown method in the CrnpClient just sends an unregistration
 * (REMOVE_CLIENT) SC_CALLBACK_REG message to the crnp server.
 *
 * Note regarding error handling: for the sake of brevity, this program just
 * exits on most errors.  Obviously, a real application would attempt to handle
 * some errors in various ways, such as retrying when appropriate.
 */

// JAXP packages
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

// standard packages
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * class CrnpClient
 * ----------------
 * See file header comments above.
 */
class CrnpClient
{
 /*
  * main
  * ----
  * The entry point of the execution, main simply verifies the
  * number of command-line arguments, and constructs an instance
  * of a CrnpClient to do all the work.
  */
 public static void main(String []args)
 {
   InetAddress regIp = null;
   int regPort = 0, localPort = 0;

   /* Verify the number of command-line arguments */
   if (args.length < 4) {
      System.out.println(
         "Usage: java CrnpClient crnpHost crnpPort "
```

```
            + "localPort (-ac | -ae | -re) "
            + "[(M | A | RG=name | R=name) [...]]");
         System.exit(1);
      }


      /*
       * We expect the command line to contain the ip/port of the
       * crnp server, the local port on which we should listen, and
       * arguments specifying the type of registration.
       */
      try {
         regIp = InetAddress.getByName(args[0]);
         regPort = (new Integer(args[1])).intValue();
         localPort = (new Integer(args[2])).intValue();
      } catch (UnknownHostException e) {
         System.out.println(e);
         System.exit(1);
      }

      // Create the CrnpClient
      CrnpClient client = new CrnpClient(regIp, regPort, localPort,
         args);

      // Now wait until the user wants to end the program
      System.out.println("Hit return to terminate demo...");

      // read will block until the user enters something
      try {
         System.in.read();
      } catch (IOException e) {
         System.out.println(e.toString());
      }

      // shutdown the client
      client.shutdown();
      System.exit(0);
   }

   /*
    * ======================
    * public methods
    * ======================
    */

    /*
     * CrnpClient constructor
     * ----------------------
```

```
  * Parses the command line arguments so we know how to contact
  * the crnp server, creates the event reception thread, and starts it
  * running, creates the XML DocumentBuilderFactory obect, and, finally,
  * registers for callbacks with the crnp server.
  */
 public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
      String []clArgs)
{
   try {

   regIp = regIpIn;
      regPort = regPortIn;
      localPort = localPortIn;
      regs = clArgs;

     /*
      * Setup the document builder factory for
      * xml processing.
      */
      setupXmlProcessing();

     /*
      * Create the EventReceptionThread, which creates a
      * ServerSocket and binds it to a local ip and port.
      */
      createEvtRecepThr();

     /*
      * Register with the crnp server.
      */
      registerCallbacks();

    } catch (Exception e) {
      System.out.println(e.toString());
      System.exit(1);
    }
 }

/*
 * processEvent
 * ---------------
 * Callback into the CrnpClient, used by the EventReceptionThread
 * when it receives event callbacks.
 */
 public void processEvent(Event event)
 {
  /*
   * For demonstration purposes, simply print the event
```

```
  * to System.out.  A real application would obviously make
  * use of the event in some way.
  */
 event.print(System.out);
}

/*
 * shutdown
 * -------------
 * Unregister from the CRNP server.
 */
public void shutdown()
{
 try {
  /* send an unregistration message to the server */
  unregister();
 } catch (Exception e) {
  System.out.println(e);
  System.exit(1);
 }
}


/*
 * ======================
 * private helper methods
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Create the document builder factory for
 * parsing the xml replies and events.
 */
private void setupXmlProcessing() throws Exception
{
 dbf = DocumentBuilderFactory.newInstance();

 // We don't need to bother validating
 dbf.setValidating(false);
 dbf.setExpandEntityReferences(false);

 // We want to ignore comments and whitespace
 dbf.setIgnoringComments(true);
 dbf.setIgnoringElementContentWhitespace(true);

 // Coalesce CDATA sections into TEXT nodes.
```

```
 dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * ------------------
 * Creates a new EventReceptionThread object, saves the ip
 * and port to which its listening socket is bound, and
 * starts the thread running.
 */
private void createEvtRecepThr() throws Exception
{
 /* create the thread object */
 evtThr = new EventReceptionThread(this);

 /*
  * Now start the thread running to begin listening
  * for event delivery callbacks.
  */
 evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
 * Creates a socket connection to the crnp server and sends
 * an event registration message.
 */
private void registerCallbacks() throws Exception
{
 System.out.println("About to register");

 /*
  * Create a socket connected to the registration ip/port
  * of the crnp server and send the registration information.
  */
 Socket sock = new Socket(regIp, regPort);
 String xmlStr = createRegistrationString();
 PrintStream ps = new PrintStream(sock.getOutputStream());
 ps.print(xmlStr);

 /*
  * Read the reply
  */
 readRegistrationReply(sock.getInputStream());

 /*
```

```
  * Close the socket connection.
  */
 sock.close();
}

/*
 * unregister
 * ----------
 * As in registerCallbacks, we create a socket connection to
 * the crnp server, send the unregistration message, wait for
 * the reply from the server, then close the socket.
 */
private void unregister() throws Exception
{
 System.out.println("About to unregister");

 /*
  * Create a socket connected to the registration ip/port
  * of the crnp server and send the unregistration information.
  */
 Socket sock = new Socket(regIp, regPort);
 String xmlStr = createUnregistrationString();
 PrintStream ps = new PrintStream(sock.getOutputStream());
 ps.print(xmlStr);

 /*
  * Read the reply
  */
 readRegistrationReply(sock.getInputStream());

 /*
  * Close the socket connection.
  */
 sock.close();
}

/*
 * createRegistrationString
 * ------------------
 * Constructs a CallbackReg object based on the command line arguments
 * to this program, then retrieves the XML string from the CallbackReg
 * object.
 */
private String createRegistrationString() throws Exception
{
 /*
  * create the actual CallbackReg class and set the port.
  */
```

```
 CallbackReg cbReg = new CallbackReg();
 cbReg.setPort("" + localPort);

 // set the registration type
 if (regs[3].equals("-ac")) {
  cbReg.setRegType(CallbackReg.ADD_CLIENT);
 } else if (regs[3].equals("-ae")) {
  cbReg.setRegType(CallbackReg.ADD_EVENTS);
 } else if (regs[3].equals("-re")) {
  cbReg.setRegType(CallbackReg.REMOVE_EVENTS);
 } else {
  System.out.println("Invalid reg type: " + regs[3]);
  System.exit(1);
 }

 // add the events
 for (int i = 4; i < regs.length; i++) {
  if (regs[i].equals("M")) {
   cbReg.addRegEvent(createMembershipEvent());
  } else if (regs[i].equals("A")) {
   cbReg.addRegEvent(createAllEvent());
  } else if (regs[i].substring(0,2).equals("RG")) {
   cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
  } else if (regs[i].substring(0,1).equals("R")) {
   cbReg.addRegEvent(createREvent(regs[i].substring(2)));
  }
 }

 String xmlStr = cbReg.convertToXml();
 System.out.println(xmlStr);
 return (xmlStr);
}

/*
 * createAllEvent
 * ----------------
 * Creates an XML registartion event with class EC_Cluster, and no
 * subclass.
 */
private Event createAllEvent()
{
 Event allEvent = new Event();
 allEvent.setClass("EC_Cluster");
 return (allEvent);
}

/*
 * createMembershipEvent
```

```
 * ----------------------
 * Creates an XML registration event with class EC_Cluster, subclass
 * ESC_cluster_memberhip.
 */
private Event createMembershipEvent()
{
 Event membershipEvent = new Event();
 membershipEvent.setClass("EC_Cluster");
 membershipEvent.setSubclass("ESC_cluster_membership");
 return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
 * subclass ESC_cluster_rg_state, and one "rg_name" nvpair (based
 * on input parameter).
 */
private Event createRgEvent(String rgname)
{
 /*
  * Create a Resource Group state change event for the
  * rgname Resource Group.  Note that we supply
  * a name/value pair (nvpair) for this event type, to
  * specify in which Resource Group we are interested.
  */
 /*
  * Construct the event object and set the class and subclass.
  */
 Event rgStateEvent = new Event();
 rgStateEvent.setClass("EC_Cluster");
 rgStateEvent.setSubclass("ESC_cluster_rg_state");

 /*
  * Create the nvpair object and add it to the Event.
  */
 NVPair rgNvpair = new NVPair();
 rgNvpair.setName("rg_name");
 rgNvpair.setValue(rgname);
 rgStateEvent.addNvpair(rgNvpair);

 return (rgStateEvent);
}

/*
 * createREvent
 * ----------------
```

```
 * Creates an XML registration event with class EC_Cluster,
 * subclass ESC_cluster_r_state, and one "r_name" nvpair (based
 * on input parameter).
 */
private Event createREvent(String rname)
{
 /*
  * Create a Resource state change event for the
  * rgname Resource.  Note that we supply
  * a name/value pair (nvpair) for this event type, to
  * specify in which Resource Group we are interested.
  */
 Event rStateEvent = new Event();
 rStateEvent.setClass("EC_Cluster");
 rStateEvent.setSubclass("ESC_cluster_r_state");

 NVPair rNvpair = new NVPair();
 rNvpair.setName("r_name");
 rNvpair.setValue(rname);
 rStateEvent.addNvpair(rNvpair);

 return (rStateEvent);
}

/*
 * createUnregistrationString
 * ------------------
 * Constructs a REMOVE_CLIENT CallbackReg object, then retrieves
 * the XML string from the CallbackReg object.
 */
private String createUnregistrationString() throws Exception
{
 /*
  * Crate the CallbackReg object.
  */
 CallbackReg cbReg = new CallbackReg();
 cbReg.setPort("" + localPort);
 cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

 /*
  * we marshall the registration to the OutputStream
  */
 String xmlStr = cbReg.convertToXml();

 // Print the string for debugging purposes
 System.out.println(xmlStr);
 return (xmlStr);
}
```

```
 /*
  * readRegistrationReply
  * ----------------------
  * Parse the xml into a Document, construct a RegReply object
  * from the document, and print the RegReply object.  Note that
  * a real application would take action based on the status_code
  * of the RegReply object.
  */
 private void readRegistrationReply(InputStream stream)
     throws Exception
 {
  // Create the document builder
  DocumentBuilder db = dbf.newDocumentBuilder();

  //
  // Set an ErrorHandler before parsing
  // Use the default handler.
  //
  db.setErrorHandler(new DefaultHandler());

  //parse the input file
  Document doc = db.parse(stream);

  RegReply reply = new RegReply(doc);
  reply.print(System.out);
 }

 /* private member variables */
 private InetAddress regIp;
 private int regPort;
 private EventReceptionThread evtThr;
 private String regs[];

 /* public member variables */
 public int localPort;
 public DocumentBuilderFactory dbf;
}

/*
 * class EventReceptionThread
 * ---------------------------
 * See file header comments above.
 */
class EventReceptionThread extends Thread
{
 /*
```

```
 * EventReceptionThread constructor
 * --------------------------------
 * Creates a new ServerSocket, bound to the local hostname and
 * a wildcard port.
 */
public EventReceptionThread(CrnpClient clientIn) throws IOException
{
 /*
  * keep a reference to the client so we can call it back
  * when we get an event.
  */
 client = clientIn;

 /*
  * Specify the IP to which we should bind.  It's
  * simply the local host ip.  If there is more
  * than one public interface configured on this
  * machine, we'll go with whichever one
  * InetAddress.getLocalHost comes up with.
  *
  */
 listeningSock = new ServerSocket(client.localPort, 50,
     InetAddress.getLocalHost());
     System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Called by the Thread.Start method.
 *
 * Loops forever, waiting for incoming connections on the ServerSocket.
 *
 * As each incoming connection is accepted, an Event object
 * is created from the xml stream, which is then passed back to
 * the CrnpClient object for processing.
 */
public void run()
{
 /*
  * Loop forever.
  */
 try {
  //
  // Create the document builder using the document
  // builder factory in the CrnpClient.
  //
  DocumentBuilder db = client.dbf.newDocumentBuilder();
```

```
   //
   // Set an ErrorHandler before parsing
   // Use the default handler.
   //
   db.setErrorHandler(new DefaultHandler());

   while(true) {
    /* wait for a callback from the server */
    Socket sock = listeningSock.accept();

    // parse the input file
    Document doc = db.parse(sock.getInputStream());

    Event event = new Event(doc);
    client.processEvent(event);

    /* close the socket */
    sock.close();
   }
   // UNREACHABLE

  } catch (Exception e) {
   System.out.println(e);
   System.exit(1);
  }
 }

 /* private member variables */
 private ServerSocket listeningSock;
 private CrnpClient client;
}

/*
 * class NVPair
 * -----------
 * This class stores a name/value pair (both Strings).  It knows how to
 * construct an NVPAIR XML message from its members, and how to parse
 * an NVPAIR XML Element into its members.
 *
 * Note that the formal specification of an NVPAIR allows for multiple values.
 * We make the simplifying assumption of only one value.
 */
class NVPair
{
 /*
  * Two constructors: the first creates an empty NVPair, the second
  * creates an NVPair from an NVPAIR XML Element.
```

```
 */
public NVPair()
{
 name = value = null;
}

public NVPair(Element elem)
{
 retrieveValues(elem);
}

/*
 * Public setters.
 */
public void setName(String nameIn)
{
 name = nameIn;
}

public void setValue(String valueIn)
{
 value = valueIn;
}

/*
 * Prints the name and value on a single line.
 */
public void print(PrintStream out)
{
 out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an NVPAIR XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.
 */
public Element createXmlElement(Document doc)
{
 // Create the element.
 Element nvpair = (Element)
     doc.createElement("NVPAIR");
 //
 // Add the name.  Note that the actual name is
 // a separate CDATA section.
 //
```

```
 Element eName = doc.createElement("NAME");
 Node nameData = doc.createCDATASection(name);
 eName.appendChild(nameData);
 nvpair.appendChild(eName);
 //
 // Add the value.  Note that the actual value is
 // a separate CDATA section.
 //
 Element eValue = doc.createElement("VALUE");
 Node valueData = doc.createCDATASection(value);
 eValue.appendChild(valueData);
 nvpair.appendChild(eValue);

 return (nvpair);
}

/*
 * retrieveValues
 * ---------------
 * Parse the XML Element to retrieve the name and value.
 */
private void retrieveValues(Element elem)
{
 Node n;
 NodeList nl;

 //
 // Find the NAME element
 //
 nl = elem.getElementsByTagName("NAME");
 if (nl.getLength() != 1) {
  System.out.println("Error in parsing: can't find "
      + "NAME node.");
  return;
 }

 //
 // Get the TEXT section
 //
 n = nl.item(0).getFirstChild();
 if (n == null || n.getNodeType() != Node.TEXT_NODE) {
  System.out.println("Error in parsing: can't find "
      + "TEXT section.");
  return;
 }

 // Retrieve the value
 name = n.getNodeValue();
```

```
  //
  // Now get the value element
  //
  nl = elem.getElementsByTagName("VALUE");
  if (nl.getLength() != 1) {
   System.out.println("Error in parsing: can't find "
       + "VALUE node.");
   return;
  }

  //
  // Get the TEXT section
  //
  n = nl.item(0).getFirstChild();
  if (n == null || n.getNodeType() != Node.TEXT_NODE) {
   System.out.println("Error in parsing: can't find "
       + "TEXT section.");
   return;
  }

  // Retrieve the value
  value = n.getNodeValue();
 }


 /*
  * Public accessors
  */
 public String getName()
 {
  return (name);
 }

 public String getValue()
 {
  return (value);
 }

 // Private member vars
 private String name, value;
}


/*
 * class Event
 * -----------
 * This class stores an event, which consists of a class, subclass, vendor,
```

```
 * publisher, and list of name/value pairs.  It knows how to
 * construct an SC_EVENT_REG XML Element from its members, and how to parse
 * an SC_EVENT XML Element into its members.  Note that there is an assymetry
 * here: we parse SC_EVENT elements, but construct SC_EVENT_REG elements.
 * That is because SC_EVENT_REG elements are used in registration messages
 * (which we must construct), while SC_EVENT elements are used in event
 * deliveries (which we must parse).  The only difference is that SC_EVENT_REG
 * elements don't have a vendor or publisher.
 */
class Event
{

 /*
  * Two constructors: the first creates an empty Event; the second
  * creates an Event from an SC_EVENT XML Document.
  */
 public Event()
 {
  regClass = regSubclass = null;
  nvpairs = new Vector();
 }

 public Event(Document doc)
 {

  nvpairs = new Vector();

  //
  // Convert the document to a string to print for debugging
  // purposes.
  //
  DOMSource domSource = new DOMSource(doc);
  StringWriter strWrite = new StringWriter();
  StreamResult streamResult = new StreamResult(strWrite);
  TransformerFactory tf = TransformerFactory.newInstance();
  try {
   Transformer transformer = tf.newTransformer();
   transformer.transform(domSource, streamResult);
  } catch (TransformerException e) {
   System.out.println(e.toString());
   return;
  }
  System.out.println(strWrite.toString());

  // Do the actual parsing.
  retrieveValues(doc);
 }
```

```
/*
 * Public setters.
 */
public void setClass(String classIn)
{
 regClass = classIn;
}

public void setSubclass(String subclassIn)
{
 regSubclass = subclassIn;
}

public void addNvpair(NVPair nvpair)
{
 nvpairs.add(nvpair);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an SC_EVENT_REG XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.  Relies on the NVPair createXmlElement ability.
 */
public Element createXmlElement(Document doc)
{
 Element event = (Element)
     doc.createElement("SC_EVENT_REG");
 event.setAttribute("CLASS", regClass);
 if (regSubclass != null) {
  event.setAttribute("SUBCLASS", regSubclass);
 }
 for (int i = 0; i < nvpairs.size(); i++) {
  NVPair tempNv = (NVPair)
      (nvpairs.elementAt(i));
  event.appendChild(tempNv.createXmlElement(doc));
 }
 return (event);
}

/*
 * Prints the member vars on multiple lines.
 */
public void print(PrintStream out)
{
 out.println("\tCLASS=" + regClass);
 out.println("\tSUBCLASS=" + regSubclass);
```

```
 out.println("\tVENDOR=" + vendor);
 out.println("\tPUBLISHER=" + publisher);
 for (int i = 0; i < nvpairs.size(); i++) {
  NVPair tempNv = (NVPair)
      (nvpairs.elementAt(i));
  out.print("\t\t");
  tempNv.print(out);
 }
}

 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the class, subclass, vendor,
 * publisher, and nvpairs.
 */
private void retrieveValues(Document doc)
{
 Node n;
 NodeList nl;

 //
 // Find the SC_EVENT element.
 //
 nl = doc.getElementsByTagName("SC_EVENT");
 if (nl.getLength() != 1) {
  System.out.println("Error in parsing: can't find "
      + "SC_EVENT node.");
  return;
 }

 n = nl.item(0);

 //
 // Retrieve the values of the CLASS, SUBCLASS,
 // VENDOR and PUBLISHER attributes.
 //
 regClass = ((Element)n).getAttribute("CLASS");
 regSubclass = ((Element)n).getAttribute("SUBCLASS");
 publisher = ((Element)n).getAttribute("PUBLISHER");
 vendor = ((Element)n).getAttribute("VENDOR");

 //
 // Retrieve all the nv pairs
 //
 for (Node child = n.getFirstChild(); child != null;
      child = child.getNextSibling())
 {
```

```
    nvpairs.add(new NVPair((Element)child));
   }
  }

  /*
   * Public accessor methods.
   */
  public String getRegClass()
  {
   return (regClass);
  }

  public String getSubclass()
  {
   return (regSubclass);
  }

  public String getVendor()
  {
   return (vendor);
  }

  public String getPublisher()
  {
   return (publisher);
  }

  public Vector getNvpairs()
  {
   return (nvpairs);
  }

  // Private member vars.
  private String regClass, regSubclass;
  private Vector nvpairs;
  private String vendor, publisher;
}


/*
 * class CallbackReg
 * -----------
 * This class stores a port and regType (both Strings), and a list of Events.
 * It knows how to construct an SC_CALLBACK_REG XML message from its members.
 *
 * Note that this class does not need to be able to parse SC_CALLBACK_REG
 * messages, because only the CRNP server must parse SC_CALLBACK_REG
 * messages.
```

```
 */
class CallbackReg
{
 // Useful defines for the setRegType method
 public static final int ADD_CLIENT = 0;
 public static final int ADD_EVENTS = 1;
 public static final int REMOVE_EVENTS = 2;
 public static final int REMOVE_CLIENT = 3;

 public CallbackReg()
 {
  port = null;
  regType = null;
  regEvents = new Vector();
 }

 /*
  * Public setters.
  */
 public void setPort(String portIn)
 {
  port = portIn;
 }

 public void setRegType(int regTypeIn)
 {
  switch (regTypeIn) {
  case ADD_CLIENT:
   regType = "ADD_CLIENT";
   break;
  case ADD_EVENTS:
   regType = "ADD_EVENTS";
   break;
  case REMOVE_CLIENT:
   regType = "REMOVE_CLIENT";
   break;
  case REMOVE_EVENTS:
   regType = "REMOVE_EVENTS";
   break;
  default:
   System.out.println("Error, invalid regType " +
       regTypeIn);
   regType = "ADD_CLIENT";
   break;
  }
 }

 public void addRegEvent(Event regEvent)
```

```
{
 regEvents.add(regEvent);
}


/*
 * convertToXml
 * ------------------
 * Constructs an SC_CALLBACK_REG XML Document from the member
 * variables.  Relies on the Event createXmlElement ability.
 */
public String convertToXml()
{
 Document document = null;
 DocumentBuilderFactory factory =
     DocumentBuilderFactory.newInstance();
 try {
  DocumentBuilder builder = factory.newDocumentBuilder();
  document = builder.newDocument();
 } catch (ParserConfigurationException pce) {
  // Parser with specified options can't be built
  pce.printStackTrace();
  System.exit(1);
 }
 Element root = (Element) document.createElement("SC_CALLBACK_REG");
 root.setAttribute("VERSION", "1.0");
 root.setAttribute("PORT", port);
 root.setAttribute("REG_TYPE", regType);
 for (int i = 0; i < regEvents.size(); i++) {
  Event tempEvent = (Event)
      (regEvents.elementAt(i));
  root.appendChild(tempEvent.createXmlElement(document));
 }
 document.appendChild(root);

 //
 // Now convert the document to a string.
 //
 DOMSource domSource = new DOMSource(document);
 StringWriter strWrite = new StringWriter();
 StreamResult streamResult = new StreamResult(strWrite);
 TransformerFactory tf = TransformerFactory.newInstance();
 try {
  Transformer transformer = tf.newTransformer();
  transformer.transform(domSource, streamResult);
 } catch (TransformerException e) {
  System.out.println(e.toString());
  return ("");
```

```
  }
  return (strWrite.toString());
 }

 // private member vars
 private String port;
 private String regType;
 private Vector regEvents;
}

/*
 * class RegReply
 * ----------
 * This class stores a status_code and status_msg (both Strings).
 * It knows how to parse an SC_REPLY XML Element into its members.
 */
class RegReply
{

 /*
  * The only constructor takes an XML Document and parses it.
  */
 public RegReply(Document doc)
 {
  //
  // Now convert the document to a string.
  //
  DOMSource domSource = new DOMSource(doc);
  StringWriter strWrite = new StringWriter();
  StreamResult streamResult = new StreamResult(strWrite);
  TransformerFactory tf = TransformerFactory.newInstance();
  try {
   Transformer transformer = tf.newTransformer();
   transformer.transform(domSource, streamResult);
  } catch (TransformerException e) {
   System.out.println(e.toString());
   return;
  }
  System.out.println(strWrite.toString());

  retrieveValues(doc);
 }

 /*
  * Public accessors
  */
 public String getStatusCode()
 {
```

```
 return (statusCode);
}

public String getStatusMsg()
{
 return (statusMsg);
}

/*
 * Prints the info on a single line.
 */
public void print(PrintStream out)
{
 out.println(statusCode + ": " +
     (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the statusCode and statusMsg.
 */
private void retrieveValues(Document doc)
{
 Node n;
 NodeList nl;

 //
 // Find the SC_REPLY element.
 //
 nl = doc.getElementsByTagName("SC_REPLY");
 if (nl.getLength() != 1) {
  System.out.println("Error in parsing: can't find "
      + "SC_REPLY node.");
  return;
 }

 n = nl.item(0);

 // Retrieve the value of the STATUS_CODE attribute
 statusCode = ((Element)n).getAttribute("STATUS_CODE");

 //
 // Find the SC_STATUS_MSG element
 //
 nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
 if (nl.getLength() != 1) {
  System.out.println("Error in parsing: can't find "
```

```
        + "SC_STATUS_MSG node.");
    return;
   }

   //
   // Get the TEXT section, if there is one.
   //
   n = nl.item(0).getFirstChild();
   if (n == null || n.getNodeType() != Node.TEXT_NODE) {
    // Not an error if there isn't one, so we
    // just silently return.
    return;
   }

   // Retrieve the value
   statusMsg = n.getNodeValue();
  }

  // private member vars
  private String statusCode;
  private String statusMsg;
}
```

索引

B

C

D

E

F

G

H

I

S

T

U

V

W

X

ユ

リ