Sun Cluster 3.0 データサービス開発ガイド

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto,, CA 94303-4900
U.S.A. 650-960-1300

Part Number 806-6723
2000 年 12 月, Revision A

原典: *Sun Cluster 3.0 Data Services Developers' Guide*

Part No: 806–1422–10

Revision A

目次

はじめに

このマニュアルでは、RMAPI (Resource Management (リソース管理) API) を使用して Sun Cluster データサービスを開発する方法について説明します。

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™ オペレーティング環境の知識があることを前提としています。

UNIX コマンドの使い方

このマニュアルは、システムの停止、システムの起動、デバイスの構成など、UNIX® の基本的なコマンドや手順については説明しません。

このような情報については、次のマニュアルを参照してください。

- Solaris ソフトウェア環境の AnswerBook™ オンライン文書
- このマニュアル以外にシステムに付属しているソフトウェアマニュアル
- Solaris オペレーティング環境のマニュアルページ

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'**

ただし AnswerBook2 では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

コード例は次のように表示されます。

- C シェルプロンプト

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

- Bourne シェルおよび Korn シェルのプロンプト

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

■ スーパーユーザーのプロンプト

```
system# 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。

関連マニュアル

アプリケーション	タイトル	パート番号
最新情報	『*Sun Cluster 3.0* ご使用にあたって』	806-6735
ハードウェア	『*Sun Cluster 3.0 Hardware Guide*』	806-1420
インストール	『*Sun Cluster 3.0* ソフトウェアのインストール』	806-6725
管理	『*Sun Cluster 3.0* のシステム管理』	806-6731
データサービスの登録と構成	『*Sun Cluster 3.0* データサービスのインストールと構成』	806-6729

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

問い合わせについて

Sun Cluster をインストールまたは使用しているときに問題が発生した場合は、ご購入先に連絡し、次の情報をお伝えください。

- 名前と電子メールアドレス (利用している場合)
- 会社名、住所、および電話番号
- システムのモデルとシリアル番号
- オペレーティング環境のリリース番号 (たとえば、Solaris 7)
- Sun Cluster のリリース番号(たとえば、Sun Cluster 3.0)

ご購入先に知らせる、システム上の各ノードについての情報を収集するには、次のコマンドを使用します。

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する
`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
`prtdiag -v`	システム診断情報を表示する
`scinstall -pv`	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

コマンド	機能
scrgadm -pvv	既存のリソースタイプ、リソースグループ、およびリソースについての静的な属性について、詳細なリストを表示する
scstat -g	すべてのリソースおよびリソースグループについて、動的な状態情報を表示する

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。

RMAPI の概要

このマニュアルでは、RMAPI (Resource Management (リソース管理) API) を使用して、Oracle、iPlanet™ Web Server、DNS などのソフトウェアアプリケーション用の高可用性 (HA) データサービスを作成するためのガイドラインを説明します。したがって、このマニュアルはデータサービスの開発者を対象としており、手順、操作を行うのはデータサービスの開発者を想定しています。

この章では、RMAPI を使用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。

Sun Cluster とは

Sun Cluster 3.0 システムを使用すると、アプリケーションを高可用性でスケーラブルなリソース (データサービス) として実行および管理できます。RGM (Resource

Gropu Manager) というクラスタ機能は、高可用性のための機構を提供します。この機能へのプログラミングインタフェースの要素には、次のようなものがあります。

- コールバックメソッド - RGM がクラスタ上のアプリケーションを制御するために使用します。
- API コマンドと関数 - コールバックメソッドがクラスタ内の要素についての情報にアクセスするときに使用します。
- プロセス管理機能 - クラスタ上のプロセスを監視および再起動します。

RGM は各クラスタ上でデーモンとして動作して、事前構成したポリシーに従って、選択したノード上のリソースを自動的に起動および停止します。リソースの高可用性を実現するために、RGM は、ノードが異常終了または再起動すると、影響を受けるノード上でリソースを停止し、別のノード上でリソースを起動します。RGM はまた、リソースに固有なモニター (監視機能) を起動および停止することによって、障害のあるリソースを検出し、別のノードに再配置したり、さまざまな視点からリソース性能を監視したりできます。

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ます。フェイルオーバーリソースとは、同時に 1 つのノード上だけでオンラインになることができるリソースのことです。スケーラブルリソースとは、同時に複数のノード上でオンラインになることができるリソースのことです。

リソース管理オブジェクトのモデル

ここでは、まず、基本的な用語をいくつか紹介します。次に、さまざまな要素の API を組み合わせて高可用性のアプリケーションを作成する方法について説明します。

RGM および関連する API 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。

まず、開発者は、リソースタイプ ha-oracle を実装します。これは、既存の Oracle DBMS アプリケーションを高可用性にするためのリソースタイプです。次に、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。次に、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分類することによって、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。同様

に、開発者は、もう 1 つのリソースタイプ ha-calendar を作成します。これは、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するためのリソースタイプです。次に、クラスタ管理者は、財務カレンダリソースと財務データベースリソースを同じリソースグループに分類することによって、両方のリソースを同じノード上で実行したり、一緒にフェイルオーバーできるようにします。

リソースタイプ

リソースタイプは、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、およびクラスタの静的な構成の一部を形成するプロパティセットからなります。RGM はリソースタイププロパティを使用して、特定のタイプのリソースを管理します。

ソフトウェアアプリケーションに加えて、リソースタイプは、他のシステムリソース (ネットワークアドレスなど) も表すことができます。

リソースタイプの開発者は、リソースタイププロパティを指定し、その値をリソースタイプ登録 (RTR) ファイルに設定します。リソースタイプ登録ファイルの形式は明確に定義されています。詳細は、21ページの「リソースとリソースタイププロパティの設定」と `rt_reg(4)` のマニュアルページを参照してください。また、リソースタイプ登録ファイルの例については、69ページの「リソースタイプ登録ファイルの定義」を参照してください。

表 A–1 に、リソースタイププロパティのリストを示します。

クラスタ管理者は、リソースタイプの実装と実際のアプリケーションをクラスタにインストールして、管理コマンドで登録します。そして、登録手順で、リソースタイプ登録ファイルの情報をクラスタ構成に入力します。データサービスの登録手順については、『*Sun Cluster 3.0* データサービスのインストールと構成』を参照してください。

リソース

リソースは、そのリソースタイプからプロパティと値を継承します。さらに、開発者は、リソースタイプ登録ファイルでリソースプロパティを宣言できます。リソースプロパティのリストについては、表 A–2を参照してください。

クラスタ管理者は、リソースタイプ登録 (RTR) ファイルにプロパティを指定することによって、特定のプロパティの値を変更できます。たとえば、プロパティ定義に

値の許容範囲を指定しておきます。これにより、プロパティが調節可能なときに、作成時、常時、不可などを指定できます。このような許容範囲内であれば、クラスタ管理者は管理コマンドでプロパティを変更できます。

クラスタ管理者は、同じタイプのリソースをたくさん作成して、各リソースに独自の名前とプロパティ値セットを持たせることができます。これによって、実際の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

各リソースはリソースグループに構成する必要があります。RGM はすべてのリソースを、同じノード上にあるグループ、オンライン (online) とオフライン (offline) に分けます。RGM がリソースをグループ、オンラインまたはオフラインに分けるとき、グループ内の個々のリソース上でコールバックメソッドを呼び出します。

リソースグループが現在オンラインであるノードのことを主ノードと呼びます。リソースグループは、自分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、クラスタ管理者が設定した独自の Nodelist プロパティを持っており、この Nodelist プロパティがリソースグループのすべての潜在的な主ノード (つまり、マスター) を識別します。

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

RGM は 2 種類のリソースグループ、フェイルオーバー (failover) とスケーラブル (scalable) を定義します。フェイルオーバーリソースグループは、同時に 1 つのノード上だけでオンラインになることができます。一方、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なることができます。RGM は、各種類のリソースグループを作成するためのプロパティセットを提供します。このようなプロパティについての詳細は、33ページの「フェイルオーバーリソースの実装」 と 34ページの「スケーラブルリソースの実装」 を参照してください。

リソースグループのプロパティのリストについては、表 A–3 を参照してください。

RGM

RGM (Resource Group Manager) は rgmd デーモンとして実装され、クラスタの各メンバー (ノード) 上で動作します。rgmd プロセスはすべてお互いに通信し、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。

- ノードが起動またはクラッシュしたとき、RGM は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するために、適切なマスター上で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発行して、次のアクションの 1 つを要求できます。
  - リソースグループをマスターする権利を変更する。
  - リソースグループ内の特定のリソースを有効または無効にする。
  - リソース、リソースグループ、またはリソースタイプを作成、削除、変更する。

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。このような活動のことを「再構成」と呼びます。状態の変更を個々のリソースにもたらすために、RGM はリソースタイプに固有なコールバックメソッドをそのリソース上で呼び出します。コールバックメソッドについては、18ページの「コールバックメソッド」 を参照してください。

RGM の管理インタフェース

RGM オブジェクトを管理するための Sun Cluster 3.0 コマンドは、scrgadm(1M)、scswitch(1M)、scstat(1M) -g です。

scrgadm(1M) コマンドは、RGM が使用するリソースタイプ、リソースグループ、およびリソースオブジェクトを表示、作成、構成、削除します。こ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、同じプログラミングコンテキスト内ではアプリケーションインタフェースとして使用されません (この章の後半を参照)。ただし、scrgadm(1M) は API が動作するクラスタ構成を構築するためのツールです。管理インタフェースを理解することは、アプリケーションインタフェースを理解するための背景を知ることになります。このコマンドで実行できる管理タスクの詳細については、scrgadm(1M) のマニュアルページを参照してください。

scswitch(1M) コマンドは、指定されたノード上にあるリソースグループのオンラインとオフラインを切り替えます。そして、リソースまたはそのモニターを有効または無効にします。このコマンドで実行できる管理タスクの詳細については、scswitch(1M) のマニュアルページを参照してください。

scstat(1M) -g コマンドは、すべてのリソースグループおよびリソースについての現在の動的な状態を表示します。

コールバックメソッド

RMAPI (Resource Management (リソース管理) API) は、リソースタイプを実装するときに使用します。リソースタイプの鍵となる要素はコールバックメソッドです。コールバックメソッドとは、RGM から呼び出され、クラスタ上のリソースを制御するプログラムのことです。API はコールバックメソッドの引数と戻り値を定義します。

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。

RMAPI は、次のカテゴリのコールバックメソッドを提供します。

- メソッドの管理と初期化
  - START と STOP 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 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 は、管理アクションによって設定されるプロパティを確認します。

- UPDATE は、オンラインリソースのプロパティ設定を更新します。

- ネット関連メソッド
  - 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」に構成された後に、特別な起動アクションまたは停止アクションを行います。

- 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についての詳細は、第 4 章と rt_callbacks(1HA) のマニュアルページを参照してください。また、コールバックメソッドの使用例については、第 5 章を参照してください。

アクセスメソッド

コールバックメソッドの実装をサポートするために、API は、リソースプロパティや他のクラスタ情報にアクセスするメソッドという形で、RGM へのインタフェースを提供します。アクセスメソッドは、シェルコマンドと C 関数の両方の形で提供されます。

API は、次のようなコマンドと関数を提供します。

- リソース、リソースタイプ、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 プロパティと Status_msg プロパティを設定する。
- リソースグループの再起動または再配置を要求する。

アクセスメソッドについての詳細は、第 4 章を参照してください。また、アクセスメソッドの使用例については、第 5 章を参照してください。

第 2 章

RMAPI の使用方法

この章では、RMAPI (Resource Management (リソース管理) API) を使用してリソースタイプを実装するための詳細な方法について説明します。

この章の内容は、次のとおりです。

リソースとリソースタイププロパティの設定

Sun Cluster は、データサービスの静的な構成を定義するためのリソースタイププロパティを提供します。リソースタイププロパティは、リソースのタイプ、そのバージョン、API のバージョンなどを指定できると同時に、各コールバックメソッドへ

のパスも指定できます。表 A–1 に、すべてのリソースタイププロパティのリストを示します。

リソースタイププロパティは、リソースタイプ登録 (RTR) ファイルで宣言します。RTR ファイルは、クラスタ管理者が Sun Cluster でデータサービスを登録するときの、データサービスの初期構成を定義します。`Installed_nodes` の例外を除いて、クラスタ管理者はリソースタイププロパティを構成できません。

注 - 表 A–1 は、リソースタイププロパティとともに、そのプロパティが任意、必須、または条件付きのどれであるかも示します。任意プロパティは、必ずしも RTR ファイルに指定する必要はありません。任意プロパティを RTR ファイルに指定しなかった場合、システムがデフォルト値を提供します。この表には、任意プロパティのデフォルト値も示します。必須プロパティを RTR ファイルに宣言しなかった場合、データサービスの登録は失敗します。条件付きプロパティを RTR ファイルに宣言しなかった場合、RGM はそのプロパティを作成しないため、クラスタ管理者は使用できません。

次に、RTR ファイルにおけるリソースタイププロパティのエントリの例を示します。

```
# リソースタイプの登録情報の例
Resource_type = example_RT;
Vendor_id = SUNW;
Pkglist = SUNWxxx;
RT_Basedir = /opt/SUNWxxx;
START    = bin/service_start;
STOP    = bin/service_stop;
```

ヒント - RTR ファイルの最初のエントリには、`Resource_type` プロパティを宣言する必要があります。宣言しないと、リソースタイプの登録は失敗します。

Sun Cluster はまた、`Failover_mode`、`Thorough_probe_interval`、およびメソッドタイムアウトなど、リソースの静的な構成を定義するリソースプロパティも提供します。`Resource_state` や `Status` などの動的なリソースプロパティは、管理しているリソースのアクティブな状態を反映します。リソースプロパティに加えて、リソースはそのリソースタイププロパティも継承します。表 A–2 に、リソースプロパティのリストを示します。

リソースタイププロパティと同様に、リソースプロパティも RTR ファイルに宣言します。Sun Cluster が提供するリソースプロパティ、いわゆる、システム定義プロパティの場合、特定の属性を 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とそのデフォルト値を提供します。RTR ファイルを使用すれば、デフォルト値を変更できます。

また、RTR ファイルには、新しいリソースプロパティも指定できます。このようなプロパティのことを拡張プロパティと呼びます。拡張プロパティを指定するには、Sun Cluster が提供するプロパティ属性セットを使用します。表 A–4 に、リソースプロパティを変更および定義するための属性のリストを示します。

慣習上、RTR ファイルでは、リソースプロパティ宣言はリソースタイプ宣言の後に続きます。エントリは左中括弧で始まり、右中括弧で終わります。次に、RTR ファイルにおけるリソース宣言の例を示します。

```
...

# リソースプロパティ宣言は、リソースタイプ宣言の後にある中括弧で囲まれたエントリ
のリストである。

# プロパティ名宣言は、リソースプロパティエントリの左中括弧の直後にある最初の属性
である必要がある。

#

# メソッドタイムアウトには、最小値とデフォルト値を設定する。

{

 Property = Start_timeout;

 MIN=60;

 DEFAULT=300;

}

{

 Property = Stop_timeout;

 MIN=60;

 DEFAULT=300;

}

# リソースの作成時に設定できる拡張属性

{

Property = Log_level;

 Extension;

  enum {OFF, TERSE, VERBOSE};

 DEFAULT = TERSE;

 TUNABLE = AT_CREATION;

 DESCRIPTION = ``Controls the detail of message logging'';

}
```

Start_timeout と Stop_timeout はシステム定義リソースプロパティです。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) を提供します。上記の例の RTR ファイルでは、最小値を 60 秒に、デフォルト値を 300 秒に変更しています。タイムアウト値には、デフォルト値を使用するか 60 秒以上の値に変更できます

注 - 条件付きシステム定義リソースプロパティをそのタイプのリソースとして使用できるようにするには、リソースタイプ登録ファイルに宣言する必要があります。つまり、宣言していない条件付きプロパティは設定または照会できません。

リソースプロパティの重要な点は、クラスタ管理者がある条件下でリソースプロパティを構成できることです。次の表に、クラスタ管理者がリソースプロパティを構成できる条件を表す TUNABLE 属性を示します。

NONE または FALSE	不可
TRUE または ANYTIME	常時
AT_CREATION	データサービスをクラスタに追加するとき
WHEN_DISABLED	データサービスを無効にするとき

プロパティの構成可能性を制限できる属性もあります。たとえば、Min 属性や Max 属性を使用すると、整数プロパティの範囲を設定できます。リソースプロパティの属性の完全なリストについては、表 A–4 を参照してください。

コールバックメソッドの使用方法

この節では、コールバックメソッドの一般的な使用方法について説明します。

リソースとリソースグループのプロパティ情報へのアクセス

RGM がクラスタリソースの活動を制御できるようにするために、コールバックメソッドには、そのリソースプロパティへのアクセスが必要です。API は、リソースのシステム定義プロパティや拡張プロパティにアクセスするための、シェルコマンドと C 関数の両方を提供します。

リソースプロパティを設定する API 関数が存在しない (ただし、`Status` と `Status_msg` を設定する関数を除く) ため、プロパティ機構では、データサービスの動的な状態情報を格納できません。したがって、動的な状態情報は、広域ファイルに格納します。

注 - クラスタ管理者は、`scrgadm(1M)` コマンドまたは、利用可能なグラフィカル管理インタフェースを通じて、特定のリソースプロパティを設定できます。

リソースプロパティにアクセスするための C 関数は、可変引数インタフェースを持ちます。API は、操作を示す文字列値タグを定義し、可変引数リストの解釈を決定します。get アクセス関数は、初期化、最終処理、メモリー管理を行う open 関数や close 関数と共に使用します。

リソースプロパティにアクセスするには、次の 3 つの関数を一緒に使用します。

- `scha_resource_open(3HA)` 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(3HA)` は、リソース情報にアクセスします。
- `scha_resource_close(3HA)` 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。

これら 3 つの関数は 1 つのマニュアルページ内で説明しています。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)`、 `scha_resource_close(3HA)` でアクセスできます。

シェルスクリプトで使用するコマンドバージョンの `scha_resource_get` もあります。このコマンドは、フラグ付き引数として、動作タグ、リソース名、およびそのリソースグループ名をとります。他の動作タグ用に、フラグなし引数も利用できます。このアクセスコマンドについての詳細は、`scha_resource_get(1HA)` のマニュアルページを参照してください。

メソッドの呼び出し回数への非依存性

一般的に、RGM は、同じリソース上で同じメソッドを (同じ引数で) 何回も連続で呼び出しません。しかし、START メソッドが失敗した場合、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 メソッドを呼び出すことができます。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 メソッドは呼び出し回数に依存しないように組み込む必要があります。つまり、同じリソース上で STOP メソッドまたは MONITOR_STOP メソッドを (同じパラメータで) 何回も連続で呼び出しても、一回だけ呼び出したときと同じ結果になることを意味します。

また、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ており、動作していなくても、STOP メソッドと MONITOR_STOP メソッドは 0 (成功) を戻す必要があるということも意味します。

注 - INIT、FINI、BOOT、UPDATE 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。

アプリケーションの制御

ノードがクラスタに結合されるとき、または、クラスタから切り離されるとき、RGM はコールバックメソッドを使用して、実際のリソース (アプリケーション) を制御できます。

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。RGM は、リソースタイプのメソッド関数またはプログラムを、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。START メソッドは、正常に動作しているホストノード上で各リソースを再起動できる方法を RGM に提供するように実装する必要があります。

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 メソッドは戻ってはなりません。初期化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 START メソッドに設定する必要があります。リソースタイプ登録ファイルで Start_timeout プロパティのデフォルト値と最小値を設定します。

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、リソースグループがノード 1 上でオフラインになり、ノード 2 上でもう一度オンラインになると仮定します。リソースグループをオフラインにしている間、RGM は STOP メソッドをそのリソースグループ内のリソース上で呼び出して、ノード 1 上のすべての活動を停止しようとします。ノード 1 上ですべてのリソースの STOP メソッドが完了した後、RGM は、ノード 2 上でそのリソースグループをもう一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 メソッドは戻ってはなりません。最も安全な STOP の実装方法は、ローカルノード上で資源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。リソースタイプ登録ファイルで Stop_timeout プロパティを設定します。

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、システム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、すべてのエラー状態から回復するように、STOP と MONITOR_STOP メソッドを実装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べきです。

リソースの初期化と終了

RGM は、3 つの任意のメソッド INIT、FINI、BOOT を使用し、リソース上で初期化と終了コードを実行できます。リソースを管理下に置くとき (リソースが属しているリソースグループを管理していない状態から管理している状態に切り替えるとき、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を作成するとき)、RGM は INIT メソッドを呼び出して、一度だけリソースの初期化を実行します。

リソースを管理下から外すとき (リソースが属しているリソースグループを管理していない状態に切り替えるとき、または、すでに管理されているリソースグループからリソースを削除するとき)、RGM は FINI を呼び出して、リソースをクリーンアップします。クリーンアップは呼び出し回数に依存しない必要があります。つ

まり、すでにクリーンアップが行われている場合、FINI は 0 (成功) で終了する必要があります。

RGM は、新たにクラスタに結合した、つまり、起動または再起動されたノード上で、BOOT メソッドを呼び出します。

BOOT メソッドは、通常、INIT と同じ初期化を実行します。この初期化は呼び出し回数に依存しない必要があります。つまり、ローカルノード上ですでにリソースが初期化されている場合、BOOT と INIT は 0 (成功) で終了する必要があります。

リソースの監視

RGM は、自動起動するモニターをリソースに提供します。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、ローカルで再起動するか、API 関数 scha_control を呼び出して、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。

また、リソースの性能を監視して、性能を調節または報告できます。可能であれば、リソースタイプに固有な障害モニターを作成することを推奨します。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。

リソースをオフラインにするとき、RGM は、リソース自身を停止する前に、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、リソース自身を起動した後に、MONITOR_START メソッドを呼び出します。

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 3.0* データサービスのインストールと構成』を参照してください。

リソースグループのフェイルオーバーと再起動の制御

リソースモニターは API 関数 scha_control を使用して、リソースグループを他のノードにフェイルオーバーするように要求できます。妥当性検査の 1 つとして、scha_control は MONITOR_CHECK を呼び出して、検査しているノードがリ

ソースを含むリソースグループをマスターできるかどうかを判断します。MONITOR_CHECK が、そのノードは適切ではないと報告した場合、または、メソッドがタイムアウトした場合、RGM は別のノードを探して、scha_control の要求を遂行しようとします。すべてのノード上で 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。

モニターをサポートするリソースプロパティ

リソースモニターには、コールバックメソッドと同様に、リソースプロパティへの一般的なアクセスが必要です。モニターが使用できるシステム定義のリソースプロパティもあります。しかし、このようなプロパティを使用するかどうかはリソースタイプの実装によって異なります。モニター関連のプロパティは次のとおりです。

- Cheap_probe_interval
- Thorough_probe_interval
- Retry_count
- Retry_interval
- Status
- Status_msg

このようなリソースプロパティを読み取るには、scha_resource_get(1HA)(3HA) アクセスの関数またはコマンドを使用します。

Status と Status_msg の設定

Status プロパティと Status_msg プロパティは、リソースモニターによって設定され、モニターから見たリソース状態を反映します。API は、このようなプロパティを設定するための scha_resource_setstatus 関数を提供します。詳細は、scha_resource_setstatus(3HA) と 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。

モニターをサポートするリソースグループプロパティ

モニターが使用できるリソースグループプロパティもありま
す。Nodelist、Maximum_primaries、Desired_primaries、RG_state、
Resource_list、Global_resources_used です。

このようなリソースグループプロパティを読み取るには、3 種類のアクセス関数を使用します。open 関数 (scha_resourcegroup_open(3HA)) は、リソースグループのアクセスを初期化します。close 関数 (scha_resourcegroup_close(3HA)) は、アクセス関数によって割り当てられたメモリーを解放します。可変引数関数 (scha_resourcegroup_get(3HA)) は、動作タグ値によって起動され、参照引数として渡されるクライアント変数にプロパティ値を戻します。リソースグループプロパティのリストについては、表 A–3 を参照してください。

これら 3 つの関数は 1 つのマニュアルページ内で説明しています。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)、 scha_resourcegroup_close(3HA) でアクセスできます。

この機能のスクリプトで使用可能なバージョンは、単一のコマンド scha_resourcegroup_get(1HA) で実装されています。

リソースグループプロパティを直接変更できるインタフェースは存在しません。しかし、scha_control を使用して行った制御要求によって、RGM がリソースグループプロパティを変更することがあります。リソースグループプロパティは、RGM または管理アクションによって変更されます。

モニターをサポートするリソースタイププロパティ

RT_basedir や Installed_nodes のように、モニターが使用できるリソースタイププロパティもあります。このようなプロパティは、たとえば、モニターを実装するプログラムの位置を指定します。

特定のリソースタイプが継承したリソースタイププロパティを読み取るには、scha_resource_get 関数を使用します。任意のリソースタイプのプロパティにアクセスするインタフェースも提供されています。すべてのリソースタイププロパティにアクセスできます。

リソースタイプのアクセスインタフェースは、リソースとリソースグループのアクセスインタフェースのパターンと同じです。open 関数と close 関数は初期化とメモリー管理を行います。可変引数関数は、タグによって決定されたアクセスをプロパ

ティに提供します。これら 3 つの関数は 1 つのマニュアルページ内で説明しています。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)、scha_resourcetype_close(3HA) でアクセスできます。

メッセージログのリソースへの追加

状態メッセージを他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。または、scha_cluster_get(1HA)(3HA) 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 に提供されています。これらのコマンドとプログラムの詳細については、各マニュアルページを参照してください。

- プロセス監視機能: pmfadm(1M) と rpc.pmfd(1M) — プロセス監視機能 (PMF) は、プロセスとその子孫プロセスを監視し、停止した場合は再起動する方法を提供します。この機能は、pmfadm(1M) コマンド (監視するプロセスを起動および制御する) と rpc.pmfd(1M) デーモンからなります。
- halockrun(1M) — ファイルロックを保持したまま、子プログラムを実行するプログラム。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。
- hatimerun(1M) — タイムアウト制御下で、子プログラムを実行するプログラム。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。

リソースへの管理サポートの提供

リソース上での管理アクションには、リソースプロパティの設定と変更があります。このような管理アクションを行うために、API は VALIDATE と UPDATE というコールバックメソッドを定義しています。

リソースが作成されたとき、および、リソースまたはリソースグループ (リソースを含む) のプロパティが管理アクションによって更新されるとき、RGM は VALIDATE 任意メソッドを呼び出します。RGM はリソースとそのリソースグループのプロパティ値を VALIDATE メソッドに渡します。RGM は、リソースタイプの Init_nodes プロパティが示す複数のクラスタノード上で VALIDATE を呼び出します。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、作成または更新は取り消されます。

RGM が VALIDATE を呼び出すのは、リソースまたはリソースグループのプロパティが管理アクションを通じて変更されたときだけです。RGM がプロパティを設定したときや、モニターがリソースプロパティ Status や Status_msg を設定したときではありません。

RGM は、任意の UPDATE メソッドを呼び出して、プロパティが変更されたことを実行中のリソースに通知します。RGM は、管理アクションがリソースまたはそのリソースグループのプロパティの設定に成功した後に、UPDATE を呼び出します。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論理ホスト名や共有アドレスなど) 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。データサービスがフェイルオーバーするかスイッチオーバーされると、ネットワークアドレスリソースは関連するデータサービスリソースと共にクラ

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をいくつか提供します。

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

RG_mode リソースグループプロパティを使用すると、クラスタ管理者はリソースグループがフェイルオーバーまたはスケーラブルのどちらであるかを識別できます。RG_mode が FAILOVER の場合、RGM はリソースグループの Maximum_primaries プロパティを 1 に設定して、リソースグループが単一のノードでマスターされるように制限します。RGM は、Failover プロパティが TRUE であるリソースを、RG_mode が SCALABLE であるリソースグループでインスタンス化することを禁止します。

Implicit_network_dependencies リソースグループプロパティは、リソースグループ内におけるネットワークアドレスリソース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。これは、リソースグループ内のネットワークアドレスが「起動」に構成されるまで、リソースグループ内の非ネットワークアドレス (データサービス) リソースが、自分の START メソッドを呼び出さないことを意味します。ネットワークアドレスリソースには、論理ホスト名や共有アドレスなどのリソースタイプがあります。このプロパティのデフォルト値は 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とは、同時に複数のノード上でオンラインになることができるリソースのことです。スケーラブルリソースには、Sun Cluster HA for iPlanet Web Server や HA-Apache などのデータサービスがあります。

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

ブール型リソースタイププロパティ Scalable は、リソースがスケーラブルであるか (TRUE)、そうでないか (FALSE) を識別します。Scalable プロパティが TRUE であるリソースは、スケーラブルモードであると言います。Scalable プロパティが FALSE であるリソースは、フェイルオーバーモードであると言います。

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

- `Network_resources_used` – このリソースが使用する共有アドレスリソースを識別します。このプロパティのデフォルト値は空の文字列です。したがって、クラスタ管理者は、リソースを作成するときに、スケーラブルサービスが使用する実際の共有アドレスのリストを指定する必要があります。
- `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しても、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` を `NONE` または `FALSE` に設定していない場合)。有効な値は次のとおりです。

  - LB_WEIGHTED – 負荷は、`Load_balancing_weights` プロパティに設定されたウェイトに従って、さまざまなノード間に分散されます。
  - LB_STICKY – スケーラブルサービスのクライアント (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 アドレスに接続されているクライアント (クライアントの IP アドレスで識別される) は、着信しているポート番号に関わらず、常に、同じクラスタノードに送信されます。

  `Load_balancing_policy`、`LB_STICKY`、`LB_STICKY_WILD` を持つスケーラブルなサービスの場合、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。リセットされると、(同じクラスタ内にある) 今までサービスを行っていたノードとは別のノードが、後続のクライアント要求を処理します。

  同様に、サービスの新しいインスタンスをクラスタ上で開始すると、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` – 各ノードに送信される負荷を指定します。形式は *weight*`@`*node*`,`*weight*`@`*node* です。*weight* は、*node* に分散される負荷の相対的な割り当てを示す整数です。ノードに分散される負荷の割合は、このノードのウェイトを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意味します。

- Port_list – サーバーが通信するポートを識別します。このプロパティのデフォルト値は空の文字列です。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。

スケーラブルとフェイルオーバーのどちらのモードでも動作する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 ファイルで Scalable リソースプロパティを宣言します。このリソースプロパティは、値なしで宣言しても (デフォルト値は FALSE)、明示的に値を FALSE に設定してもかまいません。デフォルトでは、このリソースはフェイルオーバーモードで動作します。クラスタ管理者が管理ユーティリティで Scalable の値を TRUE に変更すれば、このリソースをスケーラブルモードで実行できます。

クラスタ管理者は、スケーラブルサービスリソースを含むようなスケーラブルリソースグループを作成できます。スケーラブルリソースは共有アドレスリソースを利用するので、クライアントには、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に見えます。スケーラブルリソースが利用する共有アドレスリソースは、異なるフェイルオーバーリソースグループに存在する必要があります。

クラスタ管理者は、RG_dependencies リソースグループプロパティを使用して、あるノード上でリソースグループがオンラインまたはオフラインになる順番を指定できます。スケーラブルリソースと (スケーラブルリソースが利用する) 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になります。スケーラブルデータサービスが起動される前には、そのネットワークアドレス (共有アドレス) リソースが「起動」に構成されている必要があります。したがって、クラスタ管理者は、共有アドレスリソースを含むリソースグループを含むように RG_dependencies プロパティを設定する必要があります。

RG_mode プロパティを使用すると、クラスタ管理者はリソースグループがフェイルオーバーまたはスケーラブルのどちらであるかを識別できます。RG_mode が SCALABLE の場合、RGM は Maximum_primaries プロパティが 1 よりも大きな値を持つこと (つまり、複数のノードが同時にそのグループをマスターすること) を許可します。RGM は、Failover プロパティが TRUE であるリソースを、RG_mode が SCALABLE であるリソースグループでインスタンス化することを禁止します。

スケーラブルリソースについての詳細は、『*Sun Cluster 3.0* の概念』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

スケーラブルリソースが作成または更新されるとき、RGM は、さまざまなリソースプロパティの妥当性検査を行います。プロパティが正しく構成されていない場合、RGM は作成または更新を拒否します。RGM は次の検査を行います。

- Network_resources_used プロパティは、空の文字列であってはならず、既存の共有アドレスリソースの名前を含む必要があります。スケーラブルリソースを含むリソースグループの Nodelist にあるすべてのノードは、指定した共有アドレスリソースの 1 つである NetIfList プロパティまたは AuxNodeList プロパティに存在する必要があります。
- スケーラブルリソースを含むリソースグループの RG_dependencies プロパティは、スケーラブルリソースの Network_resources_used プロパティに存在する、すべての共有アドレスリソースのリソースグループを含む必要があります。
- Port_list プロパティは、空の文字列であってはならず、ポートとプロトコル (tcp または udp) のペアのリストを含む必要があります。次に例を示します。

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

データサービスの作成と検証

この節では、データサービスを作成および検証する方法について説明します。

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、クラスタノード上にはない独立した (つまり、クラスタノード以外の) 開発マシンで開発を行います。このような場合、pkgadd(1M) を使用して、SUNWscdev パッケージを開発マシンにインストールする必要があります。

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。(クラスタノード以外の) 開発

マシン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、実行および検証できます。

注 - 必ず、開発バージョンの Solaris を使用してください。

この節では、次の手順を使用します。

- 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。

開発環境を設定する方法

この手順では、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. **CD-ROM** のあるディレクトリに移動します。

```
cd CD-ROM_directory
```

2. 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
```

3. **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ます。–L オプションは、静的ライブラリファイルを指定します。–R オプションは、動的ライブラリファイルを指定します。

```
# 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の開発が完了したら、クラスタに転送して検証する必要があります。この転送を行うときは、エラーが発生する可能性を減らすために、データサービスのコードと RTR ファイルを一緒にパッケージに保管して、その後、クラスタのすべてのノード上でパッケージをインストールすることを推奨します。

注 - データサービスをインストールするときは、pkgadd を使用するかどうかに関わらず、すべてのクラスタノード上にインストールする必要があります。

START と 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。どちらのメソッドが適切かを決定するには、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、論理ホスト名のアドレス構成から始まる順番で、起動手順または停止手順を行う必要があります。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、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」に構成された後に、特別な起動アクションまたは停止アクションを行います。

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後に、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 (もしあれば)。
3. ネットワークアドレスを「起動」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 (もしあれば)。

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 (もしあれば)。
2. ネットワークアドレスを「停止」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 (もしあれば)。
4. ネットワークアドレスを取り外します。

START、STOP、PRENET_START、POSTNET_STOP のうち、どのメソッドを使用するかを決定するには、まずサーバー側を考えます。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」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後に、ネットワークアドレスを「停止」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ときに、データサービスの管理ユーティリティまたはライブラリを呼び出す必要がある場合もあります。また、クライアントサーバー型ネットワークインタフェースを使用して管理を実行するような管理ユーティリティまたはライブラリを持っているデータサービスもあります。つまり、管理ユーティリティがサーバーデーモンを呼び出すので、管理ユーティリティまたはライブラリを使用するためには、ネットワークアドレスが「起動」に構成されている必要があります。このような場合は、START メソッドと STOP メソッドを使用します。

データサービスが起動および停止するときにネットワークアドレスが「停止」に構成されている必要がある場合は、PRENET_START メソッドと POSTNET_STOP メソッドを使用して データサービスを起動および停止します。クラスタ再構成、scha_control ギブオーバー、または scswitch スイッチオーバーの後、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どうかによって、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。すると、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。したがって、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 メソッドを呼び出すのは、ネットワークアドレスが「停止」に構成された後だけです。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が応答しない場合を除きます。

START メソッドと STOP メソッドを使用するか、PRENET_START メソッドと 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の要件と動作を考慮に入れる必要があります。

キープアライブの使用方法

サーバー側で TCP キープアライブを有効にしておくと、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、浪費されたリソースが無制限に大きくなり、最終的にはクライアントに障害が発生して再起動します。

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。

他にも、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 キープアライブを有効にしておくと、ある物理ホストから別の物理ホストに論理ホストがフェイルオーバーまたはスイッチオーバーしたとき、(接続の切断が) クライアントに通知されます。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、長時間かかる要求に対するサーバーからの応答をクライアントが待っていると仮定します。このような状況では、クライアントの要求メッセージはすでにサーバーに到達しており、TCP 層で認識されています。したがって、クライアントの TCP モジュールは要求メッセージを再転送し続ける必要はありません。すると、クライアントアプリケーションは要求に対する応答を待ち続けるので、結果としてブロックされます。

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ないので、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に加えて、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があります。アプリケーションレベルのキープアライブ機構を使用するには、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。

リソースグループが物理ホスト間で移動するような場合を想定して、HA データサービスが適切に動作するかどうかを検証します。たとえば、システムがクラッシュした場合や、`scswitch(1M)` 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。このように、データサービス A が自分のサービスを提供するために、データサービス B にそのサービスを提供してもらう場合、データサービス A はデータサービス B に依存していると言います。この要件を満たすために、Sun Cluster では、リソースグループ内でリソースの依存関係を構築できます。依存関係は、Sun Cluster がデータサービスを起動および停止する順番に影響します。詳細は、`scrgadm(1M)` のマニュアルページを参照してください。

あるリソースタイプのリソースが別のリソースタイプのリソースに依存する場合、データサービス開発者は、リソースとリソースグループを適切に構成するようにユーザーに指示するか、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する必要があります。依存するリソースを依存されるリソースと同じノード上で実行する必要がある場合、両方のリソースを同じリソースグループ内で構成する必要があります。

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 データサービス独自のコードで別のデータサービスの可用性をポーリングするかを決定します。依存するリソースと依存されるリソースが異なるノード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、グループ間にはリソースの依存関係を構築できないため、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せず、別のバックエンドデータサービスに依頼して、すべてのデータを格納してもらうものもあります。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の場合、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持っています。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 データベースを高可用性にできます。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。エンドユーザーは 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できます。

アポイントメントカレンダアプリケーションが Oracle データベースと同じノード上で動作する必要がある場合、エンドユーザーは、HA-ORACLE リソースと同じリソースグループ内でアポイントメントカレンダリソースを構築して、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。この依存関係を指定するには、`scrgadm`(1M) の `Resource_dependencies` プロパティを使用します。

アポイントメントカレンダリソースが HA-ORACLE リソースとは別のノード上で動作できる場合、エンドユーザーはこれらのリソースを 2 つの異なるリソースグループ内で構成します。カレンダリソースグループのリソースグループ依存関係を、Oracle リソースグループ上で構築することもできます。しかし、リソースグループ依存関係が有効になるのは、両方のリソースグループが同時に同じノード上で起動または停止されたときだけです。したがって、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、ポーリングして待機しま

す。この場合、通常、カレンダリソースタイプの START メソッドは単に成功を戻すだけです。これは、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なり、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) が行われなくなるためです。しかし、カレンダリソースの START メソッドがタイムアウトまたは非ゼロで終了すると、Oracle データベースが利用できない間、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

データサービスの要件

通常、クラスタを認識しないアプリケーションの高可用性 (HA) を実現するには、この章で説明する要件に適合する必要があります。

データサービス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。データサービスのデータは、高可用性の広域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がデータにアクセスできます。『*Sun Cluster 3.0* の概念』のクラスタファイルシステムに関する情報も参照してください。

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ア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。

クライアントサーバー環境

Sun Cluster は、クライアントサーバーネットワーク環境用に設計されています。`telnet` または `rlogin` 経由でアクセスされるサーバー上でアプリケーションが動作するタイムシェアリング環境では、Sun Cluster は拡張された可用性を提供できません。このような環境では、通常、サーバーに障害が発生しても回復できません。

障害の耐性

データサービスは障害に対する耐性を持たなければなりません。つまり、クラスタ再構成、scha_control ギブオーバー、または scswitch スイッチオーバーの後で再起動されたとき、データサービスは (必要であれば) ディスクデータをクラッシュから回復する必要があります。クラッシュからの回復 (つまり、ディスクをクラッシュから回復し、データサービスを再起動すること) はデータの完全性の問題であるため、クラッシュの耐性はデータサービスの高可用性を実現するための前提条件となります。

注 - データサービスは接続を回復できなくてもかまいません。

多重ホストデータ

高可用性の広域ファイルシステムのディスクセット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 つがディスクにアクセスできます。データサービス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、つまり、そのデータが広域 HA ファイルシステムに格納されていることが必要です。

広域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ィスクグループにマウントされます。ユーザーは、あるディスクグループをマウントされた広域ファイルシステムとして使用し、別のディスクグループをデータサービス (HA Oracle 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。

データサービスは、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もあります。データサービスが固定されたパス名を使用する場合は、データサービスのコードを変更せずに、このパス名を広域ファイルシステム内のファイルの位置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を使用する方法についての詳細は、50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ような何らかの機構を使用するように、データサービスのソースコードを変更する必要があります。この作業は、コマンド行スイッチを追加することにより行うことができます。

Sun Cluster は、ボリューム管理ソフトウェアに構成されている 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 をインストールおよび構成するとき、システム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。

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は、物理ホストではなく、論理ホストのホスト名 (つまり、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) を使用するようにデータサービスを変更する必要があります。

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テイク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

スのペアを持ちます。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、システム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 環境用に書かれているほとんどの市販のデータサービスデーモンは、多重ホームホストを適切に処理できます。ネットワーク通信を行うとき、多くのデータサービスは Solaris のワイルドカードアドレス `INADDR_ANY` にバインドします。すると、`INADDR_ANY` は、すべてのネットワークインタフェースのすべての IP アドレスを自動的に処理します。`INADDR_ANY` は、現在マシンに構成されているすべての IP アドレスに効率的にバインドします。一般的に、`INADDR_ANY` を使用するデータサービスデーモンは、変更しなくても、Sun Cluster 論理ネットワークアドレスを処理できます。

`INADDR_ANY` へのバインドと特定の IP アドレスへのバインド

Sun Cluster の論理ネットワークアドレスの概念では、多重ホーム化されていない環境でも、マシンは複数の IP アドレスを持つことができます。つまり、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 1 つの IP アドレスを持ちます。ネットワークアドレスリソースのマスターになるとき、マシンは動的に追加の IP アドレスを獲得します。ネットワークアドレスリソースのマスターを終了するとき、マシンは動的に IP アドレスを放棄します。

データサービスの中には、`INADDR_ANY` にバインドしていると、Sun Cluster 環境で適切に動作しないもあります。このようなデータサービスは、リソースグループのマスターになるとき、またマスターをやめるときに、バインドしている IP アドレスのセットを動的に変更する必要があります。このようなデータサービスが再バインドする方法の 1 つが、起動メソッドと停止メソッドを使用し、データサービスのデーモンを強制終了および再起動するという方法です。

Network_resources_used リソースプロパティを使用すると、エンドユーザーは、アプリケーションリソースをバインドすべきネットワークアドレスリソースを構成できます。この機能が必要なリソースタイプの場合、そのリソースタイプの RTR ファイルで Network_resources_used プロパティを宣言する必要があります。

リソースグループをオンラインまたはオフラインにするとき、RGM は、データサービスリソースメソッドを呼び出す順番に従って、ネットワークアドレスを取り付けて (plumb) 取り外し (unplumb)、「起動」または「停止」に構成します。詳細は、39ページの「START と 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、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の使用を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でデータサービスのデーモンを終了および再起動する場合、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、テイク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。すでに、単一サーバーの障害と高速再起動を処理するように構成されているアプリケーションとプロトコルは、上記のような場合も、リソースグループのテイクオーバーやスイッチオーバーとして処理します。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。

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

この節では、データサービスのコードを変更しないようにするために、シンボリックリンクを使用する方法について説明します。既存のデータサービスの中には、そのデータファイルへ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もあります。データサービスのコードを変更せずに、シンボリックリンクを使用できる場合もあります。

たとえば、データサービス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、`/global/phys-schost-2/mydatafile` への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には、潜在的な問題があります。つまり、データファイルの名前を内容とともに変更するデータサービス (または、その管理手順) もあります。たとえば、データサービスが更新を実行するとき、まず、新しい一時ファイル `/etc/mydatafile.new` を作成すると仮定します。次に、このデータベースは `rename(2)` システムコール (または `mv(1)` プログラム) を使用し、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の名前に変更することにより、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。

```
rename("/etc/mydatafile.new", "/etc/mydatafile");
```

`rename(2)` の操作は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、クラスタの広域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、テイク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。

このような状況の根本的な問題は、既存のデータサービスがシンボリックリンクに気付かない、つまり、シンボリックリンクを考慮するように作成されていないことです。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、データサービス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、論理ホストのファイルシステムへのデータの配置に関する問題をすべて解決できるわけではありません。

第 4 章

RMAPI リファレンス

この章では、RMAPI (Resource Management (リソース管理) API) を構成するアクセス関数やコールバックメソッドに関する情報を提供します。ここでは、各関数やメソッドについて簡単に説明します。詳細は、Resource Management API 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。

- 54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 (1HA) と C 関数 (3HA)
  - `scha_resource_get(1HA)` (`scha_resource_open_get_close(3HA)`)
  - `scha_resource_setstatus(1HA)(3HA)`
  - `scha_resourcetype_get(1HA)` `scha_resourcetype__open_get_close(3HA)`
  - `scha_resource_resourcegroup_get(1HA)(3HA)` `scha_resource_resourcegroup_open_get_close(3HA)`
  - `scha_control(1HA)(3HA)`
  - `scha_cluster_get(1HA)` `scha_resource_cluster_open_get_close(3HA)`
  - `scha_cluster_getlogfacility(3HA)`
  - `scha_cluster_getnodename(3HA)`
  - `scha_strerror(3HA)`
- 60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

- START
- STOP
- INIT
- FINI
- BOOT
- PRENET_START
- PRENET_STOP
- MONITOR_START
- MONITOR_STOP
- MONITOR_CHECK
- UPDATE
- VALIDATES

RMAPI アクセスメソッド

API は、リソース、リソースタイプ、リソースグループのプロパティ、および他のクラスタ情報にアクセスするための関数を提供します。これらの関数はシェルコマンドと C 関数の両方の形で提供されるため、リソースタイプの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。このコマンドを使用すると、次のことを行えます。

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

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。特に注記しない限り、各コマンドと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 プロパティや Status_msg プロパティを設定できます。

- scha_resource_get(1HA) – RGM の制御下にあるリソースまたは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_resource_get(3HA) と同じ情報を提供します。
- scha_resource_setstatus(1HA) – RGM の制御下にあるリソースの Status プロパティと Status_msg プロパティを設定します。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反映します。このコマンドは、C 関数 scha_resource_setstatus(3HA) と同じ機能を提供します。

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。

リソースタイプコマンド

このコマンドは、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。

- scha_resourcetype_get(1HA) – このコマンドは、C 関数 scha_resourcetype_get(3HA) と同じ機能を提供します。

リソースグループコマンド

以下の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できます。

- scha_resourcegroup_get(1HA) – RGM の制御下にあるリソースグルー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(3HA) と同じ機能を提供します。

- scha_control(1HA) – RGM の制御下にあるリソースグループの再起動、または、異なる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(3HA) と同じ機能を提供します。

クラスタコマンド

このコマンドは、クラスタについての情報 (ノード名、ノード ID、ノードの状態、クラスタ名、リソースグループなど) にアクセスします。

- scha_cluster_get(1HA) – このコマンドは、C 関数 scha_cluster_get(3HA) と同じ情報を提供します。

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。この関数を使用すると、次のことを行えます。

- リソース、リソースタイプ、リソースグループ、クラスタについての情報にアクセスする。
- モニターと併用し、リソースの Status プロパティと Status_msg プロパティを設定する。
- リソースグループの再起動と再配置を要求する。
- 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。詳細は各関数の (3HA) マニュアルページを参照してください。特に注記しない限り、各関数と同じ名前のマニュアルページがあります。C 関数の出力引数や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ます。モニターから見たリソースの状態を表します。

- scha_resource_open(3HA)、scha_resource_get(3HA)、scha_resource_close(3HA) – これらの関数は一緒に、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後に、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。この場合、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、リソースが変更されたことを示します。このメッセージは致命的なエラーメッセージではないため、関数は正常に終了します。したがって、このメッセージは無視してもかまいません。また、現在のハンドルを閉じて新しいハンドルを開き、リソースの情報にアクセスし直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内で説明しています。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)、 scha_resource_close(3HA) でアクセスできます。

- scha_resource_setstatus(3HA) – RGM の制御下にあるリソースの Status プロパティと Status_msg プロパティを設定します。この関数はリソースのモニターによって使用され、モニターから見たリソースの状態を反映します。

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

リソースタイプ関数

これらの関数は一緒に、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。

- scha_resourcetype_open(3HA)、scha_resourcetype_get(3HA)、scha_resourcetype_close(3HA)– scha_resourcetype_open 関数は、リソースタイプ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後に、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。この場合、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、リソースタイプが変更されたことを示します。このメッセージは致命的なエラーメッセージではないため、関数は正常に終了します。したがって、このメッセージは無視してもかまいません。また、現在のハンドルを閉じて新しいハンドルを開き、リソースタイプの情報にアクセスし直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内で説明しています。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)、 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(3HA)、scha_resourcegroup_get(3HA)、scha_resourcegroup_close(3HA)–これらの関数は一緒に、RGM に管理されているリソースグループについての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後に、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。この場合、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、リソースグループが変更されたことを示します。このメッセージは致命的なエラーメッセージではないため、関数は正常に終了します。したがって、このメッセージは無視してもかまいません。また、現在のハンドルを閉じて新しいハンドルを開き、、リソースグループの情報にアクセスし直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内で説明しています。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)、 scha_resourcegroup_close(3HA) でアクセスできます。

- scha_control(3HA)– RGM の制御下にあるリソースグループの再起動、または、異なるノードへの再配置を要求します。

クラスタ関数

以下の関数は、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(3HA)、scha_cluster_get(3HA)、scha_cluster_close(3HA) – これらの関数は一緒に、クラスタについての情報 (ノード名、ノード ID、ノードの状態、クラスタ名、リソースグループなど) にアクセスします。

  これら 3 つの関数は 1 つのマニュアルページ内で説明しています。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)、scha_cluster_close(3HA) でアクセスできます。

  scha_cluster_open 関数がクラスタのハンドルを戻した後に、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。この場合、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_cluster_get 関数に戻し、クラスタが変更されたことを示します。このメッセージは致命的なエラーメッセージではないため、関数は正常に終了します。したがって、このメッセージは無視してもかまいません。また、現在のハンドルを閉じて新しいハンドルを開き、クラスタの情報にアクセスし直してもかまいません。

- scha_cluster_getlogfacility(3HA) – クラスタログとして使用されているシステムログ機能番号を戻します。戻された番号を Solaris の syslog(3) 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。

- scha_cluster_getnodename(3HA) – 関数が呼び出されるクラスタノードの名前を戻します。

ユーティリティ関数

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(3HA) – scha_ 関数の 1 つから戻されるエラーコードを適切なエラーメッセージに変換します。この関数を logger(1) と共に使用すると、メッセージをシステムログ (syslog(3)) に記録できます。

RMAPI コールバックメソッド

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

注 - クライアントプログラムがクラスタシステム上の HA サービスを制御するため、コールバックメソッドはルート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、このようなコールバックメソッドには、特権付き所有者 (bin や root など) を割り当てます。さらに、このようなコールバックメソッドは、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、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、メソッドが呼び出されるタイミング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。詳細は、rt_callbacks(1HA) のマニュアルページを参照してください。

メソッドの引数

RGM はコールバックメソッドを呼び出すとき、次のような引数を使用します。

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、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、-T はリソースのタイプを示し、-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値と呼び出されるリソースグループ) を使用します。

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

終了コード

終了コードは、すべてのコールバックメソッドで共通で、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。これらすべて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。

- 0 (ゼロ) – 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、コールバックメソッドの実行の異常終了 (タイムアウトやコアダンプ) も処理します。

メソッドは、各ノード上で syslog(3) 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、それをユーザーが確認するかどうかは保証できないためです。

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。その他にも、リソース上で初期化と終了コードを実行します。

- START – この必須メソッドは、リソースを含むリソースグループをクラスタノード上でオンラインにするとき、そのノード上で呼び出されます。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 メソッドは終了してはなりません。したがって、START メソッドは終了する前にリソースをポーリングし、リソースが起動しているかどうかを判断する必要があります。さらに、このメソッドには、十分な長さのタイムアウト値を設定する必要があります。たとえば、あるリソース (データベースデーモンなど) 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を設定する必要があります。

  RGM が START メソッドの失敗に応答する方法は、Failover_mode プロパティの設定によって異なります (表 A–2 を参照)。

  リソースの START メソッドのタイムアウト値を設定するには、リソースタイプ登録ファイルの START_TIMEOUT プロパティを使用します。

- STOP – この必須メソッドは、リソースを含むリソースグループをクラスタノード上でオフラインにするとき、そのノード上で呼び出されます。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なりません。そうしないと、RGM が (実際にはアクティブであるのに) リソースが停止したと判断するため、データが破壊されることがあります。データの破壊を防ぐために最も安全な方法は、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、あるリソース (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長めのタイムアウト値を設定する必要があります。

RGM が STOP メソッドの失敗に応答する方法は、Failover_mode プロパティの設定によって異なります (表 A-2 を参照)。

リソースの STOP メソッドのタイムアウト値を設定するには、リソースタイプ登録ファイルの STOP_TIMEOUT プロパティを使用します。

- INIT – この任意メソッドは、リソースを管理下に置くとき (リソースが属しているリソースグループを管理していない状態から管理している状態に切り替えるとき、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を作成するとき) に呼び出され、一度だけリソースの初期化を実行します。このメソッドは、Init_nodes リソースプロパティが示すノード上で呼び出されます。
- FINI – この任意メソッドは、リソースを管理下から外すとき (リソースが属しているリソースグループを管理していない状態に切り替えるとき、または、すでに管理されているリソースグループからリソースを削除するとき) に呼び出され、リソースをクリーンアップします。このメソッドは、Init_nodes リソースプロパティが示すノード上で呼び出されます。
- BOOT – この任意メソッドは、INIT と同様にリソースの初期化を実行します。ただし、リソースを含むリソースグループがすでに RGM の管理下に置かれている状態で、新たにクラスタに参加したノード上で呼び出されます。このメソッドは、Init_nodes リソースプロパティが示すノード上で呼び出されます。BOOT メソッドは、起動または再起動の結果とし、ノードがクラスタに結合または再結合したときに呼び出されます。

注 - INIT、FINI、BOOT メソッドが失敗すると、syslog(3) 関数がエラーメッセージを生成しますが、それ以外は RGM のリソース管理に影響しません。

管理サポートメソッド

リソース上での管理アクションには、リソースプロパティの設定と変更があります。VALIDATE と 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行うことができます。

- VALIDATE – この任意メソッドは、リソースが作成されるとき、および、リソースまたはリソースグループ (リソースを含む) のプロパティが管理アクションによって更新されるときに呼び出されます。このメソッドは、リソースタイプの

Init_nodes プロパティが示す複数のクラスタノード上で呼び出されます。VALIDATE 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ると、作成または更新は取り消されます。

VALIDATE が呼び出されるのは、リソースまたはリソースグループのプロパティが管理アクションを通じて変更されたときだけです。RGM がプロパティを設定したときや、モニターがリソースプロパティ Status や Status_msg を設定したときではありません。

- UPDATE – この任意メソッドは、プロパティが変更されたことを実行中のリソースに通知します。UPDATE は、管理アクションがリソースまたはリソースグループのプロパティの設定に成功した後に呼び出されます。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。

  UPDATE メソッドが失敗すると、syslog(3) 関数がエラーメッセージを生成しますが、それ以外は RGM の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に相対的な順番で、起動手順または停止手順を行う必要があります。任意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」に構成された後に、特別な起動アクションまたはシャットダウンアクションを行うことができます。

- PRENET_START – この任意メソッドは、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」に構成される前に呼び出され、特別な起動アクションを行います。
- POSTNET_STOP – この任意メソッドは、同じリソースグループ内のネットワークアドレスが「停止」に構成された後に呼び出され、特別なシャットダウンアクションを行います。

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ようにも実装できます。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 – この任意メソッドは、リソースが起動した後に呼び出され、リソースを監視するモニターを起動します。
- 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 – この任意メソッドは、リソースグループがノードに再配置される前に呼び出され、ノードの信頼性を査定します。

第 5 章

サンプルアプリケーション

この章では、Sun Cluster データサービスのサンプルのアプリケーション `in.named` 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 (DNS) の実装です。サンプルのアプリケーションを使用し、RMAPI を使用し、データサービスアプリケーションを高可用性にする方法を示します。

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

この章の内容は、次のとおりです。

サンプルアプリケーションの概要

サンプルのデータサービスはクラスタのイベント (管理アクション、アプリケーションの異常終了、ノードの異常終了など) に応じて、DNS アプリケーションを起動、停止、再起動、ラスタノード間の切り替えを行います。

アプリケーションの再起動は、SC 3.0 プロセス監視機能 (PMF) によって管理されます。アプリケーションの障害が再試行最大期間または再試行最大回数を超えると、アプリケーションリソースを含むリソースグループは自動的に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、PROBE メソッドという形で障害監視機能を提供します。PROBE メソッドは、nslookup コマンドを使用し、データサービスが正常な状態であることを保証します。DNS データサービスのハングを検出すると、PROBE メソッドは、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。この方法で状況が改善されず、データサービスの問題が繰り返し検出される場合、PROBE メソッドは、データサービスをクラスタ内の別のノードにフェイルオーバーしようとします。

サンプルのアプリケーションには、具体的に、次のような機能が含まれています。

- リソースタイプ登録ファイル - データサービスの静的なプロパティを定義します。
- START コールバックメソッド - HA-DNS データサービスを含むリソースグループがオンラインになるとき、あるいは、HA-DNS リソースが有効になるときに RGM によって呼び出され、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 データサービスを含むリソースグループがオフラインになるとき、あるいは、HA-DNS リソースが無効になるときに RGM によって呼び出され、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、データ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 によって呼び出され、データ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 - システム管理者がリソースプロパティの値を変更したときに RGM によって呼び出され、障害モニターを再起動します。

リソースタイプ登録ファイルの定義

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

RTR ファイル内の情報は、クラスタ管理者が HA-DNS データサービスを登録したときに RGM によって読み取られます。

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パティ、システム定義リソースプロパティ、拡張プロパティという順番で並んでいます。詳細は、`rt_reg(4)` のマニュアルページと 21ページの「リソースとリソースタイププロパティの設定」を参照してください。

この節では、サンプルの RTR ファイルの特定のプロパティについて説明します。この節で扱うリストは、サンプルの RTR ファイルの一部だけです。サンプルの RTR ファイルの完全なリストについては、132ページの「リソースタイプ登録ファイルのリスト」を参照してください。

サンプル RTR ファイルのリソースタイププロパティ

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

```
#
# Copyright (c) 1998-2000 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#
#pragma ident ``@(#)SUNW.sample 1.1 00/05/24 SMI''

RESOURCE_TYPE = ``sample'';
VENDOR_ID = SUNW;
RT_DESCRIPTION = ``Domain Name Service on Sun Cluster'';

RT_VERSION =''1.0'';
API_VERSION = 2;
FAILOVER = TRUE;

RT_BASEDIR=/opt/SUNWsample/bin;
PKGLIST = SUNWsample;
```

```
START    =   dns_svc_start;
STOP     =   dns_svc_stop;

VALIDATE    =   dns_validate;
UPDATE      =   dns_update;

MONITOR_START    =    dns_monitor_start;
MONITOR_STOP     =    dns_monitor_stop;
MONITOR_CHECK    =  dns_monitor_check;
```

ヒント - RTR ファイルの最初のエントリには、Resource_type プロパティを宣言する必要があります。宣言しないと、リソースタイプの登録は失敗します。

注 - RGM はプロパティ名の大文字と小文字を区別します。Sun が提供する RTR ファイルにおけるプロパティの規約としては、最初の文字は大文字で、残りの文字は小文字にします。ただし、メソッド名は例外で、プロパティ属性と同様に、すべての文字を大文字にします 。

次に、これらのプロパティについての情報を説明します。

- リソースタイプ名は、Resource_type プロパティだけで指定できます (例 : sample)。あるいは、Vendor_id、“.”、Resource_type という形式でも指定できます (例 : SUNW.sample)。

  Vendor_id を接頭辞として使用する場合、リソースタイプを定義している会社の会社名を使用します。リソースタイプ名はクラスタ内で一意である必要があります。

- Rt_version プロパティは、サンプルのデータサービス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 バージョン 3.0 の元で動作していることを示します。

- Failover = TRUE は、同時に複数のノード上でオンラインになることができ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 などは、RGM によって呼び出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 に指定されたディレクトリからのパスになります。

- Pkglist は、SUNWsample をサンプルのデータサービスのインストールを含むパッケージとして識別します。

この RTR ファイルに指定されていないリソースタイププロパティ (Single_instance、Init_nodes、Installed_nodes など) は、デフォルト値を取得します。リソースタイププロパティの完全なリストとそのデフォルト値については、表 A–1 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に指定されている値を変更できません。

サンプル RTR ファイルのリソースプロパティ

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

RTR ファイルのシステム定義プロパティ

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

```
# リソースタイプ宣言の後に、中括弧に囲まれたリソースプロパティ宣言のリスト
# が続く。プロパティ名宣言は、各エントリの左中括弧の直後にある最初
# の属性である必要がある。
#
# <method>_timeout プロパティは、RGM がメソッドの呼び出しが失敗
# したという結論を下すまでの時間 (秒) を設定する。
# すべてのメソッドタイムアウトの MIN 値は 60 秒に設定されている。こ
# れは、管理者が短すぎる時間を設定することを防ぐためである。短すぎ
# る時間を設定すると、スイッチオーバーやフェイルオーバーの性能が上
# がらず、さらには、予期せぬ RGM アクションが発生する可能性がある
# (間違ったフェイルオーバー、ノードの再起動、リソースグループの
# ERROR_STOP_FAILED 状態への移行、オペレータの介入の必要性など)。
# メソッドタイムアウトに短すぎる時間を設定すると、データサービス全
# 体の可用性を下げることになる。
{
 PROPERTY = Start_timeout;
 MIN=60;
 DEFAULT=300;
}
{
 PROPERTY = Stop_timeout;
 MIN=60;
 DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
```

```
        DEFAULT=300;
}
{
        PROPERTY = Update_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Thorough_Probe_Interval;
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}
# 当該ノード上でアプリケーションを正常に起動できないと結論を下すま
# でに、ある期間 (Retry_Interval) に行う再試行の回数
{
        PROPERTY = Retry_Count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Retry_Interval には 60 の倍数を設定する。これは、秒から分に変換さ
# れ、端数が切り上げられるためである。たとえば、60 (秒) という値を指
# 定すると、1 分に変換される。
# このプロパティは再試行数 (Retry_Count) のタイミングを決定する。
{
        PROPERTY = Retry_Interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

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

Sun Cluster はシステム定義プロパティを提供しますが、リソースプロパティ属性を使用すると、異なるデフォルト値を設定できます。リソースプロパティに適用するために利用できる属性の完全なリストについては、128ページの「リソースプロパティの属性」 を参照してください。

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

- Sun Cluster は、すべてのタイムアウトに最小値 (1 秒) とデフォルト値 (3600 秒) を提供します。サンプルの RTR ファイルは、最小値をそのまま (1 秒) にし (`Stop_timeout` を除く。この最小値は 10)、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、タイムアウト値を変更することもできます。たとえば、1 よりも大きくしたり、10 よりも大きくしたり (`Stop_timeout` の場合) できます。Sun Cluster は正当な最大値を持っていません。
- `Thorough_Probe_Interval`、`Retry_count`、`Retry_interval` プロパティの `TUNABLE` 属性は `ANYTIME` に設定されています。この設定は、データサービスが動作中でも、クラスタ管理者がこれらのプロパティの値を変更できることを意味します。上記のプロパティは、サンプルのデータサービスによって実装される障害モニターによって使用されます。サンプルのデータサービスは、管理アクションによってさまざまなリソースが変更されたときに障害モニターを停止および再起動するように、`UPDATE` を実装します。103ページの「`UPDATE` メソッド」 を参照してください。
- リソースプロパティは次のように分類されます。
  - 必須—クラスタ管理者はリソースを作成するときに必ず値を指定する必要があります。
  - 任意—クラスタ管理者が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—RTR ファイルで宣言されている場合だけ、RGM はプロパティを作成します。

  サンプルのデータサービスの障害モニターは、`Thorough_probe_interval`、`Retry_count`、`Retry_interval`、`Network_resources_used` という条件付きプロパティを使用しているため、開発者はこれらのプロパティを RTR ファイルで宣言する必要があります。

RTR ファイルの拡張プロパティ

次のリストに示すように、サンプルの RTR ファイルの終わりには拡張プロパティがあります。

```
# 拡張プロパティ
#
# クラスタ管理者はこのプロパティの値を設定し、アプリケーションが使用
# する構成ファイルが入っているディレクトリを示す必要がある。このアプリ
# ケーションの場合、DNS は PXFS (通常は named.conf) 上の DNS 構成ファイ
# ルのパスを指定する。
{
 PROPERTY = Confdir;
 EXTENSION;
 STRING;
 TUNABLE = AT_CREATION;
 DESCRIPTION = ``The Configuration Directory Path'';
}
# 検証が失敗したと宣言するまでのタイムアウト値 (秒)
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 30;
        TUNABLE = ANYTIME;
        DESCRIPTION = ``Time out value for the probe (seconds)'';
}
```

サンプルの RTR ファイルは 2 つの拡張プロパティ、Confdir と Probe_timeout を定義します。Confdir 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。

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 プロパティを提供しません。開発者はこの拡張プロパティを 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。

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。

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メソッドで使用される次のような機能について説明します。

■ 75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」

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

すべてのメソッドスクリプト (VALIDATE を除く) は、リソース名を渡し、pmfadm(1M) 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します。各スクリプトは変数 PMF_TAG を定義し、pmfadm 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(1) 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、リソースグループ、リソース名を識別できます。

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

 # SIGTERM シグナルをデータサービスに送信し、合計タイムアウト値の 80% だけ待機する。
 pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
 if [ $? -ne 0 ]; then
  logger -p ${SYSLOG_FACILITY}.info \
      -t [$SYSLOG_TAG] \
      ``${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
       SIGKILL''
```

dns_monitor_stop、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
...

# モニターが動作しているかどうかを調べ、動作している場合は強制終了する。
if pmfadm -q $PMF_TAG.monitor; then
 pmfadm -s $PMF_TAG.monitor KILL
```

関数の引数の構文解析

RGM は、次に示すように、すべてのコールバックメソッド (VALIDATE を除く) を呼び出します。

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

method_name は、コールバックメソッドを実装するプログラムのパス名です。データサービスは、各メソッドのパス名を RTR ファイルに指定します。このようなパス

名は、RTR ファイルの Rt_basedir プロパティに指定されたディレクトリからのパスになります。たとえば、サンプルのデータサービスの RTR ファイルでは、ベースディレクトリとメソッド名は次のように指定されます。

```
RT_BASEDIR=/opt/SUNWsample/bin;
START = dns_svc_start;
STOP =  dns_svc_stop;
...
```

コールバックメソッドの引数はすべて、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、-T はリソースのタイプを示し、-G 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についての詳細は、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値と呼び出されるリソースグループ) を使用します。詳細は、98ページの「プロパティ更新の処理」を参照してください。

各メソッドには、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。

次に、サンプルのアプリケーションのメソッドで使用される parse_args 関数を示します。

```
#############################################################################
# プログラム引数を解析する。
#
function parse_args # [args ...]
{
   typeset opt

   while getopts 'R:G:T:' opt
   do
      case "$opt" in
      R)
         # DNS リソースの名前
         RESOURCE_NAME=$OPTARG
         ;;
      G)
         # リソースが構成されているリソース
         # グループの名前
```

(続く)

```
            RESOURCEGROUP_NAME=$OPTARG
            ;;
        T)
            # リソースタイプの名前
            RESOURCETYPE_NAME=$OPTARG
            ;;
        *)
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
          "ERROR: Option $OPTARG unknown"
          exit 1
          ;;
        esac
    done
}
```

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、他のコールバックメソッド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

エラーメッセージをエンドユーザーに出力するには、syslog 機能をメソッドに使用することを推奨します。サンプルのデータサービスのすべてのメソッドは、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 機能番号を取得します。

```
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(1) 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_FACILITY 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。

プロパティ情報の取得

ほとんどのメソッドは、データサービスのリソースとリソースタイプのプロパティについての情報を取得する必要があります。このために、API は scha_resource_get コマンドを提供しています。

リソースプロパティには 2 種類 (システム定義プロパティと拡張プロパティ) あります。システム定義プロパティは事前に定義されており、拡張プロパティはデータサービス開発者が RTR ファイルに定義します。

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

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

拡張プロパティの場合、データサービス開発者は、これが拡張プロパティであることを示す -O パラメータを指定し、最後のパラメータとしてプロパティの名前を指定する必要があります。拡張プロパティの場合、このコマンドは、プロパティのタイプと値の両方を戻します。たとえば、サンプルのデータサービスの検証プログラムは、次に示すように、probe_timeout 拡張プロパティのタイプと値を取得し、次に awk(1) 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、クラスタ上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、クラスタ上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 PRENET_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 PRENET_STOP メソッドを使用する場合は、39ページの「START と 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

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るとき、あるいは、リソースが有効になるとき、RGM はそのノード上で START メソッドを呼び出します。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。

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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、135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 の概要

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。

このメソッドは、プロセス監視機能 (pmfadm) を使用し、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、このメソッドは、一定の期間に一定の回数だけ DNS の起動を再試行します。再試行の回数と期間は、データサービスの RTR ファイル内のプロパティで指定されます。

この START メソッドは呼び出し回数に依存しないことが保証されます。RGM は、STOP メソッドの呼び出しでデータサービスを停止せずに、START メソッドを 2 回呼び出すことはありません。しかし、DNS がすでに動作している場合でも、この START メソッドはその DNS リソース上で呼び出すことができます (成功で終了します)。

構成の確認

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

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

サンプルのデータサービスでは、START メソッドは scha_resource_get(1HA) コマンドを使用して構成ディレクトリの位置を取得します。

注 - Confdir は拡張プロパティであるため、scha_resource_get はタイプと値の両方を戻します。したがって、awk(1) コマンドで値だけを取得し、シェル変数 CONFIG_DIR に格納します。

```
# リソースを追加するときにクラスタ管理者が設定した Confdir の値を見つけ
# る。
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get は拡張プロパティの「タイプ」と「値」を戻す。拡
# 張プロパティの値だけを取得する。
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

次に、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。83ページの「START の終了状態」 を参照してください。

```
# $CONFIG_DIR がアクセス可能であるかどうかを検査する。
if [ ! -d $CONFIG_DIR ]; then
 logger -p ${SYSLOG_FACILITY}.err \
   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
   "${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。存在しない場合、START メソッドはエラーメッセージを記録し、エラー状態で終了します。

```
# データファイルへの相対パス名が存在する場合、$CONFIG_DIR ディレク
# トリに移動する。
cd $CONFIG_DIR
# named.conf ファイルが $CONFIG_DIR ディレクトリ内に存在するかどうか
# を検査する。
if [ ! -s named.conf ]; then
 logger -p ${SYSLOG_FACILITY}.err \
   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
   "${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、起動時にアプリケーションがクラッシュした場合にアプリケーションを再起動するときの、期間と回数を指定できます。このため、RTR ファイルには 2 つのプロパティ Retry_count と Retry_interval があります。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 コマンドを使用して 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、–n オプションと –t オプションを使用し、これらの値を pmfadm に渡します。

```
# RTR ファイルから再試行最大回数の値を取得する。
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`
# RTR ファイルから再試行最大期間の値を取得する。この値の単位は秒であり、
# pmfadm に渡すときは分に変換する必要がある。変換時、端数は切り捨て
# られるので注意すること。たとえば、50 秒は 1 分に切り上げられる。
((RETRY_INTRVAL=`scha_resource_get -O Retry_Interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME` / 60))
# PMF の制御下で in.named デーモンを起動する。RETRY_INTERVAL の期間、
# $RETRY_COUNT の回数だけ、クラッシュおよび再起動できる。どちらかの
# 値以上クラッシュした場合、PMF は再起動をやめる。
# <$RESOURCE_NAME.named> というタグですでにプロセスが登録されている場合、
# PMF はすでにプロセスが動作していることを示す警告メッセージを送信する。
#
pmfadm -c $RESOURCE_NAME.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf
# HA-DNS が起動していることを示すメッセージを記録する。
if [ $? -eq 0 ]; then
 logger -p ${SYSLOG_FACILITY}.err \
   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
   "${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、成功で終了してはなりません。特に、他のデータサービスが依存している場合は注意する必要があります。これを実現するための 1 つの方法は、START メソッドが終了する前に、アプリケーションが動作しているかどうかを確認することです。複雑なアプリケーション (データベースなど) の場合、RTR ファイルの Start_timeout プロパティに十分高い値を設定することによって、アプリケーションが初期化され、クラッシュ回復を実行できる時間を提供します。

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

このメソッドが DNS の起動に失敗し、失敗状態で終了すると、RGM は Failover_mode プロパティを検査し、どのように対処するかを決定します。サンプルのデータサービスは明示的に Failover_mode プロパティを設定していないため、このプロパティはデフォルト値 NONE が設定されています (ただし、クラスタ管理者がデフォルト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、他のアクションは行いません。同じノード上で再起動したり、別のノードにフェイルオーバーしたりするには、ユーザーの介入が必要です。

STOP メソッド

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るとき、あるいは、HA-DNS リソースが無効になるとき、RGM は STOP メソッドを呼び出します。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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、137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 の概要

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、停止処理を規則正しく行うことです。これを実現する最良の方法は、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。これを実現する最良の方法は、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、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、STOP メソッドはプロセス監視機能 (pmfadm) を使用してプロセスを停止します。

この STOP メソッドは呼び出し回数に依存しないことが保証されます。RGM は、START の呼び出しで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 メソッドをそのリソース

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二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。次に、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% は予約されています)。次の例を参照してください。

```
STOP_TIMEOUT≥scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAMÈ
((SMOOTH_TIMEOUT=$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。動作している場合、STOP メソッドはまず 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 メソッドは 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% が経過しても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、成功で終了します。次のコード例に、STOP メソッドがどのように pmfadm を使用して DNS プロセスを停止するかを示します。

```
# in.named が動作しているかどうかを調べて、動作していれば停止する。
if pmfadm -q $RESOURCE_NAME.named; then
 # SIGTERM シグナルをデータサービスに送信し、合計タイムアウト値
 # の 80% だけ待つ。
 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]
\
```

(続く)

```
      ''${ARGV0}: Failed to stop HA-DNS with SIGTERM; Retry
with \
       SIGKILL''

  # SIGTERM シグナルでデータサービスが停止しないので、今度は
  # SIGKILL を使用し、合計タイムアウト値の 15% だけ待つ。
  pmfadm -s $RESOURCE_NAME.named -w $HARD_TIMEOUT KILL
  if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
      ''${ARGV0}: Failed to stop HA-DNS; Exiting UNSUCCESFUL''

      exit 1
  fi
 fi
else
 # この時点でデータサービスは動作していない。メッセージを記録し、
 # 成功で終了する。
 logger -p ${SYSLOG_FACILITY}.err \
          -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
          ''HA-DNS is not started''
 # HA-DNS が動作していない場合でも、成功で終了し、データサービス
 # リソースが STOP_FAILED 状態にならないようにする。
 exit 0
fi
# DNS の停止に成功。メッセージを記録し、成功で終了する。
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、成功で終了してはなりません。特に、他のデータサービスが依存している場合は注意する必要があります。そうしなければ、データが破壊される可能性があります。

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

このメソッドが DNS の停止に失敗し、失敗状態で終了すると、RGM は Failover_mode プロパティを検査し、どのように対処するかを決定します。サンプルのデータサービスは明示的に Failover_mode プロパティを設定していないた

め、このプロパティはデフォルト値 NONE が設定されています (ただし、クラスタ管理者がデフォルト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、他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、ユーザーの介入が必要です。

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(1M) 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後、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

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後、RGMによって自動的に呼び出される 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 によって呼び出される 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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、141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。

検証プログラムの概要

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(1M) を使用し、適切な DNS リソースが動作しているかどうかを確認します。DNS が動作している場合、検証プログラムは一定の期間 (Thorough_probe_interval システム定義プロパティに設定されている期間) だけ休眠し、その後、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。

プロパティ値の取得

このプログラムには、次のプロパティ値が必要です。

- 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.c ユーティリティが格納されているディレクトリを設定します。

scha_resource_get コマンドは、次に示すように、上記プロパティの値を取得し、シェル変数に格納します。

```
PROBE_INTERVAL=`scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME`
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
```

```
\
-G $RESOURCEGROUP_NAME Probe_timeout` \
PROBE_TIMEOUT=`echo $probe_timeout_info | awk '{print $2}'`
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME
\
-G $RESOURCEGROUP_NAME`
RETRY_COUNT=`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME
-G\
 $RESOURCEGROUP_NAME`
RETRY_INTERVAL=`scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME
-G\
 $RESOURCEGROUP_NAME`

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

注 - システム定義プロパティ (Thorough_probe_intervalなど) の場合、scha_resource_get は値だけを戻します。拡張プロパティ (Probe_timeoutなど) の場合、scha_resource_get はタイプと値を戻します。値だけを取得するには awk(1) 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(1M) 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。

```
# nslookup 応答用の一時ファイルを設定する。
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

while ループ自身は、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する。
- hatimerun(1M) を使用し、nslookup に Probe_timeout の値とターゲットホストを渡して起動する。
- nslookup の戻りコード (成功または失敗) に基づいて、*probefail* 変数を設定する。

- *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、他の DNS サーバーから来ているのではないことを確認する。

次に、while ループコードを示します。

```
while :
do
 # 検証が動作すべき期間は THOROUGH_PROBE_INTERVAL プロパティに指
 # 定されている。したがって、THOROUGH_PROBE_INTERVAL の間、検証
 # プログラムが休眠するように設定する。
 sleep $PROBE_INTERVAL
 # DNS がサービスを提供している IP アドレス上で nslookup コマンド
 # を実行する。
 hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
    > $DNSPROBEFILE 2>&1
  retcode=$?
  if [ $retcode -ne 0 ]; then
    probefail=1
  fi
 # nslookup への応答が HA-DNS サーバーから来ており、
 # /etc/resolv.conf ファイル内に指定されている他のネームサーバー
 # から来ていないことを確認する。
 if [ $probefail -eq 0 ]; then
      # nslookup 照会に応答したサーバーの名前を取得する。
      SERVER=` awk ' $1=="Server:" { print $2 }' \
         $DNSPROBEFILE | awk -F. ' { print $1 } ' `
      if [ -z "$SERVER" ]; then
         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 ${SYSLOG_FACILITY}.err\
  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]\
  "${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。`decide_restart_or_failover` 関数は次のような条件付きコードを実装します。

- 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 ユーティリティを使用すると、再起動間の時間を追跡できます。このユーティリティは C プログラムで、(Rt_basedir) ディレクトリ内にあります。
- Retry_count と Retry_interval のシステム定義リソースプロパティは、再起動を行う回数と期間を決定します。RTR ファイルのデフォルト値は、Retry_count が 2 回、Retry_interval が 5 分 (300 秒) です。クラスタ管理者はこのデフォルトを変更できます。
- 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。詳細は、92ページの「データサービスの再起動」を参照してください。
- API コマンド scha_control は、GIVEOVER オプションを指定すると、サンプルデータサービスを含むリソースグループをオフラインにし、別のノード上でオンラインにし直します。

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 PMF 下にまだ登録されているかどうかを調べます。サービスが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) 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。したがって、GIVEOVER オプションを指定して scha_control 関数を呼び出し、データサービスを別のノードにフェイルオーバーします。

```
function restart_service
{
```

```
        # データサービスを再起動するには、まず、データサービス自身が
        # PMF 下に登録されているかどうかを確認する。
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # データサービスの TAG が PMF 下に登録されている場合、データサービスを
                # 停止し、起動し直す。

                # 当該リソースの STOP メソッド名と STOP_TIMEOUT 値を取得する。
                STOP_TIMEOUT≧scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_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

                # 当該リソースの START メソッド名と START_TIMEOUT 値を取得する。
                START_TIMEOUT≧scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_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
                # データサービスの TAG が PMF 下に登録されていない場合、
                # データサービスが PMF 下で許可されている再試行最大回数を
                # 超えていることを示す。したがって、データサービスを再起動
                # してはならない。代わりに、同じクラスタ内にある別のノード
                # へのフェイルオーバーを試みる。
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}
```

検証プログラムの終了状態

ローカルでの再起動が失敗したり、別のノードへのフェイルオーバーが失敗したりすると、サンプルのデータサービスの PROBE プログラムは失敗で終了し、“Failover attempt failed (フェイルオーバーは失敗しました)” というエラーメッセージを記録します。

MONITOR_START メソッド

サンプルのデータサービスがオンラインになった後、RGM は MONITOR_START メソッドを呼び出し、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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、146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 の概要

このメソッドは、プロセス監視機能 (pmfadm) を使用して検証プログラムを起動します。

検証プログラムの起動

MONITOR_START メソッドは、Rt_basedir プロパティの値を取得し、PROBE プログラムへ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、MONITOR_START メソッドは検証プログラムを無限に再起動します。

```
# リソースの RT_BASEDIR プロパティを取得することによって、検証プログラ
# ムが存在する場所を見つける。
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME`
# PMF の制御下でデータサービスの検証を開始する。無限再試行オプショ
# ンを使用して検証プログラムを起動する。リソースの名前、タイプ、
# グループを検証プログラムに渡す。
```

(続く)

```
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

サンプルのデータサービスがオフラインになるとき、RGM は MONITOR_STOP メソッドを呼び出し、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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、148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 の概要

このメソッドは、プロセス監視機能 (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で終了します。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。

```
# 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、
# 検証プログラムを停止する。
if pmfadm -q $RESOURCE_NAME.monitor; then
 pmfadm -s $RESOURCE_NAME.monitor KILL
 if [ $? -ne 0 ]; then
```

```
   logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
    "${ARGV0}: Could not stop monitor for resource " \
    $RESOURCE_NAME
         exit 1
 else
   # 検証プログラムの停止に成功。メッセージを記録する。
   logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
    "${ARGV0}: Monitor for resource " $RESOURCE_NAME \
    " successfully stopped"
 fi
fi
exit 0
```

注意 - 検証プログラムを停止するときは、必ず、pmfadm で KILL シグナルを使用するようにしてください。絶対に、マスク可能なシグナル (TERM など) 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。この問題の原因は、PROBE メソッドが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ときに、scha_control を呼び出すところにあります。scha_control が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 MONITOR_STOP メソッドを呼び出したときに、MONITOR_STOP メソッドがマスク可能なシグナルを使用していると、MONITOR_STOP メソッドは scha_control が終了するのを待ち、scha_control は MONITOR_STOP メソッドが終了するのを待つため、結果として両方がハングします。

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

PROBE メソッドが、データサービスを含むリソースグループを新しいノードにフェイルオーバーしようとするとき、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 関数や syslog 機能番号を取得する方法など、すべてのメソッドに共通な機能については説明しません。このような機能については、74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、150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 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拡張プロパティが DNS 構成ディレクトリを指します。したがって、MONITOR_CHECK は VALIDATE メソッドのパスと名前、および Confdir の値を取得します。MONITOR_CHECK は、次のように、この値を VALIDATE に渡します。

```
# リソースタイプの RT_BASEDIR プロパティから VALIDATE メソッドの
# 完全パスを取得する。
RT_BASEDIR=scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME \
 -G $RESOURCEGROUP_NAME
# 当該リソースの VALIDATE メソッド名を取得する。
VALIDATE_METHOD=scha_resource_get -O VALIDATE \
 -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME

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

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

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

ノード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、98ページの「VALIDATE メソッド」を参照してください。

プロパティ更新の処理

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

VALIDATE メソッド

リソースが作成されたとき、および、リソースまたは (リソースを含む) リソースグループのプロパティが管理アクションによって更新されるとき、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、リソースまたはリソースグループのプロパティが管理アクションを通じて変更されたときだけです。RGM がプロパティを設定したときや、モニターがリソースプロパティ Status や 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する場合、MONITOR_CHECK メソッドも明示的に VALIDATE メソッドを呼び出します。

VALIDATE の概要

VALIDATE メソッドを呼び出すとき、RGM は追加の引数 (更新されるプロパティとその値など) を他のメソッドに渡します。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 parse_args 関数を実装する必要があります。

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

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

RGM が VALIDATE メソッドに渡すプロパティの中に Confdir が存在する場合、parse_args 関数はその値を取得および保存します。次に、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 メソッドは Confdir プロパティの妥当性を検査しようとします。まず、VALIDATE メソッドは scha_resource_get 関数を使用し、静的な構成から Confdir の値を取得します。次に、同じ検査を実行し、構成ディレクトリがアクセス可能であるかどうか、および、空でない named.conf ファイルがそのディレクトリ内に存在するかどうかを確認します。

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

VALIDATE メソッドの構文解析関数

RGM は、他のコールバックメソッドとは異なるパラメータを VALIDATE メソッドに渡します。したがって、VALIDATE メソッドには、他のメソッドとは異なる引数を構文解析する関数が必要です。VALIDATE メソッドや他のコールバックメソッドに渡される引数についての詳細は、rt_callbacks(1HA) のマニュアルページを参照してください。次に、VALIDATE メソッドの parse_args 関数を示します。

```
##############################################################################
# Validate 引数を構文解析する。
#
function parse_args # [args ...]
{
 typeset opt
 while getopts 'cur:x:g:R:T:G:' opt
 do
      case "$opt" in
      R)
         # DNS リソースの名前
```

(続く)

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

```
        # Confdir 拡張プロパティがコマンド行
        # 上に存在する場合、その値を記録する。
        if [ $PROPERTY == "Confdir" ]; then
            CONFDIR=$VAL
            CONFDIR_FOUND=1
        fi
        ;;
    *)
```

(続く)

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

他のメソッドの 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。

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

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

Confdir の妥当性検査

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

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

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

```
parse_args ``$@''
```

次に、VALIDATE は、VALIDATE メソッドがプロパティの更新の結果として呼び出されているのかどうか、および、Confdir 拡張プロパティがコマンド行上に存在するかどうかを検査します。次に、VALIDATE メソッドは、Confdir プロパティが値を持っているかどうかを確認します。値を持っていない場合、VALIDATE メソッドはエラーメッセージを記録し、失敗状態で終了します。

```
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
   config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
    -G $RESOURCEGROUP_NAME Confdir`
   CONFDIR=`echo $config_info | awk '{print $2}'`
fi
# Confdir プロパティが値を持っているかどうかを確認する。持っていな
# い場合、状態 1 (失敗) で終了する。
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
    "${ARGV0}: Validate method for resource "$RESOURCE_NAME " failed"
   exit 1
fi
```

注 - 上記コードにおいて、VALIDATE メソッドが更新の結果として呼び出されているのか ($UPDATE_PROPERTY == 1)、および、プロパティがコマンド行上に存在しないのか (CONFDIR_FOUND == 0) を検査し、両者が TRUE である場合に、scha_resource_get 関数を使用して Confdir の既存の値を取得するところに注目してください。Confdir がコマンド行上に存在する (CONFDIR_FOUND == 1) 場合、*CONFDIR* の値は、scha_resource_get 関数からではなく、parse_args 関数から取得されます。

次に、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。アクセス可能ではない場合、VALIDATE メソッドはエラーメッセージを記録し、エラー状態で終了します。

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

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

```
# named.conf ファイルが Confdir ディレクトリ内に存在するかどうかを
# 検査する。
if [ ! -s $CONFDIR/named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
    "${ARGV0}: File $CONFDIR/named.conf is missing or empty"
   exit 1
fi
```

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

```
# Validate メソッドが成功したことを示すメッセージを記録する。
logger -p ${SYSLOG_FACILITY}.err \
 -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
 "${ARGV0}: Validate method for resource "$RESOURCE_NAME \
 " completed successfully"
exit 0
```

VALIDATE の終了状態

VALIDATE メソッドが成功 (0) で終了すると、新しい値を持つ Confdir が作成されます。VALIDATE メソッドが失敗 (1) で終了すると、Confdir を含むすべてのプロパティが作成されず、理由を示すメッセージがクラスタ管理者に送信されます。

UPDATE メソッド

リソースのプロパティが変更されたとき、RGM は UPDATE メソッドを呼び出し、動作中のリソースにその旨を通知します。RGM は、管理アクションがリソースまたはリソースグループのプロパティの設定に成功した後に、UPDATE を呼び出します。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 の概要

UPDATE メソッドはプロパティを更新しません。プロパティの更新は RGM が行います。その代わりに、動作中のプロセスに更新が発生したことを通知します。サンプルのデータサービスでは、プロパティの更新によって影響を受けるプロセスは障害モニターだけです。したがって、UPDATE メソッドは、障害モニターを停止および再起動します。

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 で障害モニターを強制終了する必要があります。UPDATE メソッドは、障害モニターを実装する検証プログラムの位置を取得し、その後、もう一度 pmfadm で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

次に、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管理ユーザーに送信されます。障害モニターが停止できない場合、UPDATE メソッドは、エラーメッセージを管理ユーザーに送信し、失敗状態で終了します。

```
if pmfadm -q $RESOURCE_NAME.monitor; then
   # すでに動作している障害モニターを強制終了する。
   pmfadm -s $RESOURCE_NAME.monitor TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err \
        -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
        "${ARGV0}: Could not stop the monitor"
      exit 1
   else
      # DNS の停止に成功。メッセージを記録する。
      logger -p ${SYSLOG_FACILITY}.err \
        -t [$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 プロパティが指すディレクトリ) 内にありま

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管理ユーザーに送信し、成功で終了します。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 機能を通じて、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。

付録 **A**

標準プロパティ

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

この章の内容は次のとおりです。

注 - `True` や `False` などのプロパティ値は、大文字と小文字は区別されません。

リソースタイププロパティ

表 A–1 に、Sun Cluster によって定義されているリソースタイププロパティを示します。プロパティ値は、以下のように分類されます (分類の列)。

- 必須 — Resource Type Registration (RTR) ファイル内に利用値を必要とするプロパティです。値がない場合は、プロパティが属するオブジェクトを作成できません。ブランク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 — このプロパティが存在するためには、RTR ファイル内で宣言する必要があります。宣言されていない場合は、RGM はこのプロパティを作成しないため、管理ユーティリィティで利用できません。ブランクまたは空の文字列を値と

して指定できます。プロパティがRTR ファイル内で宣言されており、値が指定されていない場合には、RGM はデフォルト値を使用します。

- 条件付/明示 — このプロパティが存在するためには、明示的に値を指定して、RTR ファイル内で宣言する必要があります。宣言されていない場合は、RGM はこのプロパティを作成しないため、管理ユーティリィティで利用できません。ブランク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 — プロパティを RTR ファイル内で宣言できます。宣言しない場合は、RGM はこのプロパティを作成し、デフォルト値を使用します。プロパティがRTR ファイル内で宣言されており、値が指定されていない場合は、RGM は、プロパティが RTR ファイル内で宣言されないときのデフォルト値と同じ値を使用します。

リソースタイププロパティは、Installed_nodes を除き、管理ユーティリィティによって更新することができません。Installed_nodes は、RTR ファイル内で宣言できないため、管理者が設定する必要があります。

表 A–1 リソースタイププロパティ

プロパティ名	説明	更新の可否	分類
API_version (整数)	このリソースタイプの実装によって使用されるリソース管理 API のバージョン。 SC 3.0 のデフォルトは 2 です。	不可	任意
BOOT (文字列)	任意のコールバックメソッド。ノード上で RGM が呼び出すプログラムへのパス。このプログラムは、このタイプのリソースがすでに管理状態にあるときに、クラスタの結合または再結合を行います。このメソッドは、INIT メソッドと同様に、このタイプのリソースに対して、初期化アクションを行う必要があります。	不可	条件付/明示
Failover (ブール値)	True は、複数のノード上で同時にオンラインになることのできる任意のグループで、このタイプのリソースを構成できないことを示します。デフォルトは、False です。	不可	任意

表 A–1 リソースタイププロパティ 続く

プロパティ名	説明	更新の可否	分類
FINI (文字列)	任意のコールバックメソッド。RGM 管理からこのタイプのリソースを削除するときに RGM が呼び出すプログラムへのパス。	不可	条件付/明示
INIT (文字列)	任意のコールバックメソッド。このタイプのリソースが RGM によって管理されるようになったときに、RGM が呼び出すプログラムへのパス。	不可	条件付/明示
Init_nodes (列挙)	値には、RG_primaries (リソースをマスターできるノードだけ)、または RT_installed_nodes (リソースタイプがインストールされるすべてのノード) を指定できます。 RGM が INIT、FINI、BOOT、VALIDATE メソッドをコールするノードを示します。 デフォルト値は、RG_primaries です。	不可	任意
Installed_nodes (文字配列)	リソースタイプの実行が許可されるクラスタノード名のリスト。RGM は、自動的にこのプロパティを作成します。クラスタ管理者は値を設定できます。このプロパティは、RTR ファイル内で宣言できません。 デフォルトは、すべてのクラスタノードです。	可	クラスタ管理者は構成可能
Monitor_check (文字列)	任意のコールバックメソッド。このタイプのリソースの障害モニターが要求するフェイルオーバーを行う前に、RGM が呼び出すプログラム。	不可	条件付/明示
Monitor_start (文字列)	任意のコールバックメソッド。このタイプのリソースの障害モニターを起動するために、RGM が呼び出すプログラムへのパス。	不可	条件付/明示

表 A–1 リソースタイププロパティ 続く

プロパティ名	説明	更新の可否	分類
Monitor_stop (文字列)	Monitor_start が設定されている場合の、必須のコールバックメソッド。このタイプのリソースの障害モニターを停止するために、RGM が呼び出すプログラムへのパス。	不可	条件付/明示
Pkglist (文字配列)	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。	不可	条件付/明示
Postnet_stop (文字列)	任意のコールバックメソッド。このタイプのリソースが依存する任意のネットワークアドレスリソース (Network_resources_used) の STOP メソッドを呼び出した後で、RGM が呼び出すプログラムへのパス。ネットワークインタフェースが停止に構成された後に必要な STOP アクションを行う必要があります。	不可	条件付/明示
Prenet_start (文字列)	任意のコールバックメソッド。このタイプのリソースが依存する、任意のネットワークアドレスリソース (Network_resources_used) の START メソッドを呼び出す前に、RGM が呼び出すプログラムへのパス。ネットワークインタフェースが起動に構成された後に必要な START アクションを行う必要があります。	不可	条件付/明示
RT_basedir (文字列)	コールバックメソッドの相対パスを補うために使用するディレクトリパス。このパスは、リソースタイプパッケージのインストール場所に設定します。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。すべてのメソッドパス名が絶対パスの場合には、指定する必要はありません。	不可	必須 (絶対パスでないメソッドパスがある場合)

表 A–1 リソースタイププロパティ 続く

プロパティ名	説明	更新の可否	分類
RT_description (文字列)	リソースタイプの簡単な説明。 デフォルトは、空の文字列です。	不可	条件付
Resource_type (文字列)	リソースタイプの名前。クラスタのインストールにおいて一意でなければなりません。このプロパティは、RTR ファイルの最初のエントリで宣言される必要があります。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。 さらに、リソースタイプを識別するために、Vendor_id を指定できます。Vendor_id とリソースタイプ名は、ピリオドで区切られます (例: SUNW.http)。リソースタイプは、Resource_type と Vendor_id で完全に指定することも、Vendor_id を省略することもできます。たとえば、SUNW.http と http は、両方とも有効です。Vendor_id を指定する場合は、リソースタイプを定義する会社の株式銘柄を使用してください。クラスタ内で Vendor_id のみが異なるリソースタイプがある場合は、名前を省略できません。 デフォルトは空の文字列です。	不可	必須
RT_version (文字列)	このリソースタイプを実装する任意のバージョン文字列。	不可	条件付/明示
Single_instance (ブール値)	True の場合は、このタイプのリソースがクラスタ内に 1 つだけ存在できることを指定します。したがって、RGM は、同時に 1 つのこのリソースタイプだけに、クラスタ全体に渡っての実行を許可します。 デフォルト値は、False です。	不可	任意

表 A–1 リソースタイププロパティ 続く

プロパティ名	説明	更新の可否	分類
START (文字列)	コールバックメソッド。このタイプのリソースを開始するために RGM が呼び出すプログラムへのパス。	不可	必須 (RTR ファイルで PRENET_START メソッドが宣言されていない場合)
STOP (文字列)	コールバックメソッド。このタイプのリソースを停止するために RGM が呼び出すプログラムへのパス。	不可	必須 (RTR ファイルで POSTNET_STOP メソッドが宣言されていない場合)
UPDATE (文字列)	任意のコールバックメソッド。実行中のこのタイプのリソースのプロパティが変更された場合に、RGM が呼び出すプログラムへのパス。	不可	条件付/明示
VALIDATE (文字列)	任意のコールバックメソッド。このタイプのリソースのプロパティ値を検査するために呼び出すプログラムへのパス。	不可	条件付/明示
Vendor_ID (文字列)	Resource_type を参照してください。	不可	条件付

リソースプロパティ

表 A–2 に、Sun Cluster によって定義されているリソースプロパティを示します。プロパティ値は、以下のように分類されます (分類の列)。

- 必須 — 管理者は、管理ユーティリィティでリソースを作成するときに、必ず値を指定する必要があります。
- 任意 — 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、システムがデフォルト値を提供します。

- 条件付 — プロパティが RTR ファイルで宣言されている場合にのみ、RGM がプロパティを作成します。宣言されていない場合は、プロパティは存在せず、システム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のプロパティは、デフォルト値が RTR ファイル内で指定されているかどうかによって、必須または任意になります。詳細は、各条件付プロパティの説明を参照してください。
- 照会のみ — 管理ツールから直接設定できません。

表 A–2 は、リソースプロパティが更新可能かどうか、また、いつ更新できるかも示しています。

None または False	更新不可
True または Anytime	任意の時点
At_creation	リソースをクラスタに追加するとき
When_disabled	リソースを無効にするとき

表 A–2　リソースプロパティ

プロパティ名	説明	更新	分類
Cheap_probe_interval (整数)	リソースの即時障害検証の呼び出しの間隔 (秒数)。このプロパティは、RGM のみが作成でき、RTR ファイル内で宣言されている場合は、管理者は利用できます。 デフォルト値が RTR ファイル内で指定されている場合は、このプロパティは任意です。リソースタイプファイル内で、Tunable 属性が指定されていない場合は、プロパティの Tunable 値は、When_disabled (無効にするとき) になります。 Default 属性が RTR ファイル内のプロパティ宣言で指定されていない場合は、このプロパティは必須です。	無効にするとき	条件付
拡張プロパティ	そのリソースのタイプの RTR ファイルで宣言される拡張プロパティ。リソースタイプの実装によって、これらのプロパティを定義します。拡張プロパティに設定可能な各属性については、表 A–4 を参照してください。	特定のプロパティに依存	条件付

表 A–2 リソースプロパティ　続く

プロパティ名	説明	更新	分類
Failover_mode (列挙)	リソースでの START または STOP メソッドの呼び出しの失敗に対して、RGM がリソースグループを再配置するか、またはノードを異常終了させるかを制御します。None は、RGM が単にリソース状態をメソッド失敗に設定し、システム管理者の介入を待つことを示します。 Soft は、START メソッドが失敗したときに、RGM がリソースのグループを別のノードに再配置し、また、STOP メソッドが失敗したときに、RGM がリソース状態を設定し、システム管理者の介入を待つことを示します。 Hard は、START メソッドが失敗したときに、グループの再配置を行い、STOP メソッドが失敗したときに、クラスタノードを異常終了させることで、リソースの強制的な停止を行うことを示します。 デフォルトは、None です。	任意の時点	任意

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
Load_balancing_policy (文字列)	使用する負荷均衡ポリシーを定義する文字列。このプロパティは、スケーラブルサービスに対してのみ使用します。Scalable プロパティが RTR ファイルで宣言されている場合は、RGM は自動的にこのプロパティを作成します。 Load_balancing_policy は、次の値をとることができます。 Lb_weighted (デフォルト) — Load_balancing_weights プロパティで設定されているウエイトに従って、さまざまなノードに負荷が分散されます。 Lb_sticky — スケーラブルサービスの指定のクライアント (クライアントの IP アドレスで識別される) は、常に同じクラスタノードに送信されます。 Lb_sticky_wild — 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。 デフォルト値は、Lb_weighted です。	作成時	条件付/任意

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
`Load_balancing_weights` (文字配列)	このプロパティは、スケーラブルサービスに対してのみ使用します。`Scalable` プロパティが RTR ファイルで宣言されている場合は、RGM は自動的にこのプロパティを作成します。形式は、「*weight@node,weight@node*」になります。ここで、*weight* は、指定したノード (*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 は 3/4 を受け取ることを示します。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。 `Tunable` 属性がリソースタイプファイルに指定されていない場合は、プロパティの `Tunable` 値は `Anytime` (任意の時点) になります。このプロパティを変更すると、新しい接続時にのみ分散が変更されます。 デフォルト値は、空の文字列 ("") です。	任意の時点	条件付/任意
リソースタイプの各コールバックメソッドの *method*`_timeout` (整数)	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)。 メソッド自身が RTR ファイルで宣言されている場合、デフォルトは、3,600 秒 (1 時間) です。	任意の時点	条件付/任意

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
Monitored_ switch (列挙)	クラスタ管理者が管理ユーティリィティを使用し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 に設定されると、再び有効に設定されるまで、モニターは START メソッドを呼び出しません。リソースが、モニターのコールバックメソッドを持っていない場合は、このプロパティは存在しません。 デフォルトは Enabled です。	不可	照会のみ
Network_ resources_ used (文字配列)	リソースが使用す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。スケーラブルサービスの場合、このプロパティは別のリソースグループに存在する共有アドレスリソースを参照する必要があります。フェイルオーバーサービスの場合、このプロパティは同じリソースグループに存在する論理ホスト名または共有アドレスを参照します。Scalable プロパティが RTR ファイルで宣言されている場合、RGM は自動的にこのプロパティを作成します。Scalable が RTR ファイルで宣言されていない場合、Network_resources_used は RTR ファイルで明示的に宣言されていない限り使用できません。 Tunable 属性がリソースタイプファイルに指定されていない場合は、プロパティの Tunable 値は、 At_creation (作成時) になります。	作成時	条件付/必須

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
On_off_switch (列挙)	クラスタ管理者が管理ユーティリィティを使用し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。無効に設定されると、再び有効に設定されるまで、リソースはコールバックを呼び出しません。 デフォルトは、Disabled です。	不可	照会のみ
Port_list (文字配列)	サーバーが待機するポート番号をコンマで区切ったリスト。各ポート番号に、そのポートが使用しているプロトコルが追加されます (例: Port_list=80/tcp)。Scalable プロパティが RTR ファイルで宣言されている場合、RGM は自動的に Port_list を作成します。それ以外の場合、このプロパティは RTR ファイルで明示的に宣言されていない限り使用できません。 Apache 用にこのプロパティを設定する場合は、このマニュアルの Apache に関する章を参照してください。	作成時	条件付/必須
R_description (文字列)	リソースの簡単な説明。 デフォルトは、空の文字列です。	任意の時点	任意
Resource_dependencies (文字配列)	このリソースをオンラインにするために、順にオンラインにする必要のある同じグループ内のリソースのリスト。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。グループをオフラインにすると、このリソースを停止してから、リスト内のリソースが停止されます。このリソースが先に無効にならなければ、リスト内のリソースは無効にできません。 デフォルトは、空のリストです。	任意の時点	任意

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
Resource_ dependencies_ weak (文字配列)	グループ内のメソッド呼び出しの順序を決定する同じグループ内のリソースのリスト。RGM は、このリスト内のリソースの START メソッドを先に呼び出してから、このリソースの START メソッドを呼び出します。また、停止する場合は、このリソースの STOP メソッドを先に呼び出してから、リスト内のリソースの STOP メソッドを呼び出します。リスト内のリソースが開始に失敗した場合、または無効になっても、リソースはオンラインを維持できます。 デフォルトは、空のリストです。	任意の時点	任意
Resource_name (文字列)	リソースインスタンスの名前。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後で変更はできません。	不可	必須
各クラスタノードの Resource_state (列挙)	RGM が判断した各クラスタノード上のリソースの状態。可能な状態は次のとおりです。 Online、Offline、Stop_failed、Start_failed、Monitor_failed、Online_not_monitored、Detached。 このプロパティは、ユーザーが構成することはできません。	不可	照会のみ

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
Retry_count (整数)	リソースの起動に失敗した場合に、モニターが再起動を試みる試行回数。このプロパティは、RGM のみが作成でき、RTR ファイルで宣言されている場合は、管理者は利用できます。デフォルト値が RTR ファイルで指定されている場合は、このプロパティは任意です。 リソースタイプファイル内で Tunable 属性が指定されていない場合は、プロパティの Tunable 値は、When_disabled (無効化にするとき) になります。 Default 属性が RTR ファイルのプロパティ宣言に指定されていない場合は、このプロパティは必須です。	無効にする	条件付
Retry_interval (整数)	失敗したリソースを再起動する回数をカウントする間隔 (秒)。リソースモニターは、Retry_count と共にこのプロパティを使用します。このプロパティは、RGM のみが作成でき、RTR ファイルで宣言されている場合は、管理者は利用できます。デフォルト値が RTR ファイルで指定されている場合は、このプロパティは任意です。 リソースタイプファイル内で Tunable 属性が指定されていない場合は、プロパティの Tunable 値は、When_disabled (無効化にするとき) になります。 Default 属性が RTR ファイルのプロパティ宣言に指定されていない場合は、このプロパティは必須です。	無効にする	条件付

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
`Scalable` (ブール値)	リソースがスケーラブルかどうかを示します。このプロパティが RTR ファイルで宣言されている場合は、そのタイプのリソースに対して、RGM は、次のスケーラブルサービスを自動的に作成します。 `Network_resources_used`、`Port_list`、`Load_balancing_policy`、`Load_balancing_ weights` 。 これらのプロパティは、RTR ファイル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。 このプロパティが RTR ファイルで宣言されている場合、`Tunable` 属性は、`At_creation` (作成時) に設定する必要があります。設定しなければ、リソースの生成に失敗します。 このプロパティが RTR ファイルで宣言されていない場合、リソースはスケーラブルにはなりません。したがって、クラスタ管理者はこのプロパティを調整することができず、RGM はスケーラブルサービスプロパティを設定しません。ただし、必要に応じて、明示的に `Network_resources_used` および `Port_list` プロパティを RTR ファイルで宣言できます。これらのプロパティは、スケーラブルサービスだけでなく、非スケーラブルサービスでも有用です。	作成時	任意

表 A–2 リソースプロパティ 続く

プロパティ名	説明	更新	分類
各クラスタノードの Status (列挙)	リソースモニターによって設定されます。指定可能な値は、 degraded、faulted、unknown、 offline です。 RGM は、リソースがオンラインになると、値を unknown に設定し、オフラインになると offline に設定します。	不可	照会のみ
各クラスタノードの Status_msg (文字列)	リソースモニターによって、Status プロパティと同時に設定されます。このプロパティは、各ノードのリソースごとに設定可能です。RGM は、リソースがオフラインになると、このプロパティに空の文字列を設定します。	不可	照会のみ
Thorough_probe_interval (整数)	高オーバーヘッドのリソース障害検証の呼び出し間隔 (秒)。このプロパティは、RGM のみが作成でき、RTR ファイルで宣言されている場合は、管理者は利用できます。デフォルト値が RTR ファイルで指定されている場合は、このプロパティは任意です。 リソースタイプファイル内で Tunable 属性が指定されていない場合は、プロパティの Tunable 値は、When_disabled (無効化にするとき) になります。 Default 属性が RTR ファイルのプロパティ宣言に指定されていない場合は、このプロパティは必須です。	無効にする	条件付
Type (文字列)	このリソースがインスタントであるリソースタイプ。	不可	必須

リソースグループプロパティ

表 A–3 に、Sun Cluster によって定義されたリソースグループプロパティを示します。プロパティ値は、以下のように分類されます (分類の列)。

- 必須 — 管理者は、管理ユーティリィティでリソースグループを作成するときに、必ず値を指定する必要があります。
- 任意 — 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、システムがデフォルト値を提供します。
- 照会のみ— 管理ツールから直接設定できません。

更新の可否の列は、初期設定後に、そのプロパティが更新可能 (Y) なのか、更新できない (N) のかを示しています。

表 A–3 リソースグループプロパティ

プロパティ名	説明	更新の可否	分類
`Desired_primaries` (整数)	グループが同時にオンラインになることができるノードの数。 デフォルトは 1 です。`RG_mode` プロパティが `Failover` の場合、このプロパティの値を 1 より大きく設定することはできません。`RG_mode` プロパティが `Scalable` の場合は、1 より大きな値を設定できます。	可	任意
`Failback` (ブール値)	クラスタメンバーシップが変更されたとき、グループがオンラインになるノードセットを再計算するかどうかを指定するブール値。再計算によって、RGM はグループを優先度の低いノードでオフラインにし、優先度の高いノードでオンラインにします。 デフォルトは、`False` です。	可	任意
`Global_resources_used` (文字配列)	クラスタファイルシステムがこのリソースグループで任意のリソースに使用されるかどうかを示します。管理者は、すべての広域リソース (アスタリスク記号 `*`) または広域リソースなし (空の文字列 `""`) に指定できます。 デフォルトでは、すべての広域リソースです。	可	任意

表 A–3 リソースグループプロパティ 続く

プロパティ名	説明	更新の可否	分類
Implicit_ network_ dependencies (ブール値)	True の場合に、グループ内のネットワークアドレスリソースに対し、非ネットワークアドレスリソースの暗黙の強い依存性を RGM が強制することを指定するブール値。ネットワークアドレスリソースには、論理ホスト名と共有アドレスリソースタイプが含まれます。 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、このプロパティは効果がありません。 デフォルトは、True です。	可	任意
Maximum_ primaries (整数)	グループが同時にオンラインになることのできるノードの最大数。 デフォルトは 1 です。RG_mode プロパティが Failover の場合、このプロパティの値を 1 より大きく設定することはできません。RG_mode プロパティが Scalable の場合は、1 より大きな値を設定できます。	可	任意
Nodelist (文字配列)	優先順位に従ってグループをオンラインにできるクラスタノードのリスト。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。 デフォルトは、すべてのクラスタノードのリストになります。	可	任意
Pathprefix (文字列)	グループ内のリソースが書き込めるクラスタファイルシステムにあるディレクトリは、重要な管理ファイルを書き込めます。一部のリソースでは、このプロパティは必須です。各リソースグループの Pathprefix は、一意にする必要があります。 デフォルトは、空の文字列です。	可	任意

表 A–3 リソースグループプロパティ 続く

プロパティ名	説明	更新の可否	分類
Pingpong_ interval (整数)	再構成が生じた場合、scha_control giveover コマンドの実行結果、あるいは実行されている機能によって、どのノードでリソースグループをオンラインにするかを判断するときに RGM が使用する負以外の整数値 (秒)。 再構成において、リソースの START または PRENET_START メソッドがゼロ以外の値で終了、またはタイムアウトによって終了したことが原因で、Pingpong_interval で指定した秒数内に、リソースグループをオンラインにするのを 2 回以上失敗した場合、RGM はそのノードはリソースグループのホストとして不適切だと判断し、別のマスターを捜します。 リソースの scha_control(1ha)(3ha) コマンドまたは機能の呼び出しによって、Pingpong_interval で指定した秒数内に特定のノード上でリソースグループがオフラインになった場合、別のノードから生じる後続の scha_control 呼び出しの結果、そのノードはリソースグループのホストとして不適切だと判断されます。 デフォルト値は、3,600 秒 (1 時間) です。	可	任意
Resource_ list (文字配列)	グループに含まれるリソースのリスト。管理者はこのプロパティを直接設定しません。このプロパティは、管理者がリソースグループにリソースを追加したり、リソースを削除したときに、RGM によって更新されます。 デフォルトは、空のリストです。	不可	照会のみ
RG_ dependencies (文字配列)	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)。別のノードでグループをオンラインにする場合は、このリストは無効です。 デフォルトは、空のリストです。	可	任意
RG_ description (文字列)	リソースグループの簡単な説明。 デフォルトは空の文字列。	可	任意

表 A–3 リソースグループプロパティ 続く

プロパティ名	説明	更新の可否	分類
RG_mode (列挙)	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。このプロパティの値が Failover の場合、RGM はグループの Maximum_primaries プロパティを 1 に設定し、そのリソースグループをマスターするのを単一のノードに制限します。 このプロパティの値が Scalable の場合、RGM は Maximum_primaries プロパティが 1 より大きい値を持つことを許可し、複数のノードで同時にそのグループをマスターできるようにします。RGM は、RG-mode が Scalable に設定されているリソースグループに、Failover プロパティが True に設定されているリソースを追加することを許可しません。 Maximum_primaries に 1 が設定されている場合のデフォルトは、Failover です。Maximum_primaries に 2 以上が設定されている場合のデフォルトは、Scalable です。	不可	任意
RG_name (文字列)	リソースグループの名前。クラスタ内で一意にする必要があります。	不可	必須
各クラスタノードの RG_state (列挙)	RGM によって Online、Offline、Pending_online、Pending_offline、Error_stop_failed に設定され、各クラスタノード上のグループの状態を示します。グループが RGM の制御下にない場合は、非管理状態で存在できます。 このプロパティは、ユーザーは構成できません。 デフォルトは、Offline です。	不可	照会のみ

リソースプロパティの属性

表 A–4 に、システム定義プロパティの変更または拡張プロパティの作成に使用できるリソースプロパティの属性を示します。

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。

表 A–4 リソースプロパティの属性

プロパティ	説明
Property	リソースプロパティの名前。
Extension	このプロパティを使用すると、RTR ファイルのエントリで、リソースタイプの実装によって定義された拡張プロパティが宣言されていることを示します。使用されない場合は、そのエントリはシステム定義プロパティです。
Description	プロパティを簡潔に記述した注記 (文字列)。RTR ファイル内でシステム定義プロパティに対する Description 属性を設定することはできません。
プロパティのタイプ	指定可能なタイプは、string、boolean、int、enum、stringarray です。RTR ファイル内で、システム定義プロパティに対するタイプ属性を設定することはできません。タイプは、RTR ファイルのエントリに登録できる、指定可能なプロパティ値とタイプ固有の属性を決定します。enum タイプは、文字列値のセットです。
Default	プロパティのデフォルト値を示します。
Tunable	クラスタ管理者が、リソースのプロパティ値をいつ設定できるかを示します。管理者がプロパティを設定できないようにするには、None または False に設定します。管理者にプロパティの調整を許可する属性値は、次のとおりです。True または Anytime (任意の時点)、At_creation (リソースの作成時のみ)、When_disabled (リソースがオフラインのとき)。 デフォルトは、 True (Anytime) です。
Enumlist	enum タイプの場合、プロパティに設定できる文字列値のセット。
Min	int タイプの場合、プロパティに設定できる最小値。

表 A–4 リソースプロパティの属性 続く

プロパティ	説明
Max	int タイプの場合、プロパティに設定できる最大値。
Minlength	string および stringarray タイプの場合、設定できる文字列の最小長。
Maxlength	string および stringarray タイプの場合、設定できる文字列の最大。
Array_minsize	stringarray タイプの場合、設定できる配列要素の最小数。
Array_maxsize	stringarray タイプの場合、設定できる配列要素の最大数。

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 (RTR) ファイルの内容も示します。

この付録に含まれるコードリストは、次のとおりです。

リソースタイプ登録ファイルのリスト

リソースタイプ登録 (RTR) ファイルには、クラスタ管理者が Sun Cluster 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するリソースとリソースタイプのプロパティ宣言が含まれています。

例 **B–1** SUNW.Sample RTR ファイル

```
#
# Copyright (c) 1998-2000 by Sun Microsystems, Inc.
# All rights reserved.
#
# ドメインネームサービス (DNS) の登録情報
#


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

RESOURCE_TYPE = ''sample'';
VENDOR_ID = SUNW;
RT_DESCRIPTION = ''Domain Name Service on Sun Cluster'';

RT_VERSION =''1.0'';
API_VERSION = 2;
FAILOVER = TRUE;

RT_BASEDIR=/opt/SUNWsample/bin;
PKGLIST = SUNWsample;

START      =  dns_svc_start;
STOP      =  dns_svc_stop;

VALIDATE        = dns_validate;
UPDATE        = dns_update;

MONITOR_START     = dns_monitor_start;
MONITOR_STOP       = dns_monitor_stop;
MONITOR_CHECK       = dns_monitor_check;

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

# <method>_timeout プロパティは、RGM がメソッドの呼び出しが失敗
# したという結論を下すまでの時間 (秒) を設定する。

# すべてのメソッドタイムアウトの MIN 値は 60 秒に設定されている。こ
# れは、管理者が短すぎる時間を設定することを防ぐためである。短すぎ
# る時間を設定すると、スイッチオーバーやフェイルオーバーの性能が上
# がらず、さらには、予期せぬ RGM アクションが発生する可能性がある
```

(続く)

```
# (間違ったフェイルオーバー、ノードの再起動、リソースグループの
# ERROR_STOP_FAILED 状態への移行、オペレータの介入の必要性など)。
# メソッドタイムアウトに短すぎる時間を設定すると、データサービス全
# 体の可用性を下げることになる。
{
 PROPERTY = Start_timeout;
 MIN=60;
 DEFAULT=300;
}

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

# 当該ノード上でアプリケーションを正常に起動できないと結論を下すま
# でに、ある期間 (Retry_Interval) に行う再試行の回数
{
        PROPERTY = Retry_Count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}
# Retry_Interval には 60 の倍数を設定する。これは、秒から分に変換さ
# れ、端数が切り上げられるためである。たとえば、50 (秒) という値を指
```

(続く)

```
# 定すると、1 分に変換される。
# このプロパティは再試行数 (Retry_Count) のタイミングを決定する。
{
        PROPERTY = Retry_Interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

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

#
# 拡張プロパティ
#
# クラスタ管理者はこのプロパティの値を設定して、アプリケーションが使用
# する構成ファイルが入っているディレクトリを示す必要がある。このアプリ
# ケーションの場合、DNS は PXFS (通常は named.conf) 上の DNS 構成ファイ
# ルのパスを指定する。
{
 PROPERTY = Confdir;
 EXTENSION;
 STRING;
 TUNABLE = AT_CREATION;
 DESCRIPTION = ''The Configuration Directory Path'';
}

# 検証が失敗したと宣言するまでのタイムアウト値 (秒)
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 30;
        TUNABLE = ANYTIME;
        DESCRIPTION = ''Time out value for the probe (seconds)'';
}
```

START メソッドのコードリスト

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るとき、あるいは、リソースが有効になるとき、RGM はそのノード上で START メソッドを呼び出します。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
#
# HA-DNS の START メソッド
#
# このメソッドは PMF の制御下でデータサービスを起動する。DNS の
# in.named プロセスを起動する前に、いくつかの妥当性検査を実行する。
#
# データサービスの PMF タグは $RESOURCE_NAME.named である。PMF は、
# 指定された回数 (Retry_count) だけ、サービスを起動しようとする。そ
# して、指定された期間 (Retry_interval) 内で試行回数がこの値を超えた
# 場合、PMF はサービスの起動に失敗したことを報告する。
# Retry_count と Retry_interval は両方とも RTR ファイルに設定されて
# いるリソースプロパティである。
#pragma ident ``@(#)dns_svc_start 1.1 00/05/24 SMI''
#############################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ``$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
```

(続く)

```
                    ``ERROR: Option $OPTARG unknown''
                     exit 1
                     ;;

                esac
        done

}



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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

PMF_TAG=$RESOURCE_NAME.named

# DNS を起動するため、リソースの Confdir プロパティの値を取得する。入
# 力されたリソース名とリソースグループを使用して、リソースを追加するときにクラ
# スタ管理者が設定した Confdir の値を見つける。


config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get は拡張プロパティの「タイプ」と「値」を戻す。拡
# 張プロパティの値だけを取得する。
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

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

# データファイルへの相対パス名が存在する場合、$CONFIG_DIR ディレク
# トリに移動する。
cd $CONFIG_DIR

# named.conf ファイルが $CONFIG_DIR ディレクトリ内に存在するかどうか
# を検査する。
if [ ! -s named.conf ]; then
 logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
     ``${ARGV0} File $CONFIG_DIR/named.conf is missing or
```

(続く)

```
empty''
 exit 1
fi

# RTR ファイルから Retry_count の値を取得する。
RETRY_CNT=scha_resource_get -O Retry_Count -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME`

# RTR ファイルから Retry_interval の値を取得する。この値の単位は秒
# であり、pmfadm に渡すときは分に変換する必要がある。変換時、端数は
# 切り捨てられるので注意すること。たとえば、59 秒は 0 分に切り捨て
# られる。

((RETRY_INTRVAL =`scha_resource_get -O Retry_Interval
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME` / 60))

# PMF の制御下で in.named デーモンを起動する。$RETRY_INTERVAL の期
# 間、$RETRY_COUNT の回数だけ、クラッシュおよび再起動できる。どちら
# かの値以上クラッシュした場合、PMF は再起動をやめる。
# <$RESOURCE_NAME.named> というタグですでにプロセスが登録されて
# いる場合、PMF はすでにプロセスが動作していることを示す警告メッセ
# ージを送信する。
#
echo ``Retry interval is ``$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# HA-DNS が起動していることを示すメッセージを記録する。
if [ $? -eq 0 ]; then
 logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
         ``${ARGV0} HA-DNS successfully started''
fi
exit 0
```

STOP メソッドのコードリスト

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るとき、あるいは、HA-DNS リソースが無効になるとき、RGM は STOP メソッドを呼び出します。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
#
# HA-DNS の STOP メソッド
#
# このメソッドは、PMF を使用するデータサービスを停止する。サービス
# が動作していない場合、このメソッドは状態 0 で終了する。その他の値
# は戻さない。リソースは STOP_FAILED 状態になる。
#pragma ident ''@(#)dns_svc_stop 1.1 00/05/24 SMI''
################################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ''$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    ''ERROR: Option $OPTARG unknown''
                     exit 1
                     ;;

                esac
        done

}


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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY
```

(続く)

```
# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

PMF_TAG=$RESOURCE_NAME.named

# RTR ファイルから Stop_timeout 値を取得する。
STOP_TIMEOUT=scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME

# PMF 経由で SIGTERM シグナルを使用する規則正しい方法でデータサービ
# スを停止しようとする。SIGTERM がデータサービスを停止できるまで、
# Stop_timeout 値の 80% だけ待つ。停止できない場合、SIGKILL を送信
# して、データサービスを停止しようとする。SIGKILL がデータサービス
# を停止できるまで、Stop_timeout 値の 15% だけ待つ。停止できない場
# 合、メソッドは何か異常があったと判断し、0 以外の状態で終了する。
# Stop_timeout の残りの 5% は他の目的のために予約されている。
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

# in.named が動作しているかどうかを調べて、動作していれば停止する。
if pmfadm -q $PMF_TAG.named; then
 # SIGTERM シグナルをデータサービスに送信して、合計タイムアウト値
 # の 80% だけ待つ。
 pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
 if [ $? -ne 0 ]; then
  logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
      ``${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry
with \
       SIGKILL''

  # SIGTERM シグナルでデータサービスが停止しないので、今度は
  # SIGKILL を使用して、合計タイムアウト値の 15% だけ待つ。
  pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
  if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG] \
      ``${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL''

      exit 1
  fi
 fi
else
 # この時点でデータサービスは動作していない。メッセージを記録して、
 # 成功で終了する。
 logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
         ``HA-DNS is not started''

 # HA-DNS が動作していない場合でも、成功で終了し、データサービス
 # リソースが STOP_FAILED 状態にならないようにする。

 exit 0

fi
```

(続く)

```
# DNS の停止に成功。メッセージを記録して、成功で終了する。
logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
    ''HA-DNS successfully stopped''
exit 0
```

gettime ユーティリティのコードリスト

gettime ユーティリティは、検証の再起動間の経過時間を PROBE プログラムが追跡するための C プログラムです。このプログラムは、コンパイル後、コールバックメソッドと同じディレクトリ (RT_basedir プロパティが指すディレクトリ) に格納する必要があります。

例 B–4 gettime.c ユーティリティプログラム

```
#
# このユーティリティプログラムは、データサービスの検証メソッドによ
# って使用され、既知の参照ポイント (基準点) からの経過時間 (秒) を
# 追跡する。このプログラムは、コンパイル後、データサービスのコール
# バックメソッドと同じディレクトリ (RT_basedir) に格納しておくこと。
#pragma ident ''@(#)gettime.c 1.1 00/05/24 SMI''

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

main()
{
    printf(''%d\n'', time(0));
    exit(0);
}
```

PROBE プログラムのコードリスト

PROBE プログラムは、nslookup(1M) 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。このプログラムは、MONITOR_START コールバックメソッドによって起動され、MONITOR_STOP コールバックメソッドによって停止されます。

例 **B–5** dns_probe プログラム

```
#!/bin/ksh
#pragma ident ''@(#)dns_probe 1.1 00/04/19 SMI''
#
# HA-DNS の Probe メソッド
#
# このプログラムは、nslookup を使用して、データサービスの可用性を検査
# する。nslookup は DNS サーバーに照会することによって、DNS
# サーバー自身を捜す。サーバーが応答しない場合、あるいは、別のサー
# バーが照会に応答した場合、probe メソッドはデータサービスまたはク
# ラスタ内の別のノードになんらかの問題が発生したという結論を下す。
# 検証は、RTR ファイルの THOROUGH_PROBE_INTERVAL プロパティで設定さ
# れた間隔で行われる。
#pragma ident ''@(#)dns_probe 1.1 00/05/24 SMI''
################################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ''$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    ''ERROR: Option $OPTARG unknown''
                     exit 1
```

(続く)

```
                        ;;

                esac
        done

}


#############################################################################
# restart_service ()
#
# この関数は、まずデータサービスの STOP メソッドを呼び出し、次に START メソッドを呼び出す
# ことによって、データサービスを再起動しようとする。データサービスがすでに起動しておらず、
# データサービスのタグが PMF 下に登録されていない場合、この関数はデータサービスをクラスタ内の
# 別のノードにフェイルオーバーする。
#
function restart_service
{

        # データサービスを再起動するには、まず、データサービス自身が PMF 下に登録されて
        # いるかどうかを確認する。
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # データサービスの TAG が PMF 下に登録されている場合、データサービスを
                # 停止し、起動し直す。

                # 当該リソースの STOP メソッド名と STOP_TIMEOUT 値を取得する。
                STOP_TIMEOUT≅scha_resource_get -O STOP_TIMEOUT
\
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_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

                # 当該リソースの START メソッド名と START_TIMEOUT 値を取得する。
                START_TIMEOUT≅scha_resource_get -O START_TIMEOUT
\
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                START_METHOD≅scha_resource_get -O START
\
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD
```

(続く)

```
\
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
\
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
\
                                ''${ARGV0} Start method
failed.''
                        return 1
                fi


        else
                # データサービスの TAG が PMF 下に登録されていない場合、
                # データサービスが PMF 下で許可されている再試行最大回数を
                # 超えていることを示す。したがって、 データサービスを再起動
                # してはならない。その代わりに、同じクラスタ内にある別のノード
                # にフェイルオーバーを試みる。
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME
\
                        -R $RESOURCE_NAME
        fi

        return 0
}




################################################################################
# decide_restart_or_failover ()
#
# この関数は、検証が失敗したときに行うべきアクション、つまり、デー
# タサービスをローカルで再起動するか、クラスタ内の別のノードにフェ
# イルオーバーするかを決定する。
#
function decide_restart_or_failover
{

 # 最初の再起動の試行であるかどうかを検査する。
 if [ $retries -eq 0 ]; then
  # 最初の障害である。最初の試行の時刻を記録する。
  #
  start_time≥$RT_BASEDIR/gettimè
  retries≥expr $retries + ì
  # 最初の失敗であるので、データサービスを再起動しようと試行する。
 restart_service
  if [ $? -ne 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       ''${ARGV0} Failed to restart data service.''
   exit 1
  fi
```

(続く)

```
 else
  # 最初の障害ではない。
  current_time=$RT_BASEDIR/gettime
  time_diff=expr $current_time - $start_time
  if [ $time_diff -ge $RETRY_INTERVAL ]; then
   # この障害は再試行最大期間後に発生した。
   # したがって、再試行カウンタをリセットし、
   # 再試行時間をリセットし、さらに再試行する。
   retries=1
    start_time=$current_time
   # 前回の失敗が Retry_interval よりも以前に発生しているので、
   # データサービスを再起動しようと試行する。
   restart_service
   if [ $? -ne 0 ]; then
    logger -p ${SYSLOG_FACILITY}.err \
        -t [$SYSLOG_TAG
        ``${ARGV0} Failed to restart HA-DNS.''
    exit 1
   fi
  elif [ $retries -ge $RETRY_COUNT ]; then
   # 再試行最大期間内であり、再試行カウンタは満了
   # している。したがって、フェイルオーバーする。
  retries=0
   scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
       -R $RESOURCE_NAME
   if [ $? -ne 0 ]; then
    logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
        ``${ARGV0} Failover attempt failed.''
    exit 1
   fi
  else
   # 再試行最大期間内であり、再試行カウンタは満了
   # していない。したがって、さらに再試行する。
  retries=expr $retries + 1
   restart_service
   if [ $? -ne 0 ]; then
    logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
        ``${ARGV0} Failed to restart HA-DNS.''
    exit 1
   fi
  fi
 fi
}


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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY
```

(続く)

```
# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

PMF_TAG=$RESOURCE_NAME.named

# 検証が行われる間隔はシステム定義プロパティ THOROUGH_PROBE_INTERVAL
# に設定されている。scha_resource_get でこのプロパティの値を取得する。
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAME

# 検証用のタイムアウト値を取得する。この値は RTR ファイルの
# PROBE_TIMEOUT 拡張プロパティに設定されている。nslookup のデフォル
# トのタイムアウトは 1.5 分。
probe_timeout_info=scha_resource_get -O Extension -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME Probe_timeout
PROBE_TIMEOUT=echo $probe_timeout_info | awk '{print $2}'

# リソースの NETWORK_RESOURCES_USED プロパティの値を取得することによっ
# て、DNS がサービスを提供するサーバーを見つける。
DNS_HOST=scha_resource_get -O NETWORK_RESOURCES_USED -R
$RESOURCE_NAME -G $RESOURCEGROUP_NAME

# システム定義プロパティ Retry_count から再試行最大回数を取得する。
RETRY_COUNT=scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME

# システム定義プロパティ Retry_interval から再試行最大期間を取得する。
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R
$RESOURCE_NAME -G $RESOURCEGROUP_NAME

# リソースタイプの RT_basedir プロパティから gettime ユーティリティの
# 完全パスを取得する。
RT_BASEDIR=scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME

# 検証は無限ループで動作し、nslookup コマンドを実行し続ける。
# nslookup 応答用の一時ファイルを設定する。
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
 # 検証が動作すべき期間は THOROUGH_PROBE_INTERVAL プロパティに指
 # 定されている。したがって、THOROUGH_PROBE_INTERVAL の間、検証
 # プログラムが休眠するように設定する。
 sleep $PROBE_INTERVAL

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

(続く)

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

 # nslookup への応答が HA-DNS サーバーから来ており、
 # /etc/resolv.conf ファイル内に指定されている他のネームサーバー
 # から来ていないことを確認する。
 if [ $probefail -eq 0 ]; then
  # nslookup 照会に応答したサーバーの名前を取得する。
                SERVER= awk ` $1==''Server:'' {
print $2 }' \
   $DNSPROBEFILE | awk -F. ` { print $1 } ``
                if [ -z ``$SERVER'' ];
then
                        probefail=1
                else
                        if [ $SERVER != $DNS_HOST ]; then
                                probefail=1
                        fi
                fi
        fi

 # probefail 変数が 0 以外である場合、nslookup コマンドがタイム
 # アウトしたか、あるいは、別のサーバー (/etc/resolv.conf ファイ
 # ルに指定されている) から照会への応答が来ていることを示す。ど
 # ちらの場合でも、DNS サーバーは応答していないので、このメソッ
 # ドは decide_restart_or_failover 関数を呼び出して、データサー
 # ビスをローカルで起動するか、あるいは、別のノードにフェイルオ
 # ーバーするかを評価する。

 if [ $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
#
# HA-DNS の Monitor_Start メソッド
#
# このメソッドは、PMF の制御下でデータサービスのモニター (検証) を
# 起動する。モニターは一定の間隔でデータサービスを検証するプロセス
# で、問題が発生すると、データサービスを同じノード上で再起動するか、
# クラスタ内の別のノードにフェイルオーバーする。モニター用の PMF
# タグは $RESOURCE_NAME.monitor。
#pragma ident ``@(#)dns_monitor_start 1.1 00/05/24 SMI''
################################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ``$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
      logger -p ${SYSLOG_FACILITY}.err \
                 -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    ``ERROR: Option $OPTARG unknown''
                     exit 1
                     ;;
                esac
        done

}



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

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

(続く)

```
# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY

# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

PMF_TAG=$RESOURCE_NAME.monitor

# データサービスの RT_BASEDIR プロパティを取得することによって、検
# 証メソッドが存在する場所を見つける。
RT_BASEDIR=scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME

# PMF の制御下でデータサービスの検証を開始する。無限再試行オプショ
# ンを使用して検証メソッドを起動する。リソースの名前、タイプ、および
# グループを検証メソッドに渡す。
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
\
    -T $RESOURCETYPE_NAME

# HA-DNS のモニターが起動されたことを示すメッセージを記録する。
if [ $? -eq 0 ]; then
 logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
         ``${ARGV0} Monitor for HA-DNS successfully started''
fi
exit 0
```

MONITOR_STOP メソッドのコードリスト

このメソッドは、データサービスの PROBE プログラムを停止します。

例 **B–7** dns_monitor_stop メソッド

```
#!/bin/ksh
#
# HA-DNS の Monitor_Stop メソッド
#
# PMF を使用して動作しているモニターを停止する。
#pragma ident ``@(#)dns_monitor_stop 1.1 00/05/24 SMI''
```

(続く)

```
#############################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ''$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    ''ERROR: Option $OPTARG unknown''
                     exit 1
                     ;;

                esac
        done

}


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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY

# このメソッドに渡された引数を構文解析する。
parse_args ''$@''

PMF_TAG=$RESOURCE_NAME.monitor
```

(続く)

```
# モニターが動作しているかどうかを調べて、動作していれば停止する。
if pmfadm -q $PMF_TAG.monitor; then
  pmfadm -s $PMF_TAG.monitor KILL
 if [ $? -ne 0 ]; then
  logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      ''${ARGV0} Could not stop monitor for resource '' \
      $RESOURCE_NAME
         exit 1
 else
  # Could successfully stop the monitor. Log a message.
  logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
      ''${ARGV0} Monitor for resource '' $RESOURCE_NAME
\
      '' successfully stopped''
 fi
fi

exit 0
```

MONITOR_CHECK メソッドのコードリスト

このメソッドは、Confdir プロパティが指すディレクトリが存在するかどうかを確認します。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するとき、RGM は MONITOR_CHECK を呼び出します。

例 **B–8** dns_monitor_check メソッド

```
#!/bin/ksh
#
# 障害モニターがデータサービスを新しいノードにフェイルオーバーするとき、RGM はこのメソッドを
# 呼び出す。Monitor_check は VALIDATE メソッドを呼び出して、新しいノード上で構成ディレク
# トリおよびファイルが利用できるかどうかを確認する。
#pragma ident ''%Z%%M% %I% %E% SMI''
#############################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
 typeset opt
```

(続く)

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

  R)
  # DNS リソースの名前
  RESOURCE_NAME=$OPTARG
  ;;

  G)
  # リソースが構成されているリソースグループの名前
  RESOURCEGROUP_NAME=$OPTARG
  ;;

  T)
  # リソースタイプの名前
  RESOURCETYPE_NAME=$OPTARG
  ;;

  *)
  logger -p ${SYSLOG_FACILITY}.err \
  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
  ''ERROR: Option $OPTARG unknown''
  exit 1
  ;;

  esac
 done

}

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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY

# このメソッドに渡された引数を構文解析する。
parse_args ''$@''

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

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

# 当該リソースの VALIDATE メソッド名を取得する。
```

(続く)

```
VALIDATE_METHOD=scha_resource_get -O VALIDATE \
 -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME

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

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

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

# モニター検査が成功したことを示すメッセージを記録する。
if [ $? -eq 0 ]; then
 logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
  ''${ARGV0} Monitor check for DNS successful.''
 exit 0
else
 logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
  ''${ARGV0} Monitor check for DNS not successful.''
 exit 1
fi
```

VALIDATE メソッドのコードリスト

このメソッドは、Confdir プロパティが示すディレクトリの存在を確認します。RGM がこのメソッドを呼び出すのは、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するとき、MONITOR_CHECK メソッドはこのメソッドを呼び出します。

例 **B–9** dns_validate メソッド

```
#!/bin/ksh
#
# HA-DNS の Validate メソッド
#
# このメソッドは、リソースの Confdir プロパティを妥当性検査する。
# Validate メソッドが呼び出されるのは、リソースが作成されたときと、リソース
# プロパティが更新されたときの 2 つである。リソースが作成されたとき、
# Validate メソッドは -c フラグで呼び出され、すべてのシステム定義プ
# ロパティと拡張プロパティがコマンド行引数として渡される。リソースプロ
# パティが更新されたとき、Validate メソッドは -u フラグで呼び出され、
# 更新されるプロパティのプロパティ/値のペアだけがコマンド行引数とし
# て渡される。
#
# 例: リソースが作成されたとき、コマンド行引数は次のようになる。
#
# dns_validate -c -R <..> -G <...> -T <..>
-r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# 例: リソースプロパティが更新されたとき、コマンド行引数は次のようになる。
#
# dns_validate -u -R <..> -G <...> -T <..>
-r <sys-prop_being_updated=value>
#   または
# dns_validate -u -R <..> -G <...> -T <..>
-x <extn-prop_being_updated=value>
#
#pragma ident ''@(#)dns_validate 1.1 00/05/24 SMI''
################################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
 typeset opt

 while getopts 'cur:x:g:R:T:G:' opt
 do
                case ''$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                r)
   # メソッドはシステム定義プロパティにアクセスして
   # いない。したがって、このフラグは動作なし。
```

(続く)

```
                        ;;

                g)
    # メソッドはリソースグループプロパティにアクセスして
    # いない。したがって、このフラグは動作なし。
                        ;;

                c)
    # Validate メソッドがリソースの作成中に呼び出されてい
    # ることを示す。したがって、このフラグは動作なし。
                        ;;

                u)
    # リソースがすでに存在しているときは、プロパティの更新
    # を示す。Confdir プロパティを更新する場合、Confdir
    # がコマンド行引数に現れるはずである。現れない場合、
    # メソッドは scha_resource_get を使用して Confdir
    # を探す必要がある。
    UPDATE_PROPERTY=1
                        ;;

                x)
    # 拡張プロパティのリスト。プロパティと値のペア。区
    # 切り文字は「=」
    PROPERTY=echo $OPTARG | awk -F= '{print $1}'
    VAL=echo $OPTARG | awk -F= '{print $2}'

    # Confdir 拡張プロパティがコマンド行上に存在する場
    # 合、その値を記録する。
    if [ $PROPERTY == ''Confdir'' ]; then
     CONFDIR=$VAL
     CONFDIR_FOUND=1
    fi
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$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
```

(続く)

```
# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY

# CONFDIR の値を NULL に設定する。この後、このメソッドは Confdir プロパ
# ティの値を、コマンド行から取得するか、scha_resource_get を使
# 用して取得する。
CONFDIR=''''
UPDATE_PROPERTY=0
CONFDIR_FOUND=0

# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

# プロパティの更新の結果として呼び出されている場合、Validate メソッ
# ドはコマンド行から Confdir 拡張プロパティの値を取得する。そうでな
# い場合、scha_resource_get を使用して Confdir の値を取得する。

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

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

# 実際の Confdir プロパティ値の妥当性検査はここから始まる。

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

# named.conf ファイルが Confdir ディレクトリ内に存在するかどうかを
# 検査する。
if [ ! -s $CONFDIR/named.conf ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
            ``${ARGV0} File $CONFDIR/named.conf is missing
or empty''
        exit 1
fi

# Validate メソッドが成功したことを示すメッセージを記録する。
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
```

(続く)

```
   ''${ARGV0} Validate method for resource ''$RESOURCE_NAME
\
   '' completed successfully''

exit 0
```

UPDATE メソッドのコードリスト

RGM は、UPDATE メソッドを呼び出して、プロパティが変更されたことを実行中のリソースに通知します。

例 B–10 dns_update メソッド

```
#!/bin/ksh
#
# HA-DNS の Update メソッド
#
# 実際のプロパティの更新は RGM が行う。更新の影響を受けるのは障害モ
# ニターだけである。したがって、このメソッドは障害モニターを再起動
# する必要がある。
#pragma ident ''@(#)dns_update 1.1 00/05/24 SMI''
###############################################################################
# プログラム引数を構文解析する。
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case ''$opt'' in
                R)
                        # DNS リソースの名前
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # リソースが構成されているリソース
                        # グループの名前
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
```

(続く)

```
                        # リソースタイプの名前
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    ``ERROR: Option $OPTARG unknown''
                     exit 1
                     ;;

                esac
        done

}



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

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

# メッセージの記録に使用する syslog 機能番号を取得する。
SYSLOG_FACILITY=scha_cluster_get -O SYSLOG_FACILITY

# このメソッドに渡された引数を構文解析する。
parse_args ``$@''

PMF_TAG=$RESOURCE_NAME.monitor

# リソースの RT_BASEDIR プロパティを取得することによって、検証メソッド
# が存在する場所を見つける。
RT_BASEDIR=scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME

# Update メソッドが呼び出されると、RGM は更新されるプロパティの値を
# 更新する。このメソッドは、障害モニター (検証メソッド) が動作し
# ているかどうかを検査し、動作している場合は強制終了し、再起動
# する必要がある。
if pmfadm -q $PMF_TAG.monitor; then

 # すでに動作している障害モニターを強制終了する。
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                    ``${ARGV0} Could not stop the monitor''
                exit 1
        else
                # DNS の停止に成功。メッセージを記録する。
```

(続く)

```
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    ''Monitor for HA-DNS successfully stopped''
        fi

 # モニターを再起動する。
 pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
    -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $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

RGM の有効な名前と値

この付録では、RGM の名前と値に対する有効な文字の要件を示します。

RGM の有効な名前

RGM の名前は次の 5 つのカテゴリに分類されます。

- リソースグループ名
- リソースタイプ名
- リソース名
- プロパティ名
- 列挙型リテラル名

リソースタイプ名を除いて、すべての名前は次の規則に従う必要があります。

- ASCII であること。
- 英字で始まること。
- 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 (_)。
- 255 文字を超えないこと。

リソースタイプ名は、簡単な名前 (RTR ファイルの `Resource_type` プロパティで指定) または完全な名前 (RTR ファイルの `Vendor_id` と `Resource_type` で指定) のどちらでもかまいません。これら両方のプロパティを指定するとき、RGM は、`Vendor_id` と `Resource_type` の間にピリオドを挿入して、完全な名前を作

成します。たとえば、Vendor_id=SUNW で、Resource_type=sample の場合、完全な名前は SUNW.sample です。これは、RGM の名前において、ピリオドが有効な文字である場合だけです。

RGM の値

RGM の値は、プロパティ値と記述値という 2 つのカテゴリに分類されます。両方とも、次のような同じ規則が適用されます。

- 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。
- NULL、改行文字、コンマ、セミコロンは、値に使用できない。