Oracle® Solaris 11.3 でのシステムサービスの開発

Part No: E64117
2016 年 11 月

Part No: E64117

Copyright © 2015, 2016, Oracle and/or its affiliates. All rights reserved.

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクルまでご連絡ください。

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

このソフトウェアまたはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアまたはハードウェアは、危険が伴うアプリケーション(人的傷害を発生させる可能性があるアプリケーションを含む)への用途を目的として開発されていません。このソフトウェアまたはハードウェアを危険が伴うアプリケーションで使用する際、安全に使用するために、適切な安全装置、バックアップ、冗長性(redundancy)、その他の対策を講じることは使用者の責任となります。このソフトウェアまたはハードウェアを危険が伴うアプリケーションで使用したことに起因して損害が発生しても、Oracle Corporationおよびその関連会社は一切の責任を負いかねます。

OracleおよびJavaはオラクル およびその関連会社の登録商標です。その他の社名、商品名等は各社の商標または登録商標である場合があります。

Intel、Intel Xeonは、Intel Corporationの商標または登録商標です。すべてのSPARCの商標はライセンスをもとに使用し、SPARC International, Inc.の商標または登録商標です。AMD、Opteron、AMDロゴ、AMD Opteronロゴは、Advanced Micro Devices, Inc.の商標または登録商標です。UNIXは、The Open Groupの登録商標です。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。適用されるお客様とOracle Corporationとの間の契約に別段の定めがある場合を除いて、Oracle Corporationおよびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。適用されるお客様とOracle Corporationとの間の契約に定めがある場合を除いて、Oracle Corporationおよびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

ドキュメントのアクセシビリティについて

オラクルのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWeb サイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracle Supportへのアクセス

サポートをご契約のお客様には、My Oracle Supportを通して電子支援サービスを提供しています。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info) か、聴覚に障害のあるお客様は (http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

目次

例目次

このドキュメントの使用法

- **概要** – Oracle Solaris のサービス管理機能 (SMF) を使用する方法について説明します。SMF は、より広範な Oracle Solaris の予測的セルフヒーリング機能のコンポーネントの 1 つです。
- **対象読者** – システム管理者、およびアプリケーションを構成するためのカスタムサービスを作成するアプリケーション開発者。
- **前提知識** – Oracle Solaris システムの管理に関する経験

製品ドキュメントライブラリ

この製品および関連製品のドキュメントとリソースは http://www.oracle.com/pls/topic/lookup?ctx=E62101-01 で入手可能です。

フィードバック

このドキュメントに関するフィードバックを http://www.oracle.com/goto/docfeedback からお聞かせください。

♦ ♦ ♦ 第 1 章

1 サービス管理機能サービスの開発の概要

Oracle Solaris サービス管理機能 (SMF) フレームワークは、システムサービスおよびアプリケーションサービスを管理します。SMF は、システムの操作に不可欠な重要なシステムサービスを管理し、データベースや Web サーバーなどのアプリケーションサービスを管理します。

SMF は、構成ファイルを使用した方法に置き換わるサービスの管理機能であり、アプリケーションの起動に使用する推奨のメカニズムです。SMF は、`init` スクリプト始動メカニズム、`inetd.conf` 構成、および `rc?.d` のほとんどのスクリプトに置き換わります。

この章では、次の内容について説明します。

- SMF の詳細情報
- このリリースの新機能
- 一部の SMF コマンドを使用するために必要な特権を取得する方法

SMF のドキュメント

このガイドでは、アプリケーションのサービスサポートを提供する SMF サービスを開発する方法について説明し、次のトピックが含まれています。

- サービス作成ツールの使用。
- SMF サービスへの `inetd.conf` 構成の変換。
- 構成ファイルへの SMF サービスプロパティーの変換。このメカニズムは、SMF で管理されるが、従来のように構成ファイルを必要とするアプリケーションとやりとりするサービスを橋渡しします。
- `cron` ジョブのように、継続的にではなく定期的に実行されるサービスの作成。

次の情報については、『Oracle Solaris 11.3 でのシステムサービスの管理』を参照してください。

- サービスプロパティー値の検査や変更、サービスインスタンスの有効化、インストールされているサービスのトラブルシューティングなどのシステム管理者向けの情報。

- サービスの状態、サービスモデル、サービスリスタータ、サービスプロパティー、サービスバンドル、サービス構成プロパティーなどの概念やコンポーネントの説明。
- さまざまなタイプの依存関係とその属性と効果の説明。
- 既存のサービスの新しいインスタンスを作成したり、既存のサービスインスタンスを変更したりする方法。

次のリソースは、アプリケーション構成などのタスクを実行するためのサービスを作成して提供する追加の例を提供します。

- 『Oracle Solaris 11.3 でのImage Packaging System を使用したソフトウェアのパッケージ化と配布』 の 第 7 章,「パッケージインストールの一環としてのシステム変更の自動化」
- 『Oracle Solaris 11.3 でのImage Packaging System を使用したソフトウェアのパッケージ化と配布』 の 第 8 章,「パッケージ更新の高度なトピック」。

このリリースの新機能

今回のリリースの SMF の新機能は次のとおりです。

定期的なサービス
: 永続的な実行とは大きく異なり、定期的なサービスはスケジュールされた間隔で実行されます。

メソッドソース
: サービスメソッドは、Python やその他の言語で記述できます。

サービス管理の特権

サービスマニフェストおよびプロファイルをエクスポートおよび開発するために、特別な特権は必要ありません。`svccfg` および `svcadm` コマンドを使用してサービスの状態および構成を変更するには、追加の特権が必要です。必要な特権を得るには、次のいずれかの方法を使用します。必要なプロファイルまたは役割を判断する方法や特権を割り当てる方法など、承認、プロファイル、および役割の詳細は、『Oracle Solaris 11.3 でのユーザーとプロセスのセキュリティー保護』を参照してください。

役割
: `roles` コマンドを使用して、自分に割り当てられている役割を一覧表示します。役割の名前を付けて `su` コマンドを使用して、その役割になります。この役割とし

て、その役割に関連付けられた権利プロファイルで許可されるコマンドを実行できます。たとえば、役割にサービス構成権利プロファイルが割り当てられている場合は、`svccfg` および `svcadm` コマンドを実行して、サービスのプロパティーおよびサービスの状態を変更できます。

権利プロファイル

`profiles` コマンドを使用して、自分に割り当てられている権利プロファイルを一覧表示します。権利プロファイルで実行が許可されるコマンドを実行するには、次のメソッドのいずれかを使用します。

- `pfbash` や `pfksh` などのプロファイルシェルを使用します。
- 実行するコマンドの前に、`pfexec` コマンドを使用してください。一般に、特権付きコマンドを実行するたびに、`pfexec` コマンドを指定する必要あります。

sudo コマンド

サイトのセキュリティーポリシーに応じて、自分のユーザーパスワードで `sudo` コマンドを使用し、特権コマンドを実行できる場合があります。

開発中のサービスを使用する管理者に固有の特権が必要である場合は、23 ページの「サービスタスクのセキュリティー保護」を参照してください。

♦ ♦ ♦ 第 2 章

2 SMF を使用したアプリケーションの制御

この章には、すべての SMF サービスに適用される次の情報が記載されています。

- 命名規則とデータタイプ定義
- 標準の場所へのサービスファイルの格納などのベストプラクティス
- サービス開発のトラブルシューティング情報

この章では、`svcbundle` ツールを使用して新しいサービスの作成を開始する方法、および実行制御スクリプトを SMF サービスに変換する方法についても説明します。

後続の章では、特定の目的での特定タイプのサービスの作成について説明します。

SMF サービスの作成

SMF サービスは、1 個以上のサービスマニフェストと 0 個以上のプロファイルファイルで構成されます。サービスインスタンスは、インスタンスの作業を実行するメソッドを定義します。

サービスマニフェストには、インスタンス、依存関係、アプリケーション構成プロパティー、サービスの開始および停止時に実行するメソッドなど、特定のサービスに関連付けられたプロパティーの完全なセットが含まれています。マニフェストには、サービスの説明などテンプレート情報も記載されています。

プロファイルは、マニフェストですでに定義されているサービスのインスタンスを定義できます。プロファイルは、これらのサービスインスタンスの新しいプロパティーと、サービスマニフェストで定義されているプロパティーの新しい値を定義できます。プロファイルは、テンプレート要素を定義できません。

SMF マニフェストとプロファイルの内容と形式の詳細は、`service_bundle(4)` のマニュアルページおよび `/usr/share/lib/xml/dtd/service_bundle.dtd.1` サービスバンドル DTD を参照してください。また、命名規則とプロパティーグループタイプの割り当てについては、20 ページの「ネームサービス、インスタンス、プロパティーグループ、およびプロパティー」を、さまざまなプロパティータイプの値については、21 ページの「プロパティーグループタイプとプロパティータイプ」を参照してください。

メソッドは、デーモン、ほかのバイナリ実行可能ファイル、または実行可能スクリプトの場合があります。詳細は、22 ページの「サービスインスタンスメソッドの作成」を参照してください。

複数のマニフェストを使用して、単一のサービスを記述できます。このメソッドはたとえば、サービスの既存のマニフェストを変更することなく、サービスの新しいインスタンスを定義する場合に役立ちます。詳細は、28 ページの「複数のマニフェストを使用したサービスの作成」を参照してください。

カスタムサービスを作成する際には、次のベストプラクティスを使用してください。

- 『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービス名」で説明されているように、サービス名には接頭辞 `site` を使用します。`site` の接頭辞はサイト固有のカスタマイズ用に予約されています。`svc:/site/`*service-name* という名前のサービスは、Oracle Solaris リリースで提供されるサービスと競合しません。
- ユーザーがこのサービスに関する情報を `svcs` コマンドと `svccfg describe` コマンドから取得できるように、名前と説明メタデータをマニフェストに追加します。プロパティー値の説明も追加できます。DTD 内の `value`、`values`、および `template` の要素を参照してください。
- `svccfg validate` コマンドを使用して、サービスマニフェストファイルまたはサービスインスタンス FMRI を検証します。
- `smf_method_exit()` 関数を使用して、メソッドスクリプトの終了の成功または失敗をサービスインスタンスのログファイルで文書化します。
- 次の表に示されている標準の場所にマニフェスト、プロファイル、およびメソッドファイルを格納します。

表 1 サービス開発ファイルの標準の場所

ファイル	標準の場所
マニフェスト	`/lib/svc/manifest/site`
プロファイル	`/etc/svc/profile/site`
メソッド	`/lib/svc/method`

これらの場所に格納されたマニフェストとプロファイルは、サービスが起動する前に、ブートプロセス中に `svc:/system/early-manifest-import:default` サービスによってサービス構成リポジトリにインポートされます。インポートプロセスを早期に実行することにより、サービスが起動するよりも先に、最新のマニフェストに含まれる情報がリポジトリに確実に取り込まれます。これらの標準の場所に格納されたマニフェストとプロファイルは、`svc:/system/manifest-import` サービスが再起動するときにもインポートされます。

標準の場所にあるマニフェスト、プロファイル、およびメソッドファイルを使用して、`manifest-import` サービスを再起動し、サービスインスタンスをインストールおよび構成します。`svcs` コマンドを使用して、サービスインスタンスのステータスを確認します。

サービスバンドル生成ツールを使用した SMF サービスの作成

svcbundle サービスバンドル生成ツールを使用すると、単純なサービスを作成したり、より複雑なサービスを起動したりできます。詳細は、svcbundle(1M)のマニュアルページを参照してください。サービスバンドル DTD やほかのサービスマニフェストを使用して、より複雑なサービスを完了させることができます。

▼ svcbundle を使用して SMF サービスを作成する方法

注記 - 定期的なサービスを作成する場合は、この手順を使用しないでください。37 ページの「svcbundle を使用して、定期的なサービスを作成する方法」を参照してください。

---

1. **サービスモデルを判断します。**

   デフォルトで、svcbundle は transient サービスを作成します。このサービスの起動メソッドスクリプトが、長時間実行するデーモンを起動し、そのためにこのサービスが contract サービスであるかどうかを判断します。サービスモデルについては、『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービスモデル」、svcbundle(1M) マニュアルページの model プロパティー、および svc.startd(1M) マニュアルページの startd/duration プロパティーを参照してください。

2. **標準の場所にスクリプトをコピーします。**

   この例のサービスでは、ex_svc という名前のカスタムスクリプトを起動メソッドとして使用します。このスクリプトを /lib/svc/method/ex_svc にコピーします。

3. **初期マニフェストを作成します。**

   この例では、サービス名は site/ex_svc です。このサービスは、一時サービスであり、停止メソッドを必要としません。

   ```
   $ svcbundle -o /tmp/ex_svc.xml -s service-name=site/ex_svc \
   -s start-method=/lib/svc/method/ex_svc
   ```

   このサービスが契約サービスの場合、-s model=contract のように contract または daemon を model または duration プロパティーの値として指定します。

4. **マニフェストに対し必要な変更をすべて行います。**

   /tmp/ex_svc.xml マニフェストの内容が必要なものかを検証します。たとえば、依存関係を追加したり、メソッドのタイムアウトを調整したりする必要がある場合があります。サービスの内容とサービスのプロパティーの使用方法を説明するコメントを追加します。

5. **マニフェストが有効であることを検証します。**

   `svccfg validate` コマンドを使用して、サービスマニフェストが有効であることを確認します。

   ```
   $ svccfg validate /tmp/ex_svc.xml
   ```

6. **マニフェストを標準ディレクトリにコピーします。**

   ```
   $ cp /tmp/ex_svc.xml /lib/svc/manifest/site/ex_svc.xml
   ```

7. **マニフェストをインポートし、サービスを起動します。**

   ```
   $ svcadm restart manifest-import
   ```

8. **新しいサービスを一覧表示します。**

   新しいサービスが存在し、予想される状態になっていることを検証します。

   ```
   $ svcs ex-svc
   ```

例 1 生成されたマニフェストを自動的にインストールする

新しいサービスマニフェストに変更を加える必要がない場合、`-i` オプションを使用して、作成後すぐにプロファイルをインストールできます。`svcbundle` コマンドは、`/lib/svc/manifest/site` にマニフェストを書き込み、`manifest-import` サービスを再起動します。`/lib/svc/manifest/site` ディレクトリにおける同じ名前の既存のファイルは上書きされます。

```
$ svcbundle -i -s service-name=site/ex_svc \
-s start-method=/lib/svc/method/ex_svc
```

ネームサービス、インスタンス、プロパティーグループ、およびプロパティー

サービス名、インスタンス名、プロパティーグループ名、およびプロパティー名は、次の式に適合する必要があります。

```
([A-Za-z][_A-Za-z0-9.-]*,)?[A-Za-z][_A-Za-z0-9-]*
```

サービス名、インスタンス名、プロパティーグループ名、またはプロパティー名は大文字と小文字が区別され、英字で始まる必要があり、英数字、アンダースコア (_)、およびハイフン (-) を含むことができます。オプションで、サービス、インスタンス、プロパティーグループ、またはプロパティー名の先頭にプロバイダ名を含めることができます。プロバイダ名はコンマ (,) で区切られ、英字で始まる必要があり、1 つまたは複数のピリオド (.) を含むことができます。

FMRI では、プロパティーグループとプロパティー名は、コンマ文字がエンコードされない点を除き、Uniform Resource Identifier (URI) Generic Syntax RFC 3986 のインターネット標準に従ってエンコードされます。

次の例は、プロパティーの完全な FMRI を示しています。これは、サービスインスタンスの FMRI、そのあとに `/:properties/`、そのあとにプロパティーの名前が続きます。`svcprop` コマンドの `-f` オプションを使用して、プロパティーの完全な FMRI を表示できます。

```
svc:/application/pkg/server:default/:properties/pkg/port
```

サービス FMRI については、『Oracle Solaris 11.3 でのシステムサービスの管理』 の「サービス名」を参照してください。

プロパティーグループタイプとプロパティータイプ

プロパティーグループタイプはこのプロパティーグループのカテゴリです。プロパティーグループタイプには次が含まれます。

```
application
configfile
dependency
framework
implementation
method
template
```

新しいプロパティーグループタイプを導入できます。プロパティーグループタイプの名前は、140 文字を超えない自由形式の文字列です。

プロパティーグループを作成するときには、プロパティーグループのタイプは `application` にするか、作成した新しいタイプにしてください。プロパティーグループタイプ `configfile`、`dependency`、`framework`、`implementation`、`method`、および `template` には、SMF での特殊用途があります。タイプ `application` のプロパティーグループは、このプロパティーグループがアタッチされているサービスにのみ関係があるとみなされます。

次の表で、さまざまなタイプのプロパティーに指定できる値について説明します。この情報は、`scf_value_create(3SCF)` のマニュアルページからも入手できます。

表 2 サービスプロパティータイプの値の説明

プロパティータイプ	値の説明
`boolean`	シングルビット: `true` または `false`
`count`	符号なし 64 ビット量
`integer`	符号付き 64 ビット量
`time`	次の範囲の符号付き 64 ビットの秒または符号付き 32 ビットのナノ秒 (*ns*): `0 <= ns < 1,000,000,000`

プロパティータイプ	値の説明
astring	8 ビットの NULL 終端文字列
ustring	8 ビットの UTF-8 文字列
uri	URI 文字列
fmri	障害管理リソース識別子
host	ホスト名、IPv4 アドレス、または IPv6 アドレス
hostname	完全修飾ドメイン名
net_addr	有効な net_addr_v4 または net_addr_v6 アドレス
net_addr_v4	オプションのネットワーク部分が付いたドット付き 10 進表記の IPv4 アドレス
net_addr_v6	オプションのネットワーク部分が付いた有効な IPv6 アドレス

サービスインスタンスメソッドの作成

起動メソッドは、サービスインスタンスの作業を実行します。その他のメソッドは、たとえばサービスインスタンスを無効にしたりリフレッシュしたりするために必要なタスクを実行します。

サービスマニフェストまたはプロファイルで、メソッドは、name および exec 属性が含まれる exec_method 要素で定義されます。name 属性に指定できる値は、リスタータのマニュアルページに記載されています。たとえば、svc.startd(1M) のマニュアルページで説明されているように、マスターリスタータ /lib/svc/bin/svc.startd では、start、stop、および refresh の各メソッドがサポートされます。

restart メソッドはありません。svcadm restart および svccfg restart コマンドは、stop メソッド、start メソッドの順に実行します。

exec 属性は、メソッドが実行する動作を定義します。exec 属性に指定できる値には、次の例に示されているように、カスタムメソッドスクリプト、既存の実行可能ファイル、または SMF で定義されている特殊なトークンが含まれます。

```
exec='/lib/svc/method/tcsd.sh start'
```

慣例により、この例に示されているカスタムのスクリプトは、メソッドの name 属性の値を指定する引数を取ります。この方法で、そのサービスインスタンスのすべてのメソッドに同じスクリプトを使用できます。

```
exec='/usr/lib/zones/zonestatd'
```

この例では、既存の実行可能ファイルを指定します。

```
exec=':true'
exec=':kill'
```

トークン :kill および :true については、smf_method(5) のマニュアルページで説明されています。:true トークンは、リスタータで必要でも特定のサービスインスタンスの実装には必要ではないメソッドに使用してください。

:kill トークンを使用すると、プライマリサービス契約内のすべてのプロセスが強制終了されるため、stop メソッドに最適です。一般に、契約内のプロセスがシグナルを正常に処理するようにプログラミングされていなければ、refresh メソッドで :kill が実行されないはずです。

サービスメソッドスクリプト

スクリプトであるメソッドは、たとえば Bourne シェル互換のスクリプトまたは Python スクリプトにできます。

- ファイル /lib/svc/share/smf_include.sh は、Bourne シェル互換のメソッドスクリプト用のヘルパー関数を多数定義します。
- ファイル /usr/lib/*python-version*/vendor-packages/smf_include.py は、Python に一意の次の関数を含む、Python メソッドスクリプト用のヘルパー関数を多数定義します。
  - smf_subprocess() – サブプロセス内の指定した実行可能ファイルを開始します。プロセスは即時に返され、インスタンスは契約サービスとして機能できます。
  - smf_main() – フレーム検査を使用して、メソッドスクリプトから適切な関数を呼び出します。/usr/lib/*python-version*/vendor-packages/smf_include.py ファイルのコメントを参照してください。
- ファイル /lib/svc/share/smf_exit_codes.sh は、メソッドの終了コードを定義します。

smf_method_exit() 関数を使用して、メソッドスクリプトの終了をサービスインスタンスのログファイルで文書化します。smf_method_exit() 関数は、終了コード、終了理由を要約したトークン、および終了の詳細を説明していることがある文字列を取ります。smf_method_exit() 関数の構文については、smf_method_exit(3SCF) のマニュアルページを参照してください。終了コードのリストについては、/lib/svc/share/smf_exit_codes.sh を参照してください。

サービスタスクのセキュリティー保護

サービスを実行できるユーザーを制限したり、サービスを実行するためにユーザーが必要な特権を指定するには、次のオプションのいずれかを使用します。

- プロパティー値を読み取り、サービスプロパティーおよびプロパティー値を変更し、サービス上でアクションを実行するために必要な承認を指定するには、*_authorization プロパティーを使用します。詳細は、smf_security(5) のマニュアルページを参照してください。
- 次のプロパティーの値として要件を指定するには、method_credential 要素を使用します。

- `user`。数値またはテキスト形式のユーザー ID。空または `:default` の場合は、uid 0 およびデフォルトのホームディレクトリ `/` が使用されます。
- `group`。数値またはテキスト形式のグループ ID。空または`:default` の場合は、`passwd` データベースでユーザーに関連付けられたグループが使用されます。
- `supp_groups`。メソッドに関連付けられる補助グループ ID。コンマまたは空白で区切られます。空白または `:default` の場合は、`initgroups`(3C) が使用されます。
- `privileges`。コンマで区切られた特権のリスト。`privileges(5)` のマニュアルページを参照してください。
- `limit_privileges`。コンマで区切られた特権のリスト。`privileges(5)` のマニュアルページを参照してください。

第5章「Oracle データベースインスタンスを管理するサービスの作成」では、インスタンス制御サービスおよびリスナーサービスによって、グループ `dba` 内のユーザー `oracle` が実行する必要があるサービスが指定されます。

`network/ntp` サービスには、`privileges` プロパティーの値として必要な承認のリストが表示されます。

`network/physical` サービスには、`supp_groups` プロパティーの使用が表示されます。

- 使用するメソッドの権利プロファイルを指定します。MySQL および Apache の例については、『Oracle Solaris 11.3 でのユーザーとプロセスのセキュリティー保護』の 「拡張特権を使用したリソースのロックダウン」を参照してください。

サービス開発のベストプラクティス

サービスを開発する際は、このセクションで説明するガイドラインに従ってください。

サービスメソッドのベストプラクティス

サービス起動メソッドやその他のサービスメソッドを開発する際は、このセクションで説明するガイドラインに従ってください。

SMF メソッドの終了および役に立つ終了理由の使用

サービスでは、初期化が完了し、サービスを提供する準備ができたら正常ステータスが返されることが予測されます。

起動メソッドを終了するには、`smf_method_exit()` を使用します。`exit()` は使用しないでください。`smf_method_exit()` インタフェースには次の引数が必要です。

- `/lib/svc/share/smf_exit_codes.sh` 内または `smf_method(5)` のマニュアルページ内で定義されているメソッドの終了コードの 1 つ
- 終了の理由の簡単な説明
- 終了の理由のより長い説明

エラーメッセージが有益 (問題を解決するためのガイドラインを含む) であることを確認してください。メソッドで呼び出されたコマンドから、メッセージやその他の情報を取得する必要がある場合もあります。メソッドが既存の実行可能ファイルである場合は、終了メッセージを改善するために、メソッドスクリプト内でその実行可能ファイルを呼び出す必要がある場合もあります。これらのメッセージは、システム管理者がサービスログファイルで確認します。

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

依存関係の使用、タイムアウト使用の回避

サービスの初期化が完了するまで、起動メソッドを終了しないでください。サービスの初期化が完了する前に起動メソッドを終了した場合は、このサービスに依存するサービスを起動できません。

メソッドタスクが完了するのに長い時間がかかるという理由だけでエラーが発生するのを回避するため、`timeout_seconds` プロパティーに適した値を設定してください。

依存関係が `online` 状態に達するのに必要な時間を許可するために `timeout_seconds` 値またはその他の種類のタイムアウトや待機は使用しないでください。代わりに、依存関係を適切に宣言してください。依存関係が起動するまでに十分な時間を許可することに加えて、このサービスに `require` 依存関係が含まれるサービスでエラーが発生すると、このサービスでもエラーが発生し、適切なメッセージが表示され、エラーが発生した依存関係が起動するまでの待機が続行されないようにする必要があります。この場合も、適切に宣言された `dependency` 要素が正しい実装です。『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービス依存関係の表示」および `smf(5)` のマニュアルページの依存関係に関するセクションを参照してください。

ドキュメントの指定

`smf_template(5)` のマニュアルページの説明に従って、適切なテンプレート情報を指定します。管理者は `svccfg describe` コマンドを使用して、この情報を表示できます。

- `smf_template`(5) のマニュアルページのサービスおよびインスタンスの共通名に関するセクションの説明に従って、サービスを表す短い共通名を指定します。

- 詳細は、適切なマニュアルページ (manpage 要素) または信頼できる URL (doc_link 要素) を参照してください。
- このサービスに固有のプロパティーグループおよびプロパティーの名前、説明、選択、および制限を指定します。

サービスマニフェストの検証

サービスマニフェストを検証するには、`svccfg validate` コマンドを使用します。

`svccfg validate` コマンドがエラー「Required property group missing」で失敗した場合、部分的なマニフェストを検証しようとしている可能性があります。単一のサービスが、複数のマニフェストにまたがって指定されている可能性があります。このエラーを回避するには、28 ページの「複数のマニフェストを使用したサービスの作成」で説明されているように、複数マニフェストサービスのすべてのマニフェストで `service_bundle` 要素に `include` プロパティーを指定してください。

標準の場所の使用

サービスマニフェストおよびプロファイルを、標準の所有権とアクセス権を持つ標準の場所にコピーします。マニフェストおよびプロファイルのファイルには、標準以外の場所を使用しないでください。マニフェストおよびプロファイルの共通の場所については、『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービスバンドル」または表1を参照してください。

自身で使用するサービスを作成する場合は、サービス名の先頭に `site` を使用し (`svc:/site/`*service-name*`:`*instance-name*)、マニフェストを `/lib/svc/manifest/site` に配置します。

サービスをテストするには、マニフェストおよび関連付けられたプロファイルを正しい標準の場所に配置し、`manifest-import` サービスを起動します。関連情報については、『Oracle Solaris 11.3 でのシステムサービスの管理』 の 第 5 章,「複数のシステムの構成」および『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「機能低下、オフライン、または保守であるインスタンスの修復」を参照してください。

実行制御スクリプトの SMF サービスへの変換

このセクションでは、実行制御サービスを SMF で管理できるように、実行制御スクリプトを SMF サービスマニフェストに置き換える方法について説明します。

▼ 実行制御スクリプトを SMF サービスに変換する方法

この手順では、`svcbundle` コマンドで `rc-script` プロパティーを使用して、実行制御スクリプトを SMF サービスに変換する方法について説明します。

1. **サービスモデルを判断します。**

   デフォルトで、`svcbundle` は `transient` サービスを作成します。この実行制御スクリプトが、長時間実行するデーモンを起動し、そのためにこのサービスが `contract` サービスであるかどうかを判断します。サービスモデルについては、『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービスモデル」、`svcbundle(1M)` マニュアルページの `model` プロパティー、および `svc.startd(1M)` マニュアルページの `startd/duration` プロパティーを参照してください。

2. **初期マニフェストを作成します。**

   実行制御スクリプトを変換するには、`rc-script` プロパティー名を、`svcbundle` コマンドの `-s` オプションとともに使用します。詳細は、`svcbundle(1M)` のマニュアルページの `rc-script` プロパティーを参照するか、`svcbundle help rc-script` と入力してください。

   この例では、サービス名は `ex_con` であり、レベル 2 で実行する契約サービスです。実行レベルは、`rc-script` プロパティー値のスクリプト名のあとにあるコロンのあとに指定されています。

   ```
   $ svcbundle -o /tmp/ex_con.xml -s service-name=ex_con
         -s rc-script=/etc/init.d/ex_con:2 -s model=contract
   ```

3. **マニフェストに対し必要な変更をすべて行います。**

   `/tmp/ex_con.xml` マニフェストの内容が必要としている内容かを検証します。たとえば、依存関係を追加したり、メソッドのタイムアウトを調整したりする必要がある場合があります。サービスの内容とサービスのプロパティーの使用方法を説明するコメントを追加します。

4. **マニフェストが有効であることを検証します。**

   `svccfg validate` コマンドを使用して、サービスマニフェストが有効であることを確認します。

   ```
   $ svccfg validate /tmp/ex_con.xml
   ```

5. **マニフェストを標準ディレクトリにコピーします。**

   ```
   $ cp /tmp/ex_con.xml /lib/svc/manifest/site/ex_con.xml
   ```

6. **既存のサービスを停止します。**

   ```
   $ /etc/init.d/ex_con stop
   ```

7. **実行制御スクリプトを無効にします。**

該当する rc*n*.d ディレクトリから実行制御スクリプトへのリンクを削除します。

8. **マニフェストをインポートし、サービスを起動します。**

```
$ svcadm restart manifest-import
```

9. **新しいサービスを一覧表示します。**
   新しいサービスが存在し、予想される状態になっていることを検証します。

```
$ svcs ex-con
```

複数のマニフェストを使用したサービスの作成

新しいパッケージに、サービスのカスタムで構成されたインスタンスが必要な場合、そのパッケージは、ほかのサービスを変更せずにカスタムインスタンスのみを提供できます。

複数のマニフェストを使用してサービスを定義すると、親サービスを変更せずに、必要に応じてサービスインスタンスのみを提供できます。たとえば、サービス S のインスタンス I のみがツール T で必要な場合は、ツール T は、サービス S を再度提供することなくインスタンス I を提供できます。ツール T がシステムにインストールされていない場合、サービス S がインストールされていても、インスタンス I もインストールされません。同様に、ツール T がアンインストールされると、インスタンス I はアンインストールされ、サービス S はまだインストールされたままになり、変更されません。サービス S はあるマニフェストで指定して、インスタンス I は別のマニフェストで指定してください。サービス S のマニフェストをあるパッケージで提供して、インスタンス I のマニフェストはツール T のパッケージで提供します。ツール T のパッケージには、サービス S を提供するパッケージへの依存関係があります。

複数のマニフェストを使用して単一のサービスを指定する場合、次のサービスマニフェストの設計を使用します。

- 1 つのマニフェストに、サービス定義、テンプレートデータ、およびデフォルトのインスタンスを含めます。
- サービスの追加のインスタンスを定義する各マニフェストに、次を含めます。
  - サービス定義が含まれているマニフェストの場合と同じ `service_bundle` 要素を指定してから、`include` 属性を追加します。`include` 属性の値は、サービス定義が含まれているマニフェストの完全なファイル名です。
  - サービス定義が含まれているマニフェストの場合と同じ `service` 要素を指定します。`service` 要素内の `name` 属性の値は、サービス定義が含まれているマニフェストの場合と完全に同じです。
- 同じサービスまたはインスタンスを複数のマニフェストで提供しないでください。このタイプの競合が検出された場合、SMF は使用する定義を判別できず、インスタンスは保守状態になります。

複数のマニフェストで提供されるサービスの例は、svc:/system/console-login サービスです。console-login サービスには、次のインスタンスとマニフェストが含まれています。

svc:/system/console-login:default

マニフェスト /lib/svc/manifest/system/console-login.xml は、サービス定義、テンプレート、および default インスタンスを提供します。

svc:/system/console-login:terma

マニフェスト /lib/svc/manifest/system/console-login-terma.xml は、console-login サービスの terma インスタンスを提供します。

svc:/system/console-login:termb

マニフェスト /lib/svc/manifest/system/console-login-termb.xml は、console-login サービスの termb インスタンスを提供します。

svc:/system/console-login:vt?

マニフェスト /lib/svc/manifest/system/console-login-vts.xml は、console-login サービスの vts インスタンスを提供します。

サービス定義を含んでいるマニフェスト /lib/svc/manifest/system/console-login.xml には、次の行が含まれています。

```
<service_bundle type="manifest" name="SUNWcs:console">

<service
        name="system/console-login"
        type="service"
        version="1">

<instance name='default' enabled='true'>
</instance>
```

追加のインスタンス vt2、vt3、vt4、vt5、および vt6 を定義するマニフェスト /lib/svc/manifest/system/console-login-vts.xml には、次の行が含まれています。

```
<service_bundle type="manifest" name="SUNWcs:console"
    include="/lib/svc/manifest/system/console-login.xml">

<service
        name="system/console-login"
        type="service"
        version="1">

<instance name='vt2' enabled='true'>

        <dependency
                name='system-console'
                grouping='require_all'
                restart_on='none'
                type='service'>
                <service_fmri value='svc:/system/console-login:default' />
```

```
        </dependency>

</instance>
```

インスタンス vt3、vt4、vt5、および vt6 の定義には、vt2 インスタンスについて示すように console-login:default への同じ依存関係が含まれています。

次の例に示すように、svcs コマンドの出力には依存関係が表示されます。

```
$ svcs 'svc:/system/console-login:vt*'
STATE          STIME    FMRI
online         Dec_04   svc:/system/console-login:vt3
online         Dec_04   svc:/system/console-login:vt5
online         Dec_04   svc:/system/console-login:vt2
online         Dec_04   svc:/system/console-login:vt4
online         Dec_04   svc:/system/console-login:vt6
$ svcs -D console-login:default
STATE          STIME    FMRI
online         Dec_04   svc:/system/vtdaemon:default
online         Dec_04   svc:/system/console-login:vt3
online         Dec_04   svc:/system/console-login:vt5
online         Dec_04   svc:/system/console-login:vt2
online         Dec_04   svc:/system/console-login:vt4
online         Dec_04   svc:/system/console-login:vt6
online         Dec_04   svc:/system/console-reset:default
$ svcs -d 'svc:/system/console-login:vt*'
STATE          STIME    FMRI
...
online         Dec_04   svc:/system/console-login:default
online         Dec_04   svc:/system/vtdaemon:default
```

default、terma、および termb インスタンスは system/core-os パッケージによって提供されます。vts インスタンスは system/virtual-console パッケージによって提供されます。console-login:default インスタンスが存在することを確認するために、system/virtual-console パッケージには、system/core-os パッケージへの require 依存関係が含まれています。

サービスに含まれているインスタンスが 1 つのみの場合、ベストプラクティスは、単一のマニフェストを使用してサービスを指定することです。サービス定義が含まれているマニフェストで 1 つのインスタンスを定義します。

♦ ♦ ♦ 第 3 章

3 定期的に実行するサービスの作成

この章では、定期的に比較的短いタスクを実行するサービスを作成する方法について説明します。この章では、次の内容について説明します。

- 定期的なサービスの定義
- 定期的なサービスを作成する方法
- 定期的なサービスの実行スケジュールを指定する方法

定期的なサービス

ほとんどのシステムサービスは、長時間実行のデーモンとして実装され、管理者が介入するまで実行されます。定期的なサービスまたはスケジュールされているサービスは、比較的短いタスクを定期的に実行します。スケジュールされているサービスは、定期的なサービスのタイプの 1 つです。スケジュールされているサービスの詳細は、第4章「特定のスケジュールで実行するサービスの作成」を参照してください。

`cron` で実行するように以前に構成したタスクを実行するために、定期的なサービスを作成できます。SMF サービスを使用してサービスを IPS パッケージで提供する利点の 1 つは、SMF と IPS の依存関係機能を活用して、ほかの必須ソフトウェアがインストールされて実行されているときにのみタスクが実行されるようにできることです。別の利点は、ユーザーがパッケージをアンインストールするときに、定期的なタスクは削除され、個別に `crontab` ファイルから削除する必要はない点です。

定期的なサービスインスタンスは、システムの起動時に `svc:/system/svc/periodic-restarter` サービスによって呼び出される、定期的なリスタータ `svc.periodicd` によって管理されます。インスタンスが `online` 状態になっているときは常に、定期的なリスタータは、スケジュールされた間隔で管理するインスタンスの起動メソッドを実行します。有効にされている定期的なインスタンスは、そのインスタンスのすべての依存関係が満たされるとすぐに `online` 状態に遷移します。エラーまたは管理者の操作が発生しない場合、定期的なサービスは、起動メソッドの実行と実行の間の、関連付けられたプロセスが実行されていないときも `online` 状態を維持します。詳細は、`svc.periodicd`(1M) のマニュアルページを参照してください。

定期的なサービスの作成

定期的なサービスマニフェストは、次の例に示すように非常に単純です。37 ページの「サービスバンドル生成ツールを使用した、定期的なサービスの作成」も参照してください。

例 2 定期的なサービスマニフェスト

定期的なサービスインスタンスは、`periodic_method` 要素で完全に定義されます。この例のサービスでは、定期的なリスタータは、15 秒の最初の遅延後に起動メソッドを 30 - 35 秒ごとに実行します。管理者がこの定期的なサービスの目的を理解するには、`template` 要素が推奨されます。

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='site/sample-periodic-svc'>
    <service type='service' version='1' name='site/sample-periodic-svc'>

        <instance name='default' enabled='false'>

            <periodic_method
                period='30'
                delay='15'
                jitter='5'
                exec='/usr/bin/periodic_service_method'
                timeout_seconds='0'>
                    <method_context>
                        <method_credential user='root' group='root' />
                    </method_context>
            </periodic_method>

        </instance>

        <template>
            <common_name>
                <loctext xml:lang="C">
                    Sample Periodic Service
                </loctext>
            </common_name>
            <description>
                <loctext xml:lang="C">
                    What this service does periodically.
                </loctext>
            </description>
        </template>
    </service>
</service_bundle>
```

periodic_method 要素の指定

periodic_method 要素が存在する場合、インスタンスは定期的なリスタータに自動的に委任されます。periodic_method 要素は、service 要素内または instance 要素内で指定できます。periodic_method 要素は、定期的なサービスのメソッド情報とスケジューリング情報の両方を指定し、このセクションで説明されている属性と要素を持つことができます。period 属性と exec 属性は必須です。

定期的なサービスのスケジューリング制約属性

表2で示したように、time 値の単位は秒数です。

delay 属性
: オプション。値タイプ: time。デフォルト値: 0。サービスが online 状態に遷移してから、起動メソッドの最初の呼び出しまでの固定の秒数。

period 属性
: 必須。値タイプ: time。起動メソッドの呼び出し間の秒数。

jitter 属性
: オプション。値タイプ: time。デフォルト値: 0。period から起動メソッドが実行されるまでのランダムな最大秒数。使用される最後の秒数は、0 からこのプロパティーの値までの範囲です。

その他の定期的なサービスのスケジューリング属性

persistent 属性
: オプション。値タイプ: boolean。デフォルト値: false。スケジューリングをシステムの停止時間を越えて維持するかどうかを指定します。

  値が false であれば、定期的なサービスインスタンスが有効にされたあとで online 状態に遷移した場合と同様に、起動メソッドのスケジューリングが再開されます。

  persistent 属性の値が true で、recover 属性 (下記) の値が false の場合、停止時間から回復したインスタンスの起動メソッドのスケジューリングは、停止時間が発生する前に定義されたものと同じスケジュールで続行されます。

  persistent 属性の値が true で、recover 属性の値が true の場合は、次の recover 属性の説明を参照してください。

recover 属性

オプション。値タイプ: boolean。デフォルト値: false。システムの停止時間中に呼び出しが失われた場合にインスタンスでリカバリを実行するかどうかを指定します。

このインスタンスの persistent 属性の値が true である場合にのみ、この値は有効です。

persistent 属性の値が true で、recover 属性の値が true の場合、インスタンスがシステムの停止時間から回復したら可能なかぎりすぐに、定期的なリスタータはインスタンスの起動メソッドを呼び出します。後続の呼び出しは、period 値と jitter 値に従って発生します。

persistent プロパティーの値が true で、recover プロパティーの値が false の場合は、前述の persistent プロパティーの説明を参照してください。

定期的なサービスの起動メソッドの属性とコンテキスト

exec 属性

必須。値タイプ: astring。exec システム呼び出しに渡すために適している必要がある、行うべきアクション。smf_method(5) のマニュアルページを参照してください。

この起動メソッドはタスクを実行したあとで、period 属性で指定された時間内に終了します。

定期的なリスタータは一時的なサービスを実装しないため、SMF_EXIT_TEMP_TRANSIENT 終了コードは、定期的なサービスの起動メソッドには適用されません。定期的なサービスの起動メソッドで使用する場合、SMF_EXIT_TEMP_TRANSIENT 終了コードは SMF_EXIT_ERR_OTHER 終了コードと同じように扱われます。

定期的なサービスインスタンスでは、起動メソッドのみが使用されます。リフレッシュまたは停止メソッドが定義されている場合、警告メッセージがマニフェストのインポート時に発行され、リフレッシュまたは停止メソッドは無視されます。定期的なインスタンスがリフレッシュされると、35 ページの「サービス構成リポジトリへの定期的なサービスデータの格納」で説明されているように、定期的なリスタータは、periodic プロパティーグループ内のプロパティーの値を再読み取りします。定期的なインスタンスによって契約されているプロセスは永続的に実行されないため、定期的なリスタータは停止メソッドを必要としません。定期的なインスタンスは短期のプロセスを実行したあとで、次にスケジュールされた実行時間まで待機します。

timeout_seconds 属性

オプション。値タイプ: integer。メソッドのアクションが完了するまで待機する秒数。無限のタイムアウトを指定するには、値 0 または -1 を指定します。

exec_method 要素では timeout_seconds 属性値が必要です。この periodic_method 要素で timeout_seconds 属性の値を指定しない場合、exec_method 要素の値は無限であると想定されます。指定された timeout_seconds 値より長く起動メソッドが実行される場合、または定期的なリスタータが次の期間に起動メソッドを呼び出そうとしたときに契約済みプロセスがまだ存在する場合の起動メソッドのスケジューリングに関する説明については、40 ページの「起動メソッドの問題後のスケジュール」を参照してください。

method_context 要素

オプション。smf_method(5) のマニュアルページのメソッドコンテキストに関するセクションを参照してください。

サービス構成リポジトリへの定期的なサービスデータの格納

33 ページの「periodic_method 要素の指定」で説明されているように、periodic_method 要素は、定期的なサービスのメソッド情報 (起動メソッドの exec_method プロパティーグループ) およびスケジューリング情報 (periodic プロパティーグループ) の両方を提供します。periodic_method 要素のあるマニフェストをインポートすると、このセクションで説明されているデータは、サービス構成リポジトリに格納されます。

リスタータのプロパティ

サービスのリスタータは svc:/system/svc/periodic-restarter:default に設定されます。管理者は、svcs -l コマンドを使用してリスタータを表示することも、svccfg または svcprop コマンドを使用して general/restarter プロパティーを表示することもできます。

定期的なサービスインスタンスのオンライン状態は起動メソッドの呼び出しと呼び出しの間も維持されるため、インスタンスは、関連付けられた契約済みプロセスがシステムで実行中でなくても online 状態になることができます。オンラインの定期的なサービスインスタンスの場合、メソッドアクションが実行中であるかどうかを区別するために、auxiliary_state プロパティーには次のいずれかの値を指定できます。

running　　インスタンスはオンラインで、契約済みのプロセスが関連付けられています。

scheduled　　インスタンスはオンラインですが、契約済みのプロセスは関連付けられていません。

この状態を確認するには、管理者は `svcs -o astate` コマンドを使用して ASTATE 列を表示することも、`svccfg` または `svcprop` コマンドを使用して `general/auxiliary_state` プロパティーを表示することもできます。

`periodic` プロパティーグループ

`periodic_method` 要素のスケジューリング属性 (`period`、`delay`、`jitter`、`persistent`、および `recover`) は、`periodic` という名前のプロパティーグループのプロパティーとして格納されます。これらの属性の定義については33 ページの「`periodic_method` 要素の指定」、その使用方法の例については38 ページの「定期的なサービスの起動メソッド実行のスケジュール」を参照してください。管理者は、`svcprop` コマンドと `svccfg` コマンドを使用して、これらの `periodic` プロパティーグループのプロパティーを表示および変更できます。

起動メソッドの最後および次の呼び出し

サービス構成リポジトリには、インスタンスの次の 2 つのスケジューリング情報も格納されます。

`last_run`
: このインスタンスの起動メソッドの実行が最後に試行された絶対時間。起動メソッドが実行されたことがない場合、このプロパティーは存在しません。この時間を確認するには、管理者は `svcs -o lrun` コマンドを使用して、LRUN 列を表示できます。

`next_run`
: このインスタンスの起動メソッドの実行が次に試行される絶対時間。絶対時間が存在しない場合、このプロパティーはないか、このプロパティーに値が指定されていない可能性があります。38 ページの「定期的なサービスの起動メソッド実行のスケジュール」で説明されているように、特定の `RAND(`*jitter*`)` 値を含め、次にスケジュールされている起動メソッドの呼び出しの `next_run` の値は `last_run` の設定時に計算され、そのような呼び出しとなります。この時間を確認するには、管理者は `svcs -o nrun` コマンドを使用して、NRUN 列を表示できます。この時間は、それぞれの定期的なサービスインスタンスの `periodic-restarter` サービスによって管理されます。管理者はこの値を変更できません。

start プロパティーグループ

exec 値と timeout_seconds 値および method_context 情報は、start という名前のプロパティーグループのプロパティーとして格納されます。この start プロパティーグループは、定期的なサービスの起動メソッドを表し、ほかのサービスの起動メソッドと同じ方法で定義されます。

サービスバンドル生成ツールを使用した、定期的なサービスの作成

svcbundle コマンドを使用して、定期的なサービスを作成する場合、service-name と start-method の両方のプロパティーとともに period プロパティーを指定する必要があります。デフォルトで、svcbundle は transient サービスを作成します。-s period を指定すると、svcbundle によって定期的なサービスが作成されます。

▼ svcbundle を使用して、定期的なサービスを作成する方法

1. **標準の場所に起動メソッドをコピーします。**

   この例では、このサービスの起動メソッドの名前は per_ex です。この実行可能ファイルを /lib/svc/method/per_ex にコピーします。

2. **初期マニフェストを作成します。**

   この例では、サービス名は site/per_ex です。起動メソッドのスケジューリングの期間を指定します。bundle-type、duration、model、rc-script、refresh-method、または stop-method のどのプロパティーも指定しないでください。

   ```
   $ svcbundle -o /tmp/per_ex.xml -s service-name=site/per_ex \
         -s start-method=/lib/svc/method/per_ex -s period=30
   ```

   period プロパティーを指定すると、svcbundle によって periodic_method 要素が作成され、これにより、マニフェストのインポート時にサービスのリスタータは定期的なリスタータに設定されます。period プロパティーの値は、periodic_method 要素の period 属性の値になります。start-method プロパティーの値は、periodic_method 要素の exec 属性の値になります。

3. **マニフェストに対し必要な変更をすべて行います。**

   /tmp/ex_svc.xml マニフェストの内容が必要なものかを検証します。次のような変更を行うことができます。

   - periodic_method 要素に delay、jitter、および timeout_seconds の値を追加します。

- periodic_method 要素に method_context 要素を追加します。
- デフォルトのインスタンスの enabled 属性の値を true から false に変更します。

サービスの内容とサービスのプロパティーの使用方法を説明するコメントを追加します。

4. **マニフェストが有効であることを検証します。**

   svccfg validate コマンドを使用して、サービスマニフェストが有効であることを確認します。

5. **マニフェストを標準ディレクトリにコピーします。**

```
$ cp /tmp/per_ex.xml /lib/svc/manifest/site/per_ex.xml
```

6. **マニフェストをインポートし、サービスを起動します。**

```
$ svcadm restart manifest-import
```

7. **新しいサービスを一覧表示します。**

   新しいサービスが存在し、予想される状態になっていることを検証します。

```
$ svcs per_ex
```

定期的なサービスの起動メソッド実行のスケジュール

(periodic_method 要素の exec 属性によって指定された) 定期的なサービスインスタンスの起動メソッドの実行をスケジュールする場合は、常に periodic_method 要素の period 属性の値が必要です。periodic_method 要素のその他の属性も使用されることがあり、next_run 値が使用される場合があります。

periodic_method 要素のスケジューリング属性
(period、delay、jitter、persistent、および recover) は、periodic プロパティーグループのプロパティーとして格納されます。たとえば、定期的なサービスをインポートすると、マニフェスト内の period 属性は periodic/period プロパティーになります。管理者は、svcprop コマンドと svccfg コマンドを使用して periodic プロパティーグループのプロパティー値を表示して、svccfg コマンドを使用してこれらの値を変更できます。

period プロパティーには有効な値が必要です。jitter、delay、persistent、および recover の各プロパティーにはデフォルト値があり、明示的に設定する必要はありません。表2で示したように、time 値の単位は秒数です。

次のスケジューリングの説明では、イタリックはプロパティーの値を示します。たとえば、*jitter* は、サービス構成リポジトリ内の periodic/jitter プロパティーの値

で、これはサービスマニフェスト内の `periodic_method` 要素の `jitter` 属性の値と同じです。

インスタンスが最初に有効にされたあとでスケジュール

定期的なサービスインスタンスがはじめて有効にされると (たとえば、システムのリブート時)、インスタンスの起動メソッドの最初の実行は、インスタンスが `online` 状態に遷移してから次の秒数で行われます。

delay + RAND(*jitter*)

インスタンスの起動メソッドの後続の実行は、次の相対的な時間に行われます。

period + RAND(*jitter*)

スケジュールの変動を防ぐために、後続の実行では、前の実行の *jitter* 値は無視されます。たとえば、インスタンスの起動メソッドの 2 番目の実行は、インスタンスが `online` 状態に遷移してから次の秒数で行われます。

delay + *period* + RAND(*jitter*)

インスタンスの起動メソッドの *n* 番目の実行は、インスタンスが `online` 状態に遷移してから次の秒数で行われます。

delay + (*n*-1)*period* + RAND(*jitter*)

システムの停止時間後のスケジュール

`persistent` プロパティーと `recover` プロパティーは、定期的なインスタンスがシステムの停止時間から回復したときに、定期的なインスタンスメソッドが実行される時間を変更できます。`recover` プロパティーの値は、`persistent` プロパティーの値が `true` である場合にのみ有効です。

`persistent` プロパティーの値が `false` の場合、システムの停止時間から回復した定期的なインスタンスの起動メソッドの次の実行は、インスタンスが `online` 状態に遷移してから次の秒数で行われます。

delay + RAND(*jitter*)

`persistent` プロパティーの値が `true` で、`recover` プロパティーの値が `false` の場合、システムの停止時間から回復したインスタンスの起動メソッドの次の実行は、次の絶対時間にスケジュールされます。

next_run + (*n*)*period*

next_run プロパティーの値が将来の値である場合、*n* は 0 です。next_run プロパティーの値が過去の値である場合、将来の時間になる最小値である *n* が使用されます。後続の呼び出しは、period 値と jitter 値に従ってその時間から発生します。next_run プロパティーの説明は、36 ページの「起動メソッドの最後および次の呼び出し」に記載されています。

persistent プロパティーの値が true で、recover プロパティーの値が true の場合、インスタンスがシステムの停止時間から回復したら可能なかぎりすぐに、定期的なリスタータはインスタンスの起動メソッドを呼び出します。後続の呼び出しは、period 値と jitter 値に従ってその時間から発生します。

サービスの再起動後のスケジュール

定期的なリスタータサービス (svc:/system/svc/periodic-restarter) は、終了した場合に自動的に再起動しようとします。障害後に定期的なリスタータサービスが再起動すると、それぞれの定期的なインスタンスの起動メソッドが次の絶対時間にスケジュールされます。

next_run + (*n*)*period*

next_run プロパティーの値が将来の値である場合、*n* は 0 です。next_run プロパティーの値が過去の値である場合、将来の時間になる最小値である *n* が使用されます。後続の呼び出しは、period 値と jitter 値に従ってその時間から発生します。next_run プロパティーの説明は、36 ページの「起動メソッドの最後および次の呼び出し」に記載されています。

定期的なサービスインスタンスが再起動すると、インスタンスの起動メソッドは、インスタンスが online 状態に遷移してから次の秒数で呼び出されます。

delay + RAND(*jitter*)

起動メソッドの問題後のスケジュール

起動メソッドはタスクを実行したあと、period プロパティーで指定された時間内に終了します。定期的なリスタータが次の期間に起動メソッドを呼び出そうとしたときに契約済みのプロセスがまだ存在する場合、その期間の呼び出しはスキップされ、定期的なリスタータは次の期間に再度起動メソッドを呼び出そうとします。定期的なリスタータは、インスタンスが online 状態に遷移してから次の秒数で再度メソッドを呼び出します。ここで、*n* は、将来の時間になる最小値です。

delay + (*n*)*period* + RAND(*jitter*)

timeout_seconds 値で指定された秒数より長く起動メソッドが実行される場合、契約内のすべてのプロセスが終了し、呼び出しは致命的でない障害になります。はじめて起動メソッドが致命的でない障害で終了した場合、インスタンスは degraded 状

態になり、起動メソッドの呼び出しはスケジュールどおりに続行されます。次の 2 つの呼び出しのいずれかが成功した場合、インスタンスは `online` 状態に戻されます。`degraded` 状態から `online` 状態に遷移したあとで、定期的なリスタータは、インスタンスが最初に `offline` から `online` 状態に遷移したら次の秒数でメソッドを呼び出します。ここで、*n* は、将来の時間になる最小値です。

delay + (*n*)*period* + RAND(*jitter*)

起動メソッドの致命的でない障害が 3 回連続して発生したあとで、インスタンスは `degraded` 状態から `maintenance` 状態に遷移します。起動メソッドの最初の致命的な障害時に、サービスは `maintenance` 状態になります。『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「機能低下、オフライン、または保守であるインスタンスの修復」を参照してください。

♦ ♦ ♦ 第 4 章

特定のスケジュールで実行するサービスの作成

この章では、指定された一定の時間に比較的短いタスクを実行するサービスを作成する方法について説明します。この章では、次の内容について説明します。

- スケジュールされているサービスの定義
- スケジュールされているサービスを作成する方法
- スケジュールされているサービスの実行スケジュールを指定する方法

スケジュールされているサービス

スケジュールされているサービスとは、第3章「定期的に実行するサービスの作成」で説明されている定期的なサービスのタイプの 1 つです。定期的なサービスインスタンスの起動メソッドの各呼び出しは、最後の呼び出しに相対的な時間で発生します。スケジュールされているサービスインスタンスの起動メソッドの各呼び出しは、指定された絶対時間に発生します。オフピークの時間中などの特定の時間にタスクを実行する必要がある場合は、スケジュールされているサービスを使用します。

スケジュールされているサービスインスタンスは、定期的なリスタータサービスである `svc:/system/svc/periodic-restarter` によって管理されます。スケジュールされているサービスインスタンスの起動メソッドのスケジューリングパラメータは、サービスマニフェスト内の `scheduled_method` 要素とゼロ個以上の `schedule` プロパティーグループで定義されます。起動メソッドのスケジューリングパラメータは、サービス構成リポジトリ内のタイプ `schedule` のプロパティーグループに格納されます。定期的なリスタータは、タイプ `schedule` の各プロパティーグループの各プロパティーの値を組み合わせて、スケジュールされているサービスインスタンスの起動メソッドの呼び出しをスケジュールします。

スケジュールされているサービスの作成

次のマニフェストの例は、スケジュールされている非常に単純なサービス用です。49 ページの「サービスバンドル生成ツールを使用した、スケジュールされているサービスの作成」も参照してください。

例 3 スケジュールされているサービスマニフェスト

この例のスケジュールされているサービスインスタンスは、`scheduled_method` 要素で完全に定義されます。スケジュールされているサービスでは、タスクをスケジュールするために必要なもっとも少ないスケジューリング制約を使用するべきです。たとえば、定期的なリスタータは、毎月 1 日の 02:00 から 03:00 の間に次のサービス例の起動メソッドを実行します。02:00 から 02:01 の間にメソッドを呼び出すには、制約 `minute='0'` を追加します。

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='site/sample-periodic-svc'>
    <service type='service' version='1' name='site/sample-periodic-svc'>

        <instance name='default' enabled='false'>

            <scheduled_method
                interval='month'
                day='1'
                hour='2'
                exec='/usr/bin/scheduled_service_method'
                timeout_seconds='0'>
                    <method_context>
                        <method_credential user='root' group='root' />
                    </method_context>
            </scheduled_method>

        </instance>
    </service>
</service_bundle>
```

`scheduled_method` 要素の指定

`scheduled_method` 要素が存在する場合、サービスは定期的なリスタータに自動的に委任されます。`scheduled_method` 要素は、`service` 要素内または `instance` 要素内で指定できます。`scheduled_method` 要素は、スケジュールされているサービスのメソッド情報とスケジューリング情報の両方を指定します。`periodic_method` 要素に関しては、メソッド情報は `scheduled_method` 要素についても同じです。`scheduled_method` 要素には、このセクションで一覧表示されている属性を指定できます。`interval` 属性と `exec` 属性は必須です。

スケジュールされているサービスのスケジューリング制約の値を指定する場合、次の点に留意してください。

- タスクをスケジュールするために必要な最小数のスケジューリング制約を指定します。`interval` 値より短い期間を表す制約は、インスタンスの起動メソッドを呼び出す期間中のタイミングをより正確に指定します。
- もっとも長い時間を表す制約から、もっとも短い時間を表す制約までの、スケジューリング制約の連続したセットを指定します。たとえば、`week_of_year` 値のみは指定できても、`day_of_month` または `day` を指定しないかぎり、`week_of_year` と `hour` は指定できません。
- `frequency` の値が `1` より大きい場合、`interval` 値と等しいかこの値よりも長い期間を表す制約を指定できます。これらの制約の使用方法の詳細は、50 ページの「スケジュールされているサービスの起動メソッド実行のスケジュール」を参照してください。

スケジュールされているサービスのスケジューリング制約属性

`interval`

必須。値タイプ: `astring`。`frequency` の値に応じた、サービスの起動メソッドの呼び出し間における基本の時間の長さ。`interval` の値は、`year`、`month`、`week`、`day`、`day_of_month`、`hour`、`minute` のいずれかです。

`frequency`

オプション。値タイプ: `count`。デフォルト値: 1。定期的なリスタータによって起動メソッドが実行される前に発生する必要がある間隔の数。たとえば、`interval` が `week` で、`frequency` が 4 の場合、起動メソッドは 4 週間ごとに呼び出されます。月によっては 5 週間あるため、「4 週間ごと」は毎月 4 週目とは異なります。

`frequency` の値が `1` より大きい場合、`year` から、`interval` 値と同じ長さの時間までのすべての制約を指定する必要があります。たとえば、`interval` が `week` の場合、`year` と `week_of_year` の値を指定する必要があります。例については、51 ページの「複数の間隔ごとに 1 回の呼び出しのスケジュール」を参照してください。

`timezone`

オプション。値タイプ: `astring`。デフォルト値: システムのタイムゾーン。スケジュールの作成および解釈に使用するタイムゾーン。この属性を使用して、システムに対して構成されているタイムゾーン以外の、タイムゾーンに相対的なスケジュールを定義します。

`year`

値タイプ: `count`。グレゴリオ暦年。起動メソッドを呼び出す年。

`week_of_year`

値タイプ: `integer`。ISO 8601 の暦週日付における年の中の週の序数。有効な値は、`1 - 53` および `-1 - -53` です。負の数は、年の最終週より前の週数を指定します。52 週ある年では、`-1` は `52` と同じで、53 週ある年では、`-1` は `53` と同じです。`53` または `-53` を指定する場合、52 週のみある年では起動メソッドは実行されません。

`week_of_year` 属性と `month` 属性は同時に使用できません。

`month`

値タイプ: `astring`。グレゴリオ暦年の月。有効な値は、`1 - 12`、`-1 - -12`、月の C ロケールの完全な名前、または月の名前の C ロケールの 3 文字の省略形です。負の数は、年の最終月より前の月数を指定します。C ロケールの名前では大文字と小文字は区別されません。

`week_of_year` 属性と `month` 属性は同時に使用できません。

`day_of_month`

値タイプ: `integer`。グレゴリオ暦月の日。有効な値は、`1 - 31` および `-1 - -31` です。負の数は、月の最終日より前の日数を指定します。たとえば、4 月に `31` や `-1` のように、特定の月に存在しない日を指定した場合、起動メソッドはその月の最終日に実行されます。

`day_of_month` 属性と `day` 属性は同時に使用できません。

`weekday_of_month`

値タイプ: `integer`。指定した日 (`day` を参照) が発生する月の週の序数。有効な値は、`1 - 5` および `-1 - -5` です。負の数は、月の最終週より前の週数を指定します。たとえば、月に木曜日が 5 回ある場合、その月の第 3 木曜日 (3) は、月の最後から 2 番目の木曜日 (`-2`) とは異なります。`5` または `-1` を指定した場合、起動メソッドは、指定した `day` が 4 回のみある月では実行されません。

`weekday_of_month` を指定する場合は、`day` も指定する必要があります。

`day`

値タイプ: `astring`。ISO 8601 の標準の週の曜日。有効な値は、`1 - 7`、`-1 - -7`、日の C ロケールの完全な名前、または日の名前の C ロケールの 3 文字の省略形です。負の数は、週の終わりより前の日数を指定します。C ロケールの名前では大文字と小文字は区別されません。

`day_of_month` 属性と `day` 属性は同時に使用できません。

`hour`

値タイプ: `integer`。ISO 8601 の標準の日の時間。有効な値は、`0 - 23` および `-1 - -24` です。負の数は、日の終わりより前の時間数を指定します。スケジュールされている日に、標準時間とサマータイムの間の切り替えが行われる場合、指定した時間はその日に 2 回発生するか、その日にはまったく発生しないことがあり

ます。スケジュールされている日に指定した時間が複数回発生する場合、起動メソッドはその時間の最初の発生時にのみ実行されます。スケジュールされている日に指定した時間が発生しない場合、起動メソッドは次の時間に実行されます。

`minute`

値タイプ: `integer`。時間の分。有効な値は、`0` - `59` および `-1` - `-60` です。負の数は、時間の終わりより前の分数を指定します。

その他のスケジュールされているサービスのスケジューリング属性

`recover`

オプション。値タイプ: `boolean`。デフォルト値: `false`。システムの停止時間中に呼び出しが失われた場合にインスタンスでリカバリを実行するかどうかを指定します。

`recover` 属性の値が `true` の場合、インスタンスがシステムの停止時間から回復したら可能なかぎりすぐに、定期的なリスタータはインスタンスの起動メソッドを呼び出します。後続の呼び出しは、指定されたスケジューリング制約に従って発生します。

`recover` プロパティーの値が `false` の場合、回復呼び出しは実行されません。呼び出しは、指定されたスケジューリング制約に従って続行されます。

スケジュールされているサービスの起動メソッドの属性とコンテキスト

`exec`

必須。値タイプ: `astring`。`exec` システム呼び出しに渡すために適している必要がある、行うべきアクション。`smf_method(5)` のマニュアルページを参照してください。33 ページの「`periodic_method` 要素の指定」の追加の説明を参照してください。

`timeout_seconds`

オプション。値タイプ: `integer`。起動メソッドのアクションが完了するまで待機する秒数。無限のタイムアウトを指定するには、値 `0` または `-1` を指定します。

`exec_method` 要素では `timeout_seconds` 属性値が必要です。この `scheduled_method` 要素で `timeout_seconds` 属性の値を指定しない場合、`exec_method` 要素の値は無限であると想定されます。指定された `timeout_seconds` 値より長く起動メソッドが実行される場合、または定期的なリスタータが次にスケジュールされている期間に起動メソッドを呼び出そうとしたときに契約済みプロセスがまだ存在する場合の起動メソッドのスケジューリング

に関する説明については、55 ページの「起動メソッドの問題後のスケジュール」を参照してください。

`method_context` 要素

オプション。`smf_method(5)` のマニュアルページのメソッドコンテキストに関するセクションを参照してください。

スケジュールされているサービスデータのサービス構成リポジトリへの格納

`scheduled_method` 要素は、スケジュールされているサービスのメソッド情報 (起動メソッドの `exec_method` プロパティーグループ) とスケジューリング情報 (タイプ `schedule` のプロパティーグループ) の両方を提供します。`scheduled_method` 要素のあるマニフェストをインポートすると、このセクションで説明されているデータは、サービス構成リポジトリに格納されます。

- **リスタータのプロパティー。** `restarter` プロパティーと `auxiliary_state` プロパティーについては、35 ページの「リスタータのプロパティ」を参照してください。スケジュールされているサービスは、定期的なサービスのタイプの 1 つであることを思い出してください。
- `schedule` **プロパティーグループ。** `scheduled_method` 要素のスケジューリング属性は、タイプ `schedule` のプロパティーグループのプロパティーとして格納されます。タイプ `schedule` のプロパティーグループには、45 ページの「スケジュールされているサービスのスケジューリング制約属性」および 47 ページの「その他のスケジュールされているサービスのスケジューリング属性」で説明されているプロパティーを指定できます。これらのプロパティーの使用方法の例については、51 ページの「不規則な間隔での呼び出しのスケジュール」および 50 ページの「スケジュールされているサービスの起動メソッド実行のスケジュール」を参照してください。管理者は、`svcprop` コマンドと `svccfg` コマンドを使用して、これらのプロパティーを表示および変更できます。『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「プロパティーグループタイプでのプロパティーの表示」で説明されているように、タイプ `schedule` のプロパティーグループのすべてのプロパティーを一覧表示するには、`svcprop -g schedule` コマンドを使用します。
- **起動メソッドの最後および次の呼び出し。** `last_run` プロパティーと `next_run` プロパティーの説明については、36 ページの「起動メソッドの最後および次の呼び出し」を参照してください。50 ページの「スケジュールされているサービスの起動メソッド実行のスケジュール」で説明されているように、`next_run` の値は、次にスケジュールされている起動メソッドの呼び出しです。
- `start` **プロパティーグループ。** `exec` 値と `timeout_seconds` 値および `method_context` 情報は、`start` という名前のプロパティーグループのプロパティーとして格納されます。この `start` プロパティーグループは、スケジュール

されているサービスの起動メソッドを表し、ほかのサービスの起動メソッドと同じ方法で定義されます。

サービスバンドル生成ツールを使用した、スケジュールされているサービスの作成

svcbundle コマンドを使用して、スケジュールされているサービスを作成する場合、service-name と start-method の両方のプロパティーとともに interval プロパティーを指定する必要があります。デフォルトで、svcbundle は transient サービスを作成します。-s period を指定すると、svcbundle によって定期的なサービスが作成されます。

▼ svcbundle を使用して、スケジュールされているサービスを作成する方法

1. **標準の場所に起動メソッドをコピーします。**

   この例では、このサービスの起動メソッドの名前は sched_ex です。この実行可能ファイルを /lib/svc/method/sched_ex にコピーします。

2. **初期マニフェストを作成します。**

   この例では、サービス名は site/sched_ex です。起動メソッドのスケジューリングの間隔を指定します。bundle-type、duration、model、rc-script、refresh-method、または stop-method のどのプロパティーも指定しないでください。

   ```
   $ svcbundle -o /tmp/sched_ex.xml -s service-name=site/sched_ex \
         -s start-method=/lib/svc/method/sched_ex -s interval=week
   ```

   interval プロパティーを指定すると、svcbundle によって scheduled_method 要素が作成され、これにより、マニフェストのインポート時にサービスのリスタータは定期的なリスタータに設定されます。interval プロパティーの値は、scheduled_method 要素の interval 属性の値になります。start-method プロパティーの値は、scheduled_method 要素の exec 属性の値になります。

3. **マニフェストに対し必要な変更をすべて行います。**

   /tmp/ex_svc.xml マニフェストの内容が必要なものかを検証します。次のような変更を行うことができます。

   - scheduled_method 要素に追加の制約を指定します。
   - 51 ページの「不規則な間隔での呼び出しのスケジュール」で説明されているように、タイプ schedule の追加のプロパティーグループを指定します。
   - scheduled_method 要素に method_context 要素を追加します。

- デフォルトのインスタンスの enabled 属性の値を true から false に変更します。

サービスの内容とサービスのプロパティーの使用方法を説明するコメントを追加します。

4. **マニフェストが有効であることを検証します。**
   svccfg validate コマンドを使用して、サービスマニフェストが有効であることを確認します。

5. **マニフェストを標準ディレクトリにコピーします。**

```
$ cp /tmp/sched_ex.xml /lib/svc/manifest/site/sched_ex.xml
```

6. **マニフェストをインポートし、サービスを起動します。**

```
$ svcadm restart manifest-import
```

7. **新しいサービスを一覧表示します。**
   新しいサービスが存在し、予想される状態になっていることを検証します。

```
$ svcs sched_ex
```

スケジュールされているサービスの起動メソッド実行のスケジュール

(scheduled_method 要素の exec 属性によって指定された) スケジュールされているサービスインスタンスの起動メソッドの実行をスケジュールするには、scheduled_method 要素の interval 属性と frequency 属性の値が常に必要です。frequency 属性のデフォルト値は 1 です。interval 属性の値は明示的に設定する必要があります。タイプ schedule のプロパティーグループの scheduled_method 要素またはプロパティーに指定したその他の属性も使用され、next_run 値が使用されることがあります。

間隔ごとに 1 回の呼び出しのスケジュール

frequency の値が 1 の場合、スケジュールされているサービスインスタンスの起動メソッドは最初に、指定された最小の制約内のランダムな時間に呼び出されます。後続の呼び出しは、次の実行間隔内で次に短い制約の同一単位内で発生します。たとえば、interval が week で、ほかのスケジューリング制約が指定されていない場合、起動メソッドは最初に、その週のランダムな時間に呼び出されます。起動メソッドがた

とえば 2 日連続で呼び出されないように、後続の呼び出しは将来の週の同じ曜日に発生します。同様に、interval が week で、day と hour も指定されている場合、起動メソッドは最初に、指定された日の指定された時間のランダムな時刻に呼び出されます。後続の呼び出しは、将来の週の指定された日時の同じ分に発生します。

複数の間隔ごとに 1 回の呼び出しのスケジュール

frequency の値が 1 より大きい場合、最初の呼び出しの時間を設定するために、時間の長さが interval 値以上のスケジューリング制約が使用されます。year から、interval 値と同じ長さの時間までのすべての制約を指定する必要があります。たとえば、interval が week の場合、year と week_of_year の値を指定する必要があります。

interval が week、frequency が 2、year が 2014、week_of_year が 2 - 52 までの任意の偶数である場合、起動メソッドは偶数週ごとに呼び出されます。week_of_year が 1 - 53 までの任意の奇数である場合、起動メソッドは奇数週ごとに呼び出されます。2014 年はすでに過ぎているため、この例の week_of_year 属性の値は、起動メソッドが最初に呼び出される特定の週を示しているのではなく、この起動メソッドが呼び出されるのが次の偶数週なのか次の奇数週なのかのみを示しています。2015 年のように 53 週ある年のあとでは、偶数週に最初に呼び出された起動メソッドは奇数週に呼び出されるように、後続の呼び出しは 2 週間ごと発生します。

interval が week、frequency が 4、year が 2014、week_of_year が 1 の場合、起動メソッドは 2015 年の第 1 週、第 5 週、第 9 週などに呼び出されます。2015 年は 53 週あるため起動メソッドは第 53 週に呼び出され、2016 年には起動メソッドは第 4 週、第 8 週、第 12 週などに呼び出されます。

不規則な間隔での呼び出しのスケジュール

場合によっては、1 セットのスケジューリング制約ではスケジュールを定義するために十分ではありません。スケジュールされているサービスには、タイプ schedule のプロパティーグループを複数指定できます。定期的なリスタータは、タイプ schedule の各プロパティーグループの各プロパティーの値を組み合わせて、スケジュールされているサービスインスタンスの起動メソッドの呼び出しをスケジュールします。

スケジュールを定義するためには、スケジュールされているサービスには必要な数のスケジューリング制約を指定するべきであるのと同様に、タスクに必要なスケジュールを定義するには、スケジュールされているサービスには必要な数の schedule プロパティーグループのみを指定するべきです。

例 4 12 時間ごとの呼び出し

この例では、起動メソッドは毎日 06:00 と 18:00 に呼び出されます。`hour` 属性の値には `6` または `18` のいずれかを指定できます。

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='site/sample-periodic-svc'>
    <service type='service' version='1' name='site/sample-periodic-svc'>

        <instance name='default' enabled='false'>

            <scheduled_method
                interval='hour'
                frequency='12'
                hour='6'
                minute='0'
                exec='/usr/bin/scheduled_service_method'
                timeout_seconds='0'>
                    <method_context>
                        <method_credential user='root' group='root' />
                    </method_context>
            </scheduled_method>

        </instance>
    </service>
</service_bundle>
```

例 5 毎日 03:00 および 23:00 の呼び出し

この例では、起動メソッドを毎日 03:00 と 23:00 に呼び出すために、追加の `schedule` プロパティーグループを指定します。

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='site/sample-periodic-svc'>
    <service type='service' version='1' name='site/sample-periodic-svc'>

        <instance name='default' enabled='false'>

            <scheduled_method
                interval='day'
                hour='3'
                minute='0'
                exec='/usr/bin/scheduled_service_method'
                timeout_seconds='0'>
                    <method_context>
                        <method_credential user='root' group='root' />
                    </method_context>
            </scheduled_method>

            <property_group name='run2' type='schedule'>
                <propval name='interval' type='astring' value='day' />
                <propval name='hour' type='integer' value='23' />
                <propval name='minute' type='integer' value='0' />
            </property_group>

        </instance>
    </service>
```

```
</service_bundle>
```

例 6　火曜日と木曜日の 03:00 と 23:00 の呼び出し

毎週火曜日と木曜日の 03:00 と 23:00 に起動メソッドを呼び出すには、この例に示されているように 3 つの追加の `schedule` プロパティーグループが必要です。

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='site/sample-periodic-svc'>
    <service type='service' version='1' name='site/sample-periodic-svc'>

        <instance name='default' enabled='false'>

            <scheduled_method
                interval='week'
                day='3'
                hour='3'
                minute='0'
                exec='/usr/bin/scheduled_service_method'
                timeout_seconds='0'>
                    <method_context>
                        <method_credential user='root' group='root' />
                    </method_context>
            </scheduled_method>

            <property_group name='run2' type='schedule'>
                <propval name='interval' type='astring' value='week' />
                <propval name='day' type='astring' value='3' />
                <propval name='hour' type='integer' value='23' />
                <propval name='minute' type='integer' value='0' />
            </property_group>

            <property_group name='run3' type='schedule'>
                <propval name='interval' type='astring' value='week' />
                <propval name='day' type='astring' value='5' />
                <propval name='hour' type='integer' value='3' />
                <propval name='minute' type='integer' value='0' />
            </property_group>

            <property_group name='run4' type='schedule'>
                <propval name='interval' type=astring'' value='week' />
                <propval name='day' type='astring' value='5' />
                <propval name='hour' type='integer' value='23' />
                <propval name='minute' type='integer' value='0' />
            </property_group>

        </instance>
    </service>
</service_bundle>
```

サービスがすでにインポートされている場合、サイトプロファイルを使用するか、`svccfg` のサブコマンド `addpg` および `setprop` を使用することで、これらの追加のプロパティーグループを指定できます。

1 つの間隔において可能な複数の呼び出しの解決

単一の間隔で起動メソッドの複数の呼び出し時間が可能になるようにスケジューリング制約を指定する場合、起動メソッドは、すべての制約と一致するもっとも早い時間に呼び出されます。単一の間隔で可能になる複数の呼び出し時間の例は、サマータイムから標準時間への切り替え時に、01:00 から 02:00 までの時間が 2 回発生する場合です。

システムの停止時間後のスケジュール

`recover` プロパティーは、スケジュールされているインスタンスがシステムの停止時間から回復したときに、スケジュールされているインスタンスメソッドが実行される時間を変更できます。

`recover` プロパティーの値が `true` の場合、インスタンスがシステムの停止時間から回復したら可能なかぎりすぐに、定期的なリスタータはスケジュールされているサービスインスタンスの起動メソッドを呼び出します。後続の呼び出しは、`interval` 値と `frequency` 値および指定されているその他の制約に従って発生します。

`recover` プロパティーの値が `false` の場合、定期的なリスタータは、`next_run` プロパティーによって指定された時間に、スケジュールされているサービスインスタンスの起動メソッドを呼び出します。`next_run` プロパティーの値が過去の値の場合、将来の起動メソッドの呼び出しは、`interval` 値と `frequency` 値および指定されているその他の制約に従って発生します。`next_run` プロパティーの説明は、36 ページの「起動メソッドの最後および次の呼び出し」に記載されています。

サービスの再起動後のスケジュール

定期的なリスタータサービス (`svc:/system/svc/periodic-restarter`) は、終了した場合に自動的に再起動しようとします。定期的なリスタータサービスが障害後に再起動されると、スケジュールされている各インスタンスの起動メソッドは、`next_run` プロパティーによって指定された時間に呼び出されます。`next_run` プロパティーの値が過去の値の場合、将来の起動メソッドの呼び出しは、`interval` 値と `frequency` 値および指定されているその他の制約に従って発生します。`next_run` プロパティーの説明は、36 ページの「起動メソッドの最後および次の呼び出し」に記載されています。

スケジュールされているサービスインスタンスが再起動されると、インスタンスの起動メソッドは、`interval` 値と `frequency` 値および指定されているその他の制約に従って呼び出されます。

起動メソッドの問題後のスケジュール

起動メソッドはタスクを実行したあとで、`interval` および `frequency` プロパティーで指定された期間内に終了します。定期的なリスタータが次にスケジュールされている期間に起動メソッドを呼び出そうとしたときに契約済みのプロセスがまだ存在する場合、その期間の呼び出しはスキップされ、定期的なリスタータは次の期間に再度起動メソッドを呼び出そうとします。

`timeout_seconds` 値で指定された秒数より長く起動メソッドが実行される場合、契約内のすべてのプロセスが終了し、呼び出しは致命的でない障害になります。はじめて起動メソッドが致命的でない障害で終了すると、インスタンスは `degraded` 状態になり、起動メソッドの呼び出しは、`interval` 値と `frequency` 値および指定されているその他の制約に従ってスケジュールどおりに続行されます。次の 2 つの呼び出しのいずれかが成功した場合、インスタンスは `online` 状態に戻されます。

起動メソッドの致命的でない障害が 3 回連続して発生したあとで、インスタンスは `degraded` 状態から `maintenance` 状態に遷移します。起動メソッドの最初の致命的な障害時に、サービスは `maintenance` 状態になります。『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「機能低下、オフライン、または保守であるインスタンスの修復」を参照してください。

♦ ♦ ♦ 第 5 章

5 Oracle データベースインスタンスを管理するサービスの作成

この章では、Oracle データベースの管理に役立つ次のサービスについて説明します。

- Oracle データベースインスタンスを起動または停止するデータベースサービス
- データベースインスタンスへのクライアント接続要求の着信トラフィックを管理するプロセスであるリスナーを開始するリスナーサービス

環境の構成

この章の例では、ファイルベースストレージを使用します。ファイルベースストレージを使用する代わりに、自動ストレージ管理 (ASM) 機能を使用できます。ASM はボリュームマネージャーであり、Oracle データベースファイル用のファイルシステムです。

次の環境変数は、Oracle データベースのインストールごとに設定する必要があります。

`ORACLE_HOME` データベースがインストールされている場所。このセクションの例では、データベースインストールの場所は `/opt/oracle/product/home` です。

`ORACLE_SID` システム上の特定のデータベースを一意に識別するためのシステム ID。

この章の例では、これらの環境変数はサービスマニフェストで設定されており、メソッドスクリプトで使用されます。

Oracle データベースインスタンスを起動または停止するサービスの作成

このセクションでは、Oracle データベースインスタンス制御サービスマニフェストと、そのマニフェストで使用される起動/停止メソッドスクリプトについて説明します。

インスタンス制御サービスマニフェスト

次に、Oracle データベースインスタンス制御サービスマニフェスト `/lib/svc/manifest/site/oracle.xml` について留意すべきいくつかの特徴を示します。

- `svc:/site/application/database/oracle:default` の名前の 1 つのサービスインスタンスが定義されます。このインスタンスはデフォルトで有効になっています。マニフェストのいちばん上にある `create_default_instance` 要素を参照してください。
- このサービスでは、すべてのローカルファイルシステムがマウントされている必要があります。ファイルベースデータベースを使用している場合、データベースサービスはローカルファイルシステムに依存している必要があります。ASM を使用している場合、データベースサービスは、ASM を管理するサービスに依存している必要があります。
- このサービスは、リモートクライアント接続の確立を許可するためにリスナーサービスに依存します。61 ページの「リスナーサービスマニフェスト」にある `network` マイルストーンへの `require_all` 依存関係を参照してください。
- `method_context` 要素内の `method_environment` 要素は、`ORACLE_HOME` および `ORACLE_SID` 環境変数を定義し、この環境変数は起動または停止するデータベースインスタンスを識別します。これらの値はその後、使用するメソッドスクリプトで使用できます。

  このサービスの複数のインスタンスを作成した場合、その特定のデータベースに対して一意の `ORACLE_HOME` および `ORACLE_SID` 値を定義するには、各インスタンスに独自の `method_context` 要素が必要になる可能性があります。
- `method_context` 要素の属性は、この例に示すプロジェクトおよび作業ディレクトリに加え、リソースプールを定義できます。`method_context` 要素では `method_profile` または `method_credential` のどちらかの要素も定義できます。`method_credential` 要素では、この例で示す `user`、`group`、および `privileges` の値に加え、`supp_groups` および `limit_privileges` の値を指定できます。詳細は DTD を参照してください。
- 起動/停止メソッドスクリプトは `/lib/svc/method/oracle` です。メソッドがタイムアウトするまでの秒数は、デフォルトから増えています。
- ユーザーには、このサービスインスタンスを有効または無効にするための `solaris.smf.manage.oracle` 承認が割り当てられている必要があります。この例

では、ユーザー oracle に solaris.smf.manage.oracle 認可が割り当てられています。

```
<?xml version="1.0"?>

<!--
 Define a service to control the startup and shutdown of a database instance.
-->

<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type="manifest" name="oracle">
<service name="site/application/database/oracle" type="service" version="1">
    <create_default_instance enabled="true" />

    <!-- Using a file-backed database, so depend on the filesystem. -->

    <dependency type="service"
        name="fs-local"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/system/filesystem/local" />
    </dependency>

    <!-- Wait for the listener to be started. -->

    <dependency type="service"
        name="oracle"
        grouping="optional_all"
        restart_on="none">
        <service_fmri value="svc:/site/application/database/listener" />
    </dependency>

    <!-- Define the methods. -->

    <method_context project=":default" working_directory=":default">
        <method_credential user="oracle" group="dba" privileges=":default" />
        <method_environment>
            <envvar name="ORACLE_HOME" value="/opt/oracle/product/home" />
            <envvar name="ORACLE_SID" value="oracle" />
        </method_environment>
    </method_context>

    <exec_method type="method"
        name="start"
        exec="/lib/svc/method/oracle start"
        timeout_seconds="120"/>

    <exec_method type="method"
        name="stop"
        exec="/lib/svc/method/oracle stop"
        timeout_seconds="120" />

    <!--
         What authorization is needed to allow the framework
         general/enabled property to be changed when performing the
         action (enable, disable, etc) on the service.
    -->

    <property_group name="general" type="framework">
        <propval type="astring"
            name="action_authorization"
            value="solaris.smf.manage.oracle" />
    </property_group>
```

```
    <stability value="Evolving" />
</service>
</service_bundle>
```

名前および説明メタデータをマニフェストに追加して、ユーザーがこのサービスに関する情報を `svcs` コマンドおよび `svccfg describe` コマンドから得られるようにします。DTD 内の `template` 要素を参照してください。

`svccfg validate` コマンドを使用して、サービスマニフェストが有効であることを確認します。

インスタンス制御サービスの起動/停止メソッドスクリプト

次に、Oracle データベースインスタンス制御サービス用の起動/停止メソッドスクリプト `/lib/svc/method/oracle` を示します。このメソッドは、データベースの `dbstart` コマンドおよび `dbshut` コマンドを呼び出します。

```
#!/bin/ksh -p

. /lib/svc/share/smf_include.sh

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
export PATH=$PATH:$ORACLE_HOME/bin

function startup
{
        dbstart $ORACLE_HOME
}

function shutdown
{
        dbshut $ORACLE_HOME
}

case $1 in
    start) startup ;;
    stop)  shutdown ;;

    *) echo "Usage: $0 { start | stop }" >&2
       smf_method_exit $SMF_EXIT_ERR_FATAL
       ;;
esac

smf_method_exit $SMF_EXIT_OK
```

Oracle データベースリスナーサービスの作成

このセクションでは、Oracle データベースリスナーサービスマニフェストと、そのマニフェストで使用される起動/停止メソッドスクリプトについて説明します。

リスナーとは、データベースインスタンスへのクライアント接続要求の着信トラフィックを管理するプロセスです。データベースサービスはリスナーサービスインスタンスに依存するため、データベースサービスが起動する前に、これらのデータベースサービスの依存関係をオンラインにするようにしてください。

リスナーサービスマニフェスト

次に、Oracle データベースリスナーサービスマニフェスト `/lib/svc/manifest/site/listener.xml` について留意すべきいくつかの特徴を示します。

- `svc:/site/application/database/listener:default` の名前の 1 つのサービスインスタンスが定義されます。このインスタンスはデフォルトで有効になっています。マニフェストのいちばん上にある `create_default_instance` 要素を参照してください。
- Oracle データベースインスタンス制御サービス `svc:/site/application/database/oracle` は、このリスナーサービスに依存します。ただし、リスナーサービスの起動が失敗しても、データベースサービスは引き続き起動されます。58 ページの「インスタンス制御サービスマニフェスト」にある `optional_all` 依存関係を参照してください。
- `method_context` 要素内の `method_environment` 要素は、`ORACLE_HOME` および `ORACLE_SID` 環境変数を定義し、この環境変数は起動または停止するデータベースインスタンスを識別します。これらの値はその後、使用するメソッドスクリプトで使用できます。

  このサービスの複数のインスタンスを作成した場合、その特定のデータベースに対して一意の `ORACLE_HOME` および `ORACLE_SID` 値を定義するには、各インスタンスに独自の `method_context` 要素が必要になる可能性があります。
- `method_context` 要素の属性は、この例に示すプロジェクトおよび作業ディレクトリに加え、リソースプールを定義できます。`method_context` 要素では `method_profile` または `method_credential` のどちらかの要素も定義できます。`method_credential` 要素では、この例で示す `user`、`group`、および `privileges` の値に加え、`supp_groups` および `limit_privileges` の値を指定できます。詳細は DTD を参照してください。
- 起動/停止メソッドスクリプトは `/lib/svc/method/listener` です。メソッドがタイムアウトするまでの秒数は、デフォルトから増えています。
- ユーザーには、このサービスインスタンスを有効または無効にするための `solaris.smf.manage.oracle` 承認が割り当てられている必要があります。
- サービスは `transient` です。『Oracle Solaris 11.3 でのシステムサービスの管理』の 「サービスモデル」を参照してください。

```
<?xml version="1.0"?>

<!--
 Define a service to control the startup and shutdown of a database listener.
-->
```

```
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type="manifest" name="listener">
<service name="site/application/database/listener" type="service" version="1">
    <create_default_instance enabled="true" />

    <!--
         Wait for all local file systems to be mounted.
         Wait for all network interfaces to be initialized.
    -->

    <dependency type="service"
        name="fs-local"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/system/filesystem/local" />
    </dependency>

    <dependency type="service"
        name="network"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/milestone/network:default" />
    </dependency>

    <!-- Define the methods. -->

    <method_context project=":default" working_directory=":default">
        <method_credential user="oracle" group="dba" privileges=":default" />
        <method_environment>
            <envvar name="ORACLE_HOME" value="/opt/oracle/product/home" />
            <envvar name="ORACLE_SID" value="oracle" />
        </method_environment>
    </method_context>

    <exec_method type="method"
        name="start"
        exec="/lib/svc/method/listener start"
        timeout_seconds="150"/>

    <exec_method type="method"
        name="stop"
        exec="/lib/svc/method/listener stop"
        timeout_seconds="30" />

    <!--
         What authorization is needed to allow the framework
         general/enabled property to be changed when performing the
         action (enable, disable, etc) on the service.
    -->

    <property_group name="general" type="framework">
        <propval type="astring"
            name="action_authorization"
            value="solaris.smf.manage.oracle" />
    </property_group>

    <!-- Make the instance transient (since it backgrounds itself). -->

    <property_group name="startd" type="framework">
        <propval name="duration" type="astring" value="transient" />
    </property_group>

    <stability value="Evolving" />
```

```
</service>
</service_bundle>
```

名前および説明メタデータをマニフェストに追加して、ユーザーがこのサービスに関する情報を `svcs` コマンドおよび `svccfg describe` コマンドから得られるようにします。DTD 内の `template` 要素を参照してください。

リスナーサービスの起動/停止メソッドスクリプト

次に、Oracle データベースインスタンスリスナーサービス用の起動/停止メソッドスクリプト `/lib/svc/method/listener` を示します。このメソッドは、リスナープロセス `lsnrctl` を開始または停止します。`lsnrctl` が起動すると、データベースサービスのステータスをテストします。

```
#!/bin/ksh -p

. /lib/svc/share/smf_include.sh

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
export PATH=$PATH:$ORACLE_HOME/bin

function startup
{
        lsnrctl start

        # Wait for the listener to report ready.

        i=0
        while ! lsnrctl status | grep -i ready ; do
                ((i = i+1))
                if (( $i == 120 )) ; then
                        # It's been *at least* 2 minutes, time to give up.
                        echo "The listener failed to report ready." >&2
                        smf_method_exit $SMF_EXIT_ERR_FATAL
                fi

                sleep 1
        done

        # Ping the database once to prove it is now available.

        if ! tnsping $ORACLE_SID ; then
                smf_method_exit $SMF_EXIT_ERR_FATAL
        fi
}

function shutdown
{
        lsnrctl stop
}

case $1 in
    start) startup ;;
    stop)  shutdown ;;

    *) echo "Usage: $0 { start | stop }" >&2
       smf_method_exit $SMF_EXIT_ERR_FATAL
       ;;
```

```
esac

smf_method_exit $SMF_EXIT_OK
```

♦ ♦ ♦ 第 6 章

6 ステンシルを使用した構成ファイルの作成

アプリケーションでプロパティーの読み取りに `libscf` ライブラリインタフェースを使用できない場合は、ステンシルを使用して構成ファイルを作成できます。ステンシルサービスは、ステンシルファイルと、ステンシルサービスに定義されたプロパティー値とを使用して構成ファイルを作成します。ステンシルファイルには、現在 SMF で管理されているサービスであっても、そのサービスで必要な構成ファイルの構造的な定義が含まれます。ステンシルサービスを使用すれば、既存のアプリケーションを変更せずに SMF 構成管理を利用できます。

ステンシルは、サービスインスタンス `start` メソッドが実行される直前に、構成ファイルを生成するために使用されます。ステンシル化したプロパティー値を更新する場合は、アプリケーションが起動して構成ファイルを読み取る前に、サービスを再起動して構成ファイルに変更を組み込んでください。

Oracle Solaris では、Puppet および Kerberos 用のサービスでは、ステンシルを使用して構成ファイルを提供します。

この章では、次の内容について説明します。

- ステンシルサービスを作成する方法
- Oracle Solaris の Puppet ステンシルサービス
- Oracle Solaris の Kerberos ステンシルサービス

ステンシルサービスの作成

ステンシルファイルには、現在 SMF で管理されているサービスであっても、そのサービスで引き続き必要な構成ファイルの構造的な定義が含まれます。`svcio` ユーティリティーは、ステンシルファイル内の定義と、SMF サービスのプロパティーから構成ファイルを生成します。`svcio` ユーティリティーの詳細については `svcio(1)` のマニュアルページを、ステンシルファイルの形式については `smf_stencil(4)` のマニュアルページを参照してください。

▼ ステンシルサービスの作成方法

1. **ステンシルファイルを作成します。**
   ステンシルファイルは、構成ファイルの作成で使用する形式を `svcio` ユーティリティーに伝えます。`svcio` ユーティリティーは、ステンシルと呼ばれるテンプレートに基づいてアプレケーション固有の構成ファイルに SMF プロパティーを変換します。

2. **`configfile` プロパティーグループをサービスに追加します。**
   ステンシルサービスプロパティーグループは、構成ファイルの作成で使用するパスおよび所有権を、`svcio` ユーティリティーに伝えます。SMF は、起動またはリフレッシュメソッドを実行する前に、すべてのステンシル対応サービスに対して構成を再生成します。タイプ `configfile` のプロパティーグループは、構成ファイルの生成方法を SMF に伝えます。それぞれの `configfile` プロパティーグループは、サービスの単一の構成ファイルを記述し、SMF リポジトリに格納したほかのプロパティーからこれらのファイルを生成する方法を、`svcio` に伝えます。
   ステンシルに対応するようにサービスを構成するには、テンプレートとして使用するステンシルファイルと、生成される構成ファイルの両方のパスを含んだ管理対象の構成ファイルごとにプロパティーグループを追加します。プロパティーグループには次のプロパティーがあります。

`path`	構成ファイルの書き込み先のパス。`/etc/svc.conf` など。
`stencil`	`/lib/svc/stencils` に対する相対パスである、使用するステンシルファイルのパス。たとえば、`stencil` プロパティーの値が `svc.stencil` である場合、`/lib/svc/stencils/svc.stencil` ファイルが使用されます。
`mode`	構成ファイル (`path`) に対して使用するモード。644 など。
`owner`	構成ファイル (`path`) に対して設定する所有者。このプロパティーが設定されていない場合、ファイルの所有者は `svcio` を呼び出したユーザーになります。
`group`	構成ファイル (`path`) に対して設定するグループ。このプロパティーが設定されていない場合、グループは `path` のデフォルトグループになります。

Puppet ステンシルサービス

Puppet は、多くのシステムの構成を管理するためのツールキットです。Oracle Solaris では、Puppet アプリケーションは SMF で管理されます。

Puppet サービスの高度な概要

system/management/puppet パッケージをインストールすると、puppet:master と puppet:agent の 2 つの SMF サービスインスタンスが得られます。これらのインスタンスはデフォルトで無効になっています。

これらのインスタンスを有効にしたあと、次のコマンドを実行すると、puppet:master と puppet:agent の両方が契約サービスであることが示されます。

```
$ svcs -p puppet
STATE          STIME    FMRI
online         17:19:32 svc:/application/puppet:agent
               17:19:32     2565 puppet
online         17:19:32 svc:/application/puppet:master
               17:19:32     2567 puppet
```

次のコマンドは、契約サービスによって開始されたプロセスに関する少し詳しい情報を示します。

```
$ ps -o pid,args -p 2565,2567
  PID COMMAND
 2565 /usr/ruby/1.9/bin/ruby /usr/sbin/puppet agent --logdest /var/log/puppet/puppet-
 2567 /usr/ruby/1.9/bin/ruby /usr/sbin/puppet master --logdest /var/log/puppet/puppet
```

ps 出力からわかるように、puppet は /var/log/puppet 内のログファイルに書き込みます。

```
$ ls /var/log/puppet
puppet-agent.log  puppet-master.log
```

初期 Puppet 構成ファイル

Puppet では、/etc/puppet/puppet.conf という名前の構成ファイルを使用すると想定しています。/usr/sbin/puppet アプリケーションは、application/puppet サービスインスタンスに設定されたプロパティーからではなく、/etc/puppet/puppet.conf から構成情報を読み取ります。必要な構成ファイルを提供するために、それぞれの puppet インスタンスは、ステンシルファイルと configfile プロパティーグループを提供します。configfile プロパティーグループは、svcio ユーティリティーに対して実行し、指定された構成ファイルを作成するように指示します。ステンシルファイルは、正しい形式でサービスプロパティー値のデータを構成ファイルに書き込むために使用されます。

次のコマンドは、タイプ configfile のプロパティーグループにある、すべての puppet サービスプロパティーを示します。この出力は、puppet サービスの両方のインスタンスが、同じ値の同じ configfile プロパティーを持っていることを示します。各 puppet サービスインスタンスは、構成ファイルへのパス、構成ファイルのモード、およびステンシルファイルへのパスを提供します。

```
$ svcprop -g configfile puppet
svc:/application/puppet:master/:properties/puppet_stencil/mode astring 0444
```

```
svc:/application/puppet:master/:properties/puppet_stencil/path astring /etc/puppet/
puppet.conf
svc:/application/puppet:master/:properties/puppet_stencil/stencil astring puppet.stencil
svc:/application/puppet:agent/:properties/puppet_stencil/mode astring 0444
svc:/application/puppet:agent/:properties/puppet_stencil/path astring /etc/puppet/
puppet.conf
svc:/application/puppet:agent/:properties/puppet_stencil/stencil astring puppet.stencil
```

次のコマンドは、これらのインスタンスプロパティーが親サービスから継承されていることを確認します。

```
$ svccfg -s puppet listprop -l all puppet_stencil
puppet_stencil          configfile  manifest
puppet_stencil/mode    astring     manifest              0444
puppet_stencil/path    astring     manifest              /etc/puppet/puppet.conf
puppet_stencil/stencil astring     manifest              puppet.stencil
$ svccfg -s puppet:agent listprop -l all puppet_stencil
$ svccfg -s puppet:master listprop -l all puppet_stencil
```

インフラストラクチャーの場合、たとえば `puppet:agent1` および `puppet:agent2` インスタンスが必要になることがあります。その場合、69 ページの「Puppet 構成ファイルの変更」に示すように、プロパティー値をカスタマイズし、インスタンスごとにプロパティーを追加します。

次に、構成ファイル `/etc/puppet/puppet.conf` の初期コンテンツを示します。

```
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppetlabs.com/guides/configuring.html
# for details.
```

Puppet ステンシルファイル

ステンシルファイルのコンテンツは、構成ファイルに書き込まれるプロパティーとほかの情報を伝えます。`puppet_stencil/stencil` プロパティーの値である `puppet.stencil` パスは、`/lib/svc/stencils` に対する相対パスです。次に、ステンシルファイル `/lib/svc/stencils/puppet.stencil` のコンテンツを示します。

```
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppetlabs.com/guides/configuring.html
# for details.
; walk each instance and extract all properties from the config PG
$%/(svc:/$%s:(.*)/:properties)/ {
$%{$%1/general/enabled:?
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}
}
```

ステンシルファイルでは、`svc:/$%s:(.*)/:properties` (または `%1`) は、`svc:/application/puppet:agent/:properties` および `svc:/application/puppet:`

master/:properties に展開します。ここで .* (または %2) はすべてのインスタンスに一致します。インスタンス名は続いて、構成ファイルのブロックにラベルを付けるために使用されます。.* (または %3) の次のオカレンスは、%1 サービスインスタンスの config プロパティーグループ内のすべてのプロパティーに一致します。ステンシルは、svcio に対して、サービスインスタンスからのプロパティー名とそのプロパティーの値を構成ファイルに書き込むように指示します。

Puppet 構成ファイルの変更

67 ページの「初期 Puppet 構成ファイル」に記しているように、最初はリテラルコメント行だけが構成ファイルに書き込まれます。ステンシルファイルの general/enabled プロパティーの値のテストにより、構成ファイルへのプロパティー値の書き込みが防止されます。次のコマンドは、general/enabled プロパティーの値がデフォルトでは false であることを示します。

```
$ svcprop -p general/enabled puppet
svc:/application/puppet:master/:properties/general/enabled boolean false
svc:/application/puppet:agent/:properties/general/enabled boolean false
```

svcadm enable コマンドを使用してインスタンスを有効にしても、general/enabled プロパティーの値は変更しません。general/enabled プロパティーの値を true に変更しインスタンスを再起動すると、そのインスタンスの config プロパティーグループのすべてのプロパティーが構成ファイルに書き込まれます。

```
$ svccfg -s puppet:agent setprop general/enabled=true
$ svcprop -p general/enabled puppet:agent
false
$ svcadm refresh puppet:agent
$ svcprop -p general/enabled puppet:agent
true
$ svcadm restart puppet:agent
```

次のコマンドは、最初、config プロパティーグループの唯一のプロパティーが各インスタンスのログファイルへのパスであることを示します。

```
$ svcprop -p config puppet
svc:/application/puppet:master/:properties/config/logdest astring /var/log/puppet/puppet-
master.log
svc:/application/puppet:agent/:properties/config/logdest astring /var/log/puppet/puppet-
agent.log
```

有効になったインスタンスの config プロパティーが、インスタンス名でラベルが付けられたブロック内の構成ファイルに追加されました。

```
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppetlabs.com/guides/configuring.html
# for details.

[agent]
```

```
logdest = /var/log/puppet/puppet-agent.log
```

Puppet 構成ドキュメントでは、構成ファイルには [main]、[agent]、および [master] のブロックが存在する可能性があると記されています。[main] ブロック内の構成は、エージェントおよびマスターの両方に適用されます。Puppet エージェントの場合、[agent] ブロック内の構成が [main] ブロック内の同じ構成をオーバーライドします。Puppet マスターの場合、[master] ブロック内の構成が [main] ブロック内の同じ構成をオーバーライドします。エージェントとマスターに共通した構成に [main] ブロックを使用する場合は、puppet:main インスタンスとそのインスタンスに対して適切な config プロパティーを作成します。

次のコマンドは、Puppet 構成ファイルに構成を追加する方法を示します。

```
$ svccfg -s puppet:agent
svc:/application/puppet:agent> setprop config/report=true
svc:/application/puppet:agent> setprop config/pluginsync=true
svc:/application/puppet:agent> refresh
svc:/application/puppet:agent> exit
$ svcadm restart puppet:agent
$ cat /etc/puppet/puppet.conf
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppetlabs.com/guides/configuring.html
# for details.

[agent]

logdest = /var/log/puppet/puppet-agent.log
pluginsync = true
report = true
```

プロパティーの削除やプロパティー値の変更にも、同様のコマンドを使用できます。『Oracle Solaris 11.3 でのシステムサービスの管理』 の 第 4 章, 「サービスの構成」を参照してください。main インスタンスを追加するには、『Oracle Solaris 11.3 でのシステムサービスの管理』 の 「サービスインスタンスの追加」に示されているように svccfg add コマンドを使用します。

Kerberos ステンシルサービス

ステンシルを使用する Oracle Solaris サービスのもう 1 つの例が Kerberos です。次のコマンドは、configfile プロパティーグループが krb5_conf であり、ステンシルファイルが /lib/svc/stencils/krb5.conf.stencil であり、構成ファイルが /etc/krb5/krb5.conf であることを示します。

```
$ svcprop -g configfile svc:/system/kerberos/install:default
krb5_conf/disabled boolean true
krb5_conf/group astring sys
krb5_conf/mode integer 644
krb5_conf/owner astring root
krb5_conf/path astring /etc/krb5/krb5.conf
```

```
krb5_conf/stencil astring krb5.conf.stencil
```

これらの値は、次に示したサービスマニフェストで設定されます。これらは、`svccfg setprop` を使用して変更できます。

```
<property_group type="configfile" name="krb5_conf">
        <propval name="disabled" type="boolean" value="true" />
        <propval name="path" type="astring"
                value="/etc/krb5/krb5.conf" />
        <propval name="stencil" type="astring"
                value="krb5.conf.stencil" />
        <propval name="mode" type="integer" value="0644" />
        <propval name="owner" type="astring" value="root" />
        <propval name="group" type="astring" value="sys" />
</property_group>
```

ステンシルファイルは `/lib/svc/stencils/krb5.conf.stencil` で、結果として生成される構成ファイルは `/etc/krb5/krb5.conf` で確認できます。

索引