Tivoli System Automation for Multiplatforms 4.1.0.7

管理者とユーザーのガイド

SA88-7250-05

お願い

本書および本書で紹介する製品を ご使用になる前に、175 ページの『特記事項』に記載されている情報をお読みください。

「*System Automation for Multiplatforms* 管理者とユーザーのガイド」のこの版は、IBM Tivoli System Automation for Multiplatforms バージョン 4 リリース 1 モディフィケーション 0、 プログラム番号 5724-M00、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書は、SA88-7250-04 の改訂版です。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典：
SC34-2698-05
Tivoli System Automation for Multiplatforms 4.1.0.6
Administrator's and User's Guide

発行：
日本アイ・ビー・エム株式会社

担当：
トランスレーション・サービス・センター

目次

図

表

本書について

本書では、IBM Tivoli System Automation for Multiplatforms (System Automation for Multiplatforms) が提供するポリシー・ベースの自動リカバリー機能を実装および使用する方法について説明します。

System Automation for Multiplatforms を使用すると、AIX® クラスター (IBM® System p 上)、Linux® クラスター (IBM System x、System z®、System i®、System p 上)、および Windows クラスター (IBM System x 上) 上のリソースの可用性が高くなります。

本書の対象読者

本ガイドは、System Automation for Multiplatforms の自動化機能およびフェイルオーバー機能を使用する必要があるシステム管理者およびオペレーターを対象としています。

詳細情報の参照先

Tivoli System Automation ライブラリーは、本書 (Tivoli System Automation for Multiplatforms について説明しています) を含め、以下の資料から構成されています。

- *System Automation for Multiplatforms* 管理者とユーザーのガイド (SA88-7250-01)
- *Tivoli System Automation for Multiplatforms* インストールと構成のガイド (SA88-7249-01)
- *Tivoli System Automation for Multiplatforms* リファレンス・ガイド (SA88-7251-01)
- *Tivoli System Automation for Multiplatforms* 高可用性ポリシー・ガイド *(SA88-7252-01)*

資料一式を、次のサイトからダウンロードできます。

```
http://www.ibm.com/support/knowledgecenter/SSRM2X/welcome
```

Tivoli System Automation ライブラリーには、System Automation Application Manager について説明する以下の資料が用意されています (本書も含まれています)。

- *System Automation Application Manager Administrator's and User's Guide* (SC34-2701-00)
- *System Automation Application Manager Installation and Configuration Guide*、SC34-2702-00
- *System Automation Application Manager Reference and Problem Determination Guide*、SC34-2703-00

これらの資料は、以下のページからダウンロードすることができます。

```
http://www.ibm.com/support/knowledgecenter/SSPQ7D/welcome
```

IBM Tivoli System Automation ホーム・ページには、サポート・リンクおよび保守パッケージのダウンロードなど、役に立つ最新情報が記載されています。 IBM Tivoli System Automation のホーム・ページは 以下からアクセスできます。

```
www.ibm.com/software/tivoli/products/sys-auto-multi/
```

表記規則

本書では、以下の強調表示の規則を使用しています。

表 *1.* 本書の強調表示の規則	
太字	コマンド、サブルーチン、キーワード、ファイル、構造体、ディレクトリー、およびシステムによって名前が事前に定義されているその他の項目を示します。また、ユーザーが選択するボタン、ラベル、およびアイコンなどのグラフィカル・オブジェクトも 示します。

表 1. 本書の強調表示の規則 (続き)	
イタリック	ユーザーが指定する実際の名前または値のパラメーターを示します。
モノスペース	具体的なデータ値の例、画面に表示されるものと同様のテキスト例、プログラマーが作成するものと同様のプログラム・コードの一部の例、システムからのメッセージ、または実際に入力する必要がある 情報を示します。

本資料では、リソース、リソース・グループ、同値、および関係を示すために シンボルを使用します。使用するシンボルは以下のとおりです。

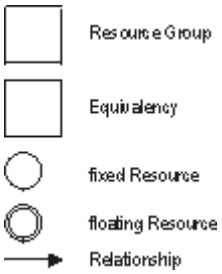


図 1. このガイドで使用するシンボル

ISO 9000

本製品の開発および製造において、ISO 9000 審査登録済みの品質システムが 使用されました。

RSCT 関連情報

以下の IBM Reliable Scalable Cluster Technology (RSCT) の資料は System Automation for Multiplatforms CD として入手できます。

- *RSCT* 管理ガイド
- *RSCT for AIX 5L:* テクニカル・リファレンス
- *RSCT for Multiplatforms:* テクニカル・リファレンス
- *RSCT* メッセージ
- *RSCT Diagnosis Guide*

RSCT について詳しくは、『IBM Cluster systems』を参照してください。

詳しくは、『Linux on IBM zSeries and S/390®: High Availability for z/VM® and Linux』IBM Redpaper を参照してください。

資料の入手方法

System Automation for Multiplatforms の資料は、以下の Web サイトでも入手可能です (リリース時点で有効)。

```
www.ibm.com/servers/eserver/clusters/library/
www.ibm.com/servers/eserver/zseries/software/sa/
www.ibm.com/software/sysmgmt/products/support/
```

連絡先電子メール・アドレス

以下は英語のみの対応となります。電子メールでのご連絡を希望される場合は、eservdoc@de.ibm.com までコメントをお寄せください。

このリリースの新機能

System Automation for Multiplatforms バージョン 4.1.0 の新機能について概説します。

新規 samcc コマンドによるコマンド行での操作の向上

System Automation for Multiplatforms バージョン 4.1.0.2 には新規コマンド samcc が追加されました。このコマンドをコマンド行インターフェースでの操作コンソールとして使用できます。詳しくは、「」を参照してください。

追加プラットフォーム・サポート

System Automation for Multiplatforms バージョン 4.1.0.1 では以下の新規プラットフォームをサポートします。

- SUSE SLES 12 (64 ビット)
- Red Hat RHEL 7 (64 ビット)
- Ubuntu 14.04 (64 ビット): System x、Power Systems (リトル・エンディアンのみ)

System Automation for Multiplatforms バージョン 4.1.0.2 では以下の新規プラットフォームをサポートします。

- Red Hat RHEL 7.1 on Power Systems Little Endian (64 ビット)

System Automation for Multiplatforms バージョン 4.1.0.3 では以下の新規プラットフォームをサポートします。

- AIX 7.2

System Automation for Multiplatforms バージョン 4.1.0.4 では以下の新規プラットフォームをサポートします。

- Ubuntu 16.04 (64 ビット): System x、Power Systems (リトル・エンディアンのみ)。

詳しくは、「*System Automation for Multiplatforms* インストールと構成のガイド」を参照してください。

System Automation for Multiplatforms バージョン 4.1.0.5 では以下の新規プラットフォームをサポートします。

- SUSE SLES 15 (64 ビット)
- Ubuntu 18.04 (64 ビット): System x、Power Systems (リトル・エンディアンのみ)。

System Automation for Multiplatforms バージョン 4.1.0.5 では以下のサポートが追加されました。

- SAP Netweaver 7.5.3 ENSA2

System Automation for Multiplatforms バージョン 4.1.0.6 では以下の新規プラットフォームをサポートします。

- Red Hat RHEL 8 (64 ビット)
- Ubuntu 20.04 (64 ビット): System x, Power Systems (リトル・エンディアンのみ)

System Automation for Multiplatforms バージョン 4.1.0.6 では以下のサポートが追加されました。

- S/4HANA 1809 の SAP NetWeaver サポートの追加
- S/4HANA 1909 の SAP NetWeaver サポートの追加
- Oracle 19c のサポートの追加
- SAP HANA 2.0 SPS 04 リビジョン 046 のサポートの追加

System Automation for Multiplatforms バージョン 4.1.0.7 では次の新規プラットフォームがサポートされます。

- AIX 7.2 TL5

System Automation for Multiplatforms バージョン 4.1.0.7 では次のサポートが追加されました。

- S/4HANA 2020 の SAP NetWeaver サポートの追加
- SAP HANA 2.0 SPS 05 リビジョン 050 のサポートの追加

SAP の高可用性ポリシーの改善

SAP Central Services 高可用性ポリシーは、別個に課金される System Automation for Multiplatforms のオプション・フィーチャーとして使用可能です。この SAP Central Services 高可用性ポリシーが、SAP Netweaver テクノロジーに適応するようになりました。

ユーザーは、システム自動化ポリシーに干渉することなく、SAP ユーザー・インターフェースを使用して SAP Netweaver スタックの開始と停止を実行することができます。SAP Software Update Manager では、更新プロセス中にシステム自動化を使用不可にすることなく、Netweaver ソリューションを更新できます。

サポートされる SAP 構成オプションは、SAP Central Services のフェイルオーバーの Java、ABAP、および DUAL スタック・サポートです。また、以下の構成オプションもサポートされます。

- アプリケーション・サーバー (主要アプリケーション・サーバーと追加のアプリケーション・サーバーの代わりに再始動)
- SAProuter のフェイルオーバー
- SAP Web ディスパッチャーのフェイルオーバー
- データベースに対する依存関係サポートを行ってからの始動

System Automation for Multiplatforms バージョン 4.1.0.2 では以下のサポートが追加されました。

- SAP HANA System Replication フェイルオーバー

サポートされる SAP カーネルのバージョンは 7.20 以上です。

詳しくは、「*System Automation for Multiplatforms* 高可用性ポリシー・ガイド」を参照してください。

アプリケーション障害に関する情報の収集

samwhy プログラムは、System Automation で制御されているアプリケーションに関してアプリケーション障害の検出とその障害の分析ができるようになっている、使いやすい簡易ツールです。samwhy は、発生した事象についてオペレーターが把握するのを支援し、それに対する System Automation の対応の仕方について理由を明らかにするものです。

詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

エンドツーエンド自動化アダプターの高可用性の単純化

追加の自動化ポリシーおよび仮想 IP アドレスが不要になっています。

詳しくは、「System Automation for Multiplatforms インストールと構成のガイド」を参照してください。

非 root ユーザーでのエンドツーエンド自動化アダプターの実行

デフォルトでは、エンドツーエンド自動化アダプターは root ユーザーで実行されます。このリリースでは、非 root ユーザーで実行されるようにアダプターをセットアップすることもできるようになりました。

詳しくは、「System Automation for Multiplatforms インストールと構成のガイド」を参照してください。

第 1 章 製品概要

System Automation for Multiplatforms では、Linux および AIX のクラスター上で稼働するアプリケーションの可用性を管理します。サポートされているプラットフォームについて詳しくは、「*Tivoli System Automation for Multiplatforms インストールと構成のガイド*」を参照してください。System Automation for Multiplatforms では、以下の機能を提供します。

高可用性

System Automation for Multiplatforms は、システムが継続的に使用可能となる高可用性環境を提供します。その自己修復インフラストラクチャーは、システム問題によって引き起こされるダウン時間を防ぎます。リカバリー・インフラストラクチャーにより、システムの不正な操作、トランザクション、およびプロセスが検出され、ユーザーの作業に影響を与えることなく修正アクションが開始されます。

System Automation for Multiplatforms は、障害の高速検出、およびアプリケーションのコンポーネントおよびその関係に関する高度な知識を使用することにより、メインフレームと同様の高可用性を提供します。これにより、オペレーター介入を 必要とすることなく、Linux または AIX クラスター内の 同一または異なるシステム上で、失敗したリソースと完全なビジネス・アプリケーションを素早く、確実にリカバリーすることができます。 オペレーターは、手動モニターから解放され、アプリケーションのコンポーネントと関係を覚えておく必要がなくなるため、 オペレーター・エラーのリスクが減少します。

自動リカバリー

System Automation for Multiplatforms には、障害時にシステムとアプリケーションが迅速に再開されるようにするための自動リカバリー・メカニズムが用意されています。

System Automation for Multiplatforms は、Linux または AIX クラスター内の 同一または異なるシステム上で、失敗したリソースと完全なビジネス・アプリケーションを素早く、確実に再始動します。 このリカバリー機能により、アプリケーションとサービス停止のリスクが大幅に減少します。

フェイルオーバー

System Automation for Multiplatforms は、クラスター内でのアプリケーションの自動移動とそれに関連したアクションを提供することにより、クラスター環境を管理します。

System Automation for Multiplatforms は、関連するリソース間の関係をクラスター規模で管理します。ノード間でアプリケーションを移動する必要がある場合、開始/停止関係、ノード要件、および何らかの事前または事後アクションは、System Automation for Multiplatforms によって 自動的に処理されます。これにより、システム・オペレーターによる手動でのコマンド入力が不要になり、オペレーター・エラーのリスクが減少します。

高信頼性スケーラブル・クラスター・ テクノロジー (RSCT)

Reliable Scalable Cluster Technology (RSCT) は、System Automation for Multiplatforms に完全に統合された製品です。RSCT は、AIX および Linux 用の包括的なクラスタリング環境を提供する ソフトウェア・プロダクトのセットです。System Automation for Multiplatforms は RSCT インフラストラクチャーを使用して、改善されたシステム可用性、スケーラビリティー、および使いやすさをクラスターに提供します。

RSCT の基本サブシステムがクラスター内通信を処理します。このクラスター内通信では、複数の通信チャネルを使用できます。IP ベースの通信では、ネットワーク・プロトコルのユーザー・データグラム・プロトコル (UDP) および TCP/IP を使用します。非 IP ベースの通信では、SCSI ディスクの専用領域への書き込みを行います。

RSCT には、次の基本サブシステムが含まれています。

RMC (リソース・モニターおよび制御サブシステム):
RMC は、単一システムのモニターおよびクラスター内ノードのモニターに使用されます。クラスター内では、RMC は、クラスター全体を通して、サブシステムとリソースに対するグローバル・アクセス

を提供します。このため、RMC は、クラスター用の単一のモニターおよび管理インフラストラクチャーとなります。

HAGS (High Availability Group Services サブシステム):
HAGS は、調整、メッセージング、および同期の分散サービスです。このサービス は、リカバリー・リソース・マネージャーを調整します。HAGS のデフォルトの クラスター内通信プロトコルは UDP です。

HATS (High Availability Topology Services サブシステム):
HATS は、アダプター およびノード障害検出のためのスケーラブルなハートビート、およびピア・ドメインにおける信頼性のあるメッセージング・サービスを提供します。

RSCT ハートビート機構は、ブロードキャスト ping を時折実行します。これは、ネットワーク・インターフェース・アダプターの可用性を判別できない場合に顕著です。この機能の目的は、このブロードキャスト ping を送信するネットワーク・インターフェース・アダプターがまだ作動可能であるかどうかを検出することにあります (これは、他のノードがこのブロードキャスト ping に応答するかどうかによって判別できます)。HATS のデフォルトの クラスター内通信プロトコルは UDP です。

SRC (システム・リソース・コントローラー):
システム・リソース・コントローラー (SRC) は、最初に始動され、RSCT と System Automation for Multiplatforms を実行するのに必要な他のすべてのデーモン・プロセスを始動するブートストラップ・プロセスです。これは、障害時のデーモン・プロセスの再始動も行います。システム・リソース・コントローラーは、システム・マネージャーによるサブシステムの作成と制御を容易にして、コマンドとサブルーチンのセットを提供します。システム・リソース・コントローラーは、`inittab` から開始されます

RSCT および System Automation for Multiplatforms によって提供されるリソース・マネージャー

リソース・クラスは、リソース・マネージャー (RM) によって管理されます。特定のリソースの管理を担当するリソース・マネージャーは、リソース・クラスによって異なります。リソース・マネージャーは、リソースと RMC との間に位置する ソフトウェア層です。リソース・マネージャーは、システム・リソース・コントローラー (SRC) によって制御されるデーモンとして実行されます。

RSCT は、単一システム上およびクラスター全体を通じて、System Automation for Multiplatforms リソースを管理するリソース・マネージャーのコア・セットを提供します。システム・リソース・コントローラーは、システム・マネージャーによるサブシステムの作成と制御を容易にして、コマンドとサブルーチンのセットを提供します。RSCT リソース・マネージャー・セットは、System Automation for Multiplatforms に固有の次のリソース・マネージャーによって拡張されます。

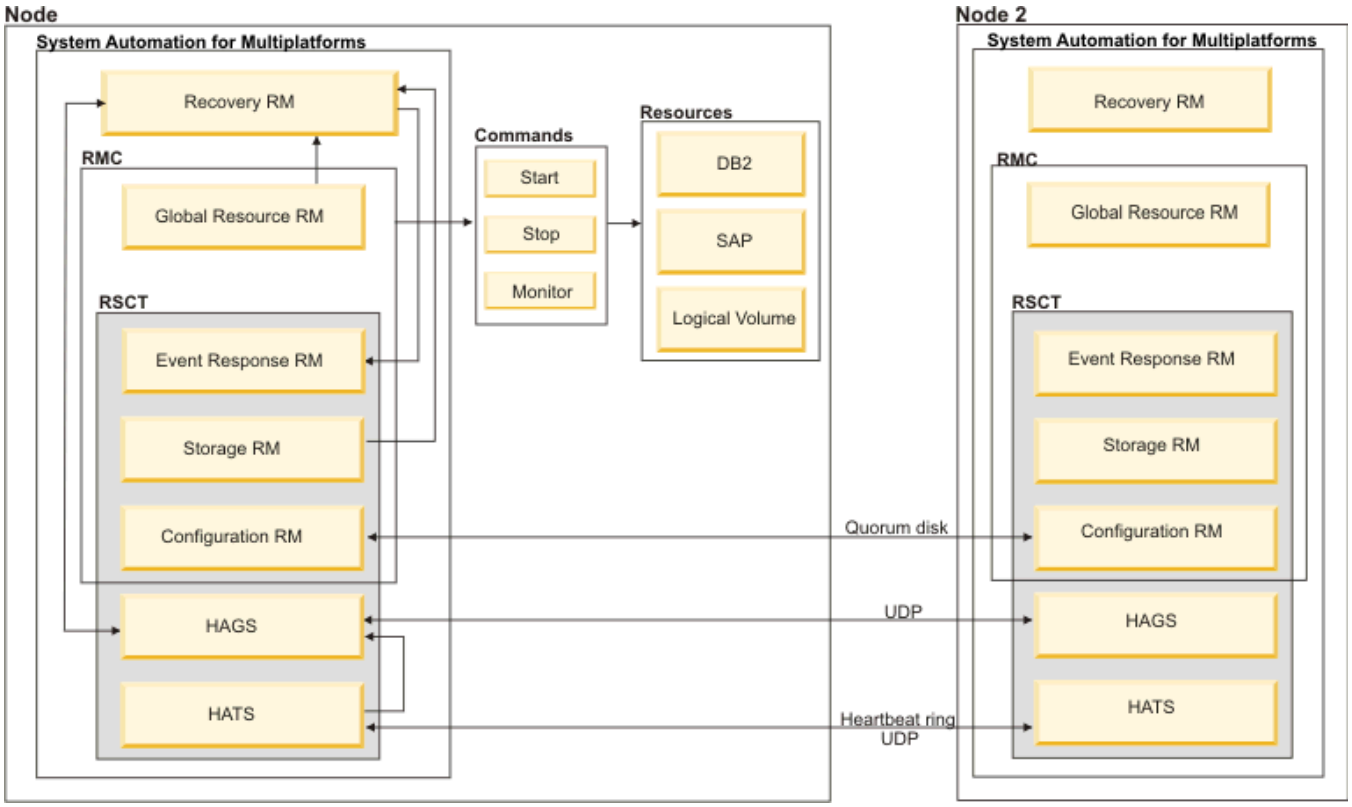


図 2. *System Automation for Multiplatforms* アーキテクチャーの概要

リカバリー・リソース・マネージャー (リカバリー *RM*)

リカバリー RM (`IBM.RecoveryRM`) は、System Automation for Multiplatforms の決定エンジンとして機能します。

関係とリソースの可用性が定義される自動化ポリシーを作成すると、そのポリシー情報がリカバリー RM に提供されます。 リカバリー RM は、1 つのリカバリー RM がマスターとして指定されて、クラスター内のすべてのノードで稼働します。マスターのリカバリー RM は、ストレージ RM やグローバル・リソース RM などの さまざまなリソース・マネージャーによって提供される モニター情報を評価します。介入を必要とする状態が発生したら、リカバリー RM は、必要に応じてリソースに対する開始または停止操作を行う決定を実行します。

グローバル・リソース *RM*

グローバル・リソース RM (IBM.GblResRM) は、以下の 2 つのリソース・クラスをサポートします。

IBM.Application
: IBM.Application リソース・クラスは、一般的なアプリケーション・リソースの 動作を定義します。このクラスは、処理を開始、停止、およびモニターするために 使用できます。汎用クラスとして、フレキシブルであり、さまざまなリソースをモニター および制御するために使用できます。自動化するほとんどのアプリケーションは、このクラスのアプリケーションです。IBM.Application の 詳細については、108 ページの『IBM.Application リソース・クラス』を参照してください。

IBM.ServiceIP
: このアプリケーション・クラスは、インターネット・プロトコル (IP) アドレス・リソースの 動作を定義します。これにより、アダプターに IP アドレスを割り当てることができます。事実上、これにより IP アドレスをノード間で「浮動」させることができます。IBM.ServiceIP の 詳細については、124 ページの『IBM.ServiceIP リソース・クラス』 を参照してください。

イベント応答リソース・マネージャー (イベント応答 *RM*)

イベント応答 RM (IBM.ERRM) では、この条件に対する応答として 1 つ以上のアクション を起動する任意の条件を待機できます。

例:

条件:
ファイル・システムの使用率がディスク容量の 95% に達した場合、リソースを停止します。

アクション:
リソース・グループが「オフラインに失敗」になると、E メールが送信されます。

ストレージ・リソース・マネージャー

ストレージ・リソース・マネージャー (IBM.StorageRM) は、自動化ドメイン内のストレージ・リソースのモニターと制御を行います。ストレージ・リソース・マネージャーは、それが提供するリソース・クラスのインスタンスに物理および論理ストレージ・エンティティーをマッピングすることによって、ピア・ドメイン内での RMC とこれらのエンティティー間のインターフェースを提供します。

ストレージ・リソース・マネージャーは、自動化ドメイン内の各ノード上でデーモン・プロセスとして実行され、ローカル接続された物理ディスク (および関連ストレージ・エンティティー) に関する情報を収集し、これらの情報をリソース・クラス・インスタンスにマップします。各個別ノードからのこれらの個別のストレージ・リソース・ビューが一つに集められ、ストレージ・リソース・マネージャーに、自動化ドメイン内のストレージ・リソースのグローバル・ビューが提供されます。

共用ストレージ環境 (自動化ドメイン内の複数ノードが同じディスク・サブシステムに対するアクセス権を持ち、したがって、ディスク・サブシステム内に含まれる同じディスクに対するアクセス権を持つ) では、ストレージ・リソース・マネージャーは、ストレージ・リソースのグローバル・ビューによって、共用ディスクを識別し、予備ハードウェアによる順次アクセスを提供できるようになります。

構成リソース・マネージャー (構成 *RM*)

構成 RM (IBM.ConfigRM) は、クラスター定義に使用されます。また、クラスターのある部分が通信を失った場合にデータ保全性を確保する手段である クォーラム・サポートが提供されています。

テスト・リソース・マネージャー (テスト *RM*)

テスト RM (IBM.TestRM) はテスト・リソースを管理し、それらの運用状態を操作するために使用できます。テスト RM は実際のリソースは制御しません。

テスト RM は、ピア・ドメイン・モードでのみ作動可能であり、リソース・クラス IBM.Test を提供します。テスト RM についての詳細は、129 ページの『テスト・リソース・マネージャーの使用』を参照してください。

クォーラム・サポート

構成および操作クォーラムを使用して、クラスター内のクリティカル・リソースを保護できます。

概要

クラスター (ピア・ドメインともいいます) は、クラスター内のエレメント間での通信が 不可能になった場合、2 つ以上のサブクラスターに分割されます。サブクラスターは相互に認識していないため、他のサブクラスターの 1 つで既に稼働中のアプリケーションの新規インスタンスを、System Automation for Multiplatforms が開始することがあります。例えば、障害からのリカバリーを実行するために、アプリケーションが共用ディスクへのアクセスを必要とする場合に、ディスクへの同時アクセスがあると、データ破壊が発生する可能性があります。

そのようなリソースを、クリティカルであると 表現します。クリティカル・リソースは、どの時点においても複数のノードで 稼働させることができないリソースです。そのようなリソースが 2 つ以上の別個のノードでアクティブである場合、クラスターのデータ保全性が危険にさらされます。こうしたクリティカル・リソースを保護するため、System Automation for Multiplatforms では、1 つのサブクラスターのみが存続し、その他のサブクラスターは解除される、というようにします。System Automation for Multiplatforms は、このようにしてシステム障害またはネットワーク区画化によって発生するデータ破壊を防止します。

クラスターが複数のサブクラスターに分割される場合、構成リソース・マネージャーは、どのサブクラスターに大多数のノードがあるかを判別します。大多数とは、そのサブクラスターに、クラスター内で定義されているすべてのノードの半分を超えるノードがある場合を指します。大多数のノードがあるサブクラ

スターは操作クォーラムを持ちます。このサブクラスターは存続してアクティブ・クラスターとなり、その他のサブクラスター (1 つ以上) は解除されます。

物理マシンが複数ある仮想環境、例えば AIX LPAR や、z/VM の下の Linux on System z ゲストでは、物理マシン間でノードを分散させる必要があります。ノードの大部分をホストしているマシンが使用不可になった場合に、クォーラムが失われなくすることができます。

クリティカル・リソースは、構成クォーラムおよび操作クォーラムによって保護されます。

構成クォーラム

構成クォーラムは、クラスター内の構成変更がいつ受け入れられるのかを決定します。クラスター定義の保全性は、以下の大多数のルールによって 確保されます。クラスターの構成に影響を与える操作は、n/2+1 のノードが アクティブである場合にのみ実行可能です。この n は、クラスター内で定義されている ノードの数です。ただし、特定の操作に関しては、大多数ルールが無効になるか、別の構成クォーラム・ルールが適用されることがあります。

- ちょうど半分のノードがオンラインである場合、および少なくとも 1 つの オフライン・ノードから正常に構成を除去できる場合は、**rmrpnode** コマンドを 使用して、ノードを除去できます。また、このコマンドの `-f` オプションを使用して、大多数ルールを無効にすることができます。
- **startrpdomain** コマンドのクォーラム・ルールは n/2 ですが、すべてのノード (-A) オプションまたはローカル・ノード (-L) オプションを使用してこれを無効にすることができます。

注 : タイ状態の場合は、コマンド **chrg -o** online (または offline) `group_name` を使用して、リソース・グループを開始および停止できます。

RSCT について詳しくは、「Reliable Scalable Cluster Technology」を参照してください。

操作クォーラム

操作クォーラムは、他のリソースとの競合を発生させることなく、リソースを安全にアクティブにすることができるかどうかを決定するために使用します。操作クォーラムは、オンライン・ノードの数、および特定のタイ状態を解決するためのタイ・ブレーカーに基づいて決定されます。サブクラスターの半数を超えるノードがアクティブの場合、そのサブクラスターは操作クォーラムを所有します。

操作クォーラムが存在する場合、System Automation for Multiplatforms でリソースまたはリソース・グループを操作でき、それらをオンラインにすることができます。クォーラムが存在しない場合、System Automation for Multiplatforms はリソースに対してアクションを実行できません。

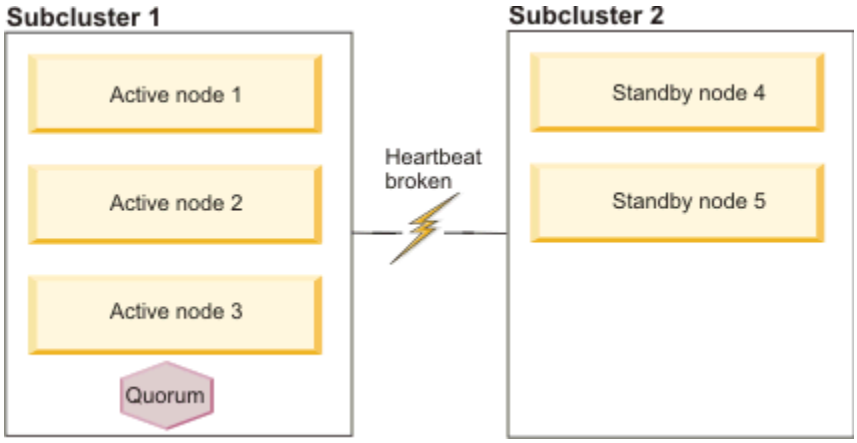


図 *3.* 異なる *2* つのサブクラスターでノードの数が不揃いである例

大多数のノードがあるサブクラスターは、操作クォーラムを保持します。クリティカル・リソースは、アクティブにできます。5 ページの図 3 では、サブクラスター 1 のノードはすべてアクティブのままであり、サブクラスター 2 のノードはシャットダウンされています。

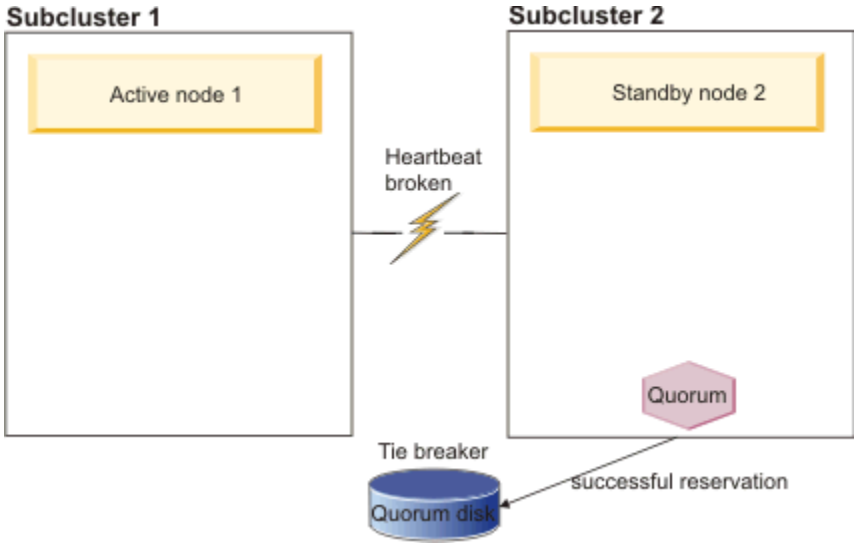


図 *4.* 異なる *2* つのサブクラスターでノードの数が揃っている例

サブクラスターのノード数が同じである場合は、いずれのサブクラスターがクォーラム・ディスクを先に予約するかによって、タイ・ブレーカーが、クォーラムを持つサブクラスターを決定します。6 ページの図 4 では、サブクラスター 2 がクォーラム・ディスクを先に予約したため、サブクラスター 1 のノードがシャットダウンされます。サブクラスター 2 のノードが開始されます。

AIX および *Linux* 上でのクリティカル・リソース保護方式

クォーラムを失った AIX または Linux サブクラスターでクリティカル・リソースがアクティブな場合、ConfigRM は、サブクラスター内の各ノードの「CritRsrcProtMethod」属性を使用して、どのようにしてシステムを終了させるかを決定します。保護方法は、カーネル・パニックのシミュレーションを使用した即時システム・シャットダウンに基づいています。以下の 7 つの保護方式があります。

表 *2. CritRsrcProtMethod* の設定

意味	**値**
オペレーティング・システムをハード・リセットし、リブートする (デフォルト)	1
オペレーティング・システムを停止する	2
オペレーティング・システムを同期ありでハード・リセットし、リブートする	3
同期ありで停止する	4
保護なし。システムは作動を続行する	5
RSCT サブシステムを終了して再始動する	6
すべての保護を無効にする	7

例:

1. ドメイン規模のクリティカル・リソース保護メソッドを変更するには、 `-c` フラグを指定して chrsrc を入力します。

   ```
   chrsrc -c IBM.PeerNode CritRsrcProtMethod=3
   ```

2. 単一ノードで、ドメイン規模のデフォルトのクリティカル・リソース保護メソッドをオーバーライド するには、`-s` フラグを指定し、ノードを識別するストリングを選択して、`chrsrc` コマンドを入力します。

   ```
   chrsrc -s"Name==’node1’" IBM.PeerNode CritRsrcProtMethod=3
   ```

sync を使用した保護方法は、オペレーティング・システムが停止またはリセットされる前にファイルシステム・バッファーをディスクに送信します。これにより、ファイルシステムのデータ損失、またはデータの不整合が発生する可能性が低くなります。sync を使用した保護方法は、特定のシチュエーションにおいては安全な解決方法にならない場合があります。 例えば、ファイル・システムのフラッシュ中に、操作クォーラムを取得したサブクラスター内で開始した直後の他のアプリケーション構成要素からデータがアクセスされる可能性があります。 特定のシステムおよびアプリケーション環境で、このようなシチュエーションが発生する可能性があるかどうかを判別する必要があります。

使用されている保護方法に関わりなく、ジャーナリング・ファイル・システムを使用する必要があります。これにより、ファイルシステム自体の破損を防ぐことができ、 システム・リセット後のファイルシステムのリカバリー機能が大幅に向上します。

ノードに対する保護方法はさまざまです。クラスター全体の各ノードに対して同じ保護方法が設定されます。

ディスク・ハートビートの使用可能化

ディスク・ハートビートを使用可能にして、クラスター環境内のデータ保全性を確保できます。

ディスク・ハートビートでは、ネットワーク障害とノード障害を区別することができるため、ディスク・ハートビートを使用すると、クラスター分割が発生する可能性を減らすことができます。

ネットワーク障害は、7 ページの図 5 に示すように、ノード間および 1 つのノードから共有ディスクへのネットワーク接続が失敗した場合に発生します。

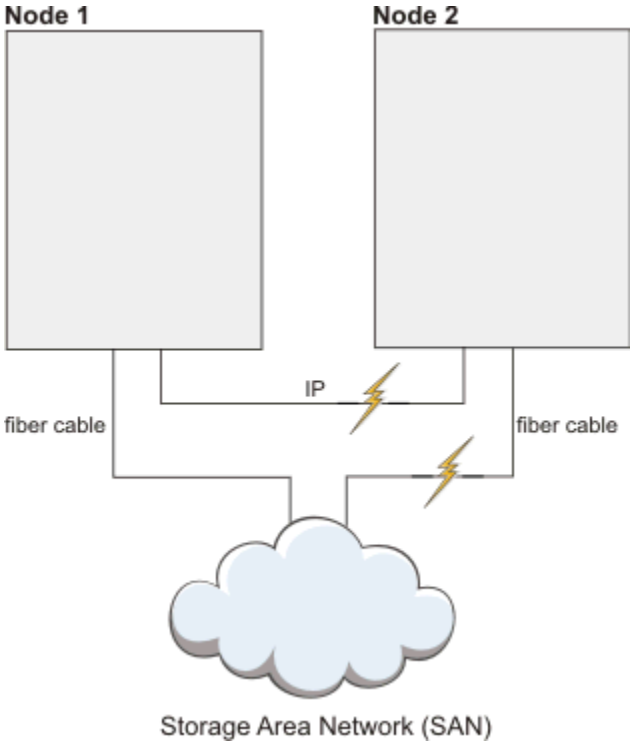


図 *5. 2* つのノードと共有ディスクの場合のネットワーク障害

8 ページの図 6 に示すように、1 つのノードがこれ以上到達できない場合にノード障害が発生します。

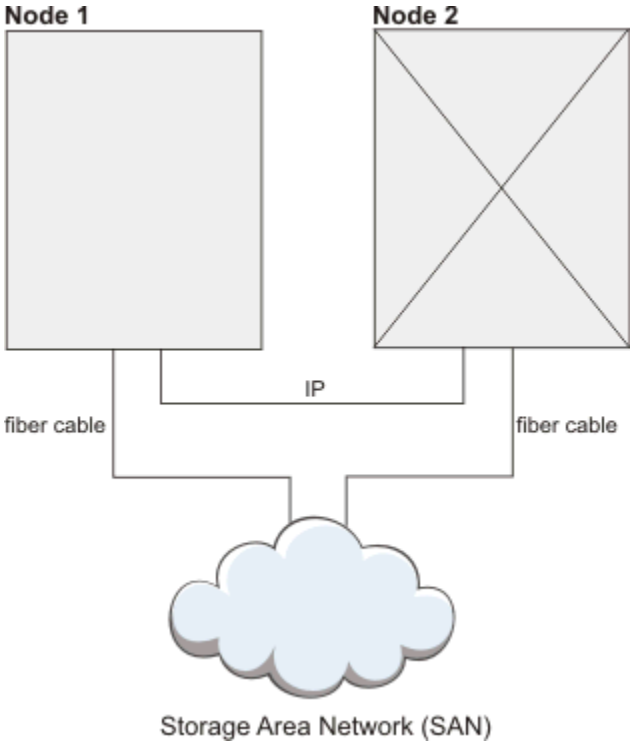


図 *6. 2* つのノードと共有ディスクの場合のノード障害

クラスター分割を回避できる場合、クリティカル・リソースの保護は必要ありません。システムをリブートする必要はありません。データ保全性の問題も回避されます。

クラスター分割が発生した場合、ハートビートを発信しているディスクにアクセスできなくなった ノードは、重要データにもアクセスできなくなります。クリティカル・リソースの保護は、データの破壊を防ぐのに役立ちます。ディスク・ハートビートは、ディスクにアクセスできないノードがデータを変更できないため、クリティカル・リソース保護ルールの緩和を許可します。

注 :

1. ディスク・ハートビート機能は、ピア・ドメインが既にオンラインである場合に限って使用可能にすることができます。
2. ディスク・ハートビート機能は、2 つのノード間にのみ定義できます。3 個以上のノードの場合は、各ペアを個別に接続する必要があります。

適切な物理ボリューム、論理ボリューム、または Linux 上のマルチパス・デバイスを見つけます。このボリューム上のデータが消去されます。次のコマンドを使用して、ハートビート・インターフェース・リソースを作成します。

```
CT_MANAGEMENT_SCOPE=2
mkrsrc IBM.HeartbeatInterface attributes [Force=0|1]
```

属性

名前
36 文字までの任意の名前。

DeviceInfo
有効なディスク ID またはボリューム ID:

- /dev/hdisk: ロー・ディスク
- LVID: 論理ボリューム
- MPATH: マルチパス・デバイス
- PVID: 物理ボリューム

CommGroup
IBM.CommunicationGroup にあるインスタンスの名前。Force パラメーターが 1 の場合に作成されます。

NodeNameList
このハートビート・インターフェースのノード・ペアであり、例えば、{'node1','node2'} です。

MediaType
2 (ディスク)

ハートビート・リングごとに通信グループが 1 つ作成されます。これは、 標準的なネットワーク・ベースのハートビート機能でも同じです。通信グループは、 ハートビート・デバイスと一緒に作成されます。 通信グループは、ネットワーク・ベースのグループ同様に調整 できます。ディスク・ハートビート機能では、PingGracePeriodMilliSec は変更できません。

セットアップしたディスク・ハートビートを確認するには、以下のタスクを実行します。

- システム・セットアップの構成で、ディスク・ハートビート機能に使用されるディスクがピア・ノードによって予約されていないことを確認します。
- ディスク・ハートビート機能は、以下のコマンドを使用してテストできます。

```
dhb_read -p <device-name> -t    # run it on a sender side
dhb_read -p <device-name> -r      # run it on a receiving side
```

  総合的な検証のためには、送信側ノードと受信側ノードを交換して、これらのコマンドを再実行してください。テストが動作しない場合は、ディスク予約のためサポートされていない可能性があるか、あるいはシステム・セットアップまたは構成の互換性がありません。
- ノード間で以下のシステム・コールが正しく動作することを確認します。

```
open("<dev>", O_RDWR|O_DIRECT), pread() and pwrite();
```

タイ・ブレーカーのタイプ

クラスターがサブクラスターに区画化されており、それぞれのノード数が同じ、というタイがある場合、構成リソース・マネージャーは、タイ・ブレーカーを使用して、いずれのサブクラスターに操作クォーラムを持たせるかを判別します。サブクラスターは、タイ・ブレーカーが予約されている場合、操作クォーラムを所有します。

以下のタイプのタイ・ブレーカーがあります。

表 3. タイ・ブレーカーのタイプ

タイ・ブレーカー	オペレーティング・システム	説明
Operator	AIX、Linux	このタイ・ブレーカーは、オペレーターまたは管理者に判断を求めます。管理者が明示的にタイをブレークするまで、どのサブクラスターも 操作クォーラムを持ちません。操作クォーラムの状態は PendingQuorum に設定され、ネットワークが修復されるか、障害になったノードが修復されてオンラインになるか、またはオペレーターがどのサブクラスターを適用し、どのサブクラスターを解除するかを決定するまで、その状態が維持されます。これは、各アクティブ・サブクラスターのノードで ResolveOpQuorumTie アクションを開始することによって実行されます。
Fail	AIX、Linux	このタイ・ブレーカーは、疑似タイ・ブレーカーです。つまり、タイを解決しません。どのサブクラスターも操作クォーラムを持ちません。
SCSI	Linux on POWER®、および Linux on System x	このタイ・ブレーカーは、クラスターのすべてのノードで SCSI ディスクが共用されていることを前提とします。タイ・ブレーカーの予約は、SCSI reserve コマンドまたは persistent reserve コマンドによって実行されます。

表 3. タイ・ブレーカーのタイプ (続き)		
タイ・ブレーカー	**オペレーティング・システム**	**説明**
SCSIPR	Linux on System x および Linux on System z	このタイ・ブレーカーは、クラスターのすべてのノードで SCSI ディスクが共用されていることを前提とします。タイ・ブレーカーの予約は、SCSI persistent reserve コマンドによって実行されます。
ECKD	Linux on System z	このタイ・ブレーカー・タイプは、クラスターのすべてのノードで ECKD ディスクが共用されていることを前提とします。ディスクへの排他的アクセスは、ECKD reserve コマンドを使用して実行されます。
DISK	AIX	DISK タイ・ブレーカーを使用する場合は、AIX デバイス名を使用して SCSI 物理ディスクまたは SCSI 型物理ディスクを指定できます。また、SCSI ディスクがクラスターの 1 つ以上のノードにより共用されることを前提としています。タイ・ブレーカーの予約は、SCSI reserve コマンドまたは persistent reserve コマンドによって実行されます。このタイプのタイ・ブレーカーを作成する場合は、物理ディスクを識別する `DeviceInfo` 永続リソース属性を設定する必要があります。SCSI 物理ディスクおよび SCSI 型ディスクのみがサポートされ ています。ファイバー・チャネル、iSCSI、および Serial Storage Architecture Connections を介して接続されている物理ディスクが適しています。
EXEC	AIX、Linux	この汎用タイ・ブレーカー・インプリメンテーションは、タイ・ブレーカー操作のためにカスタム実行可能プログラムを呼び出します。ネットワーク・タイ・ブレーカー `samtb_net` はこの製品に組み込まれていて、EXEC タイ・ブレーカー実行可能プログラムとしてインプリメントされています。詳しくは、System Automation for Multiplatforms インストールと構成のガイドを参照してください。

奇数のノードを含むクラスターが何らかの理由でサブクラスターに区画化された場合、半数より多いノードが使用可能なサブクラスターの方にクォーラムがあります。例えば、3 つのノードのクラスターの場合、2 つの使用可能なノードがあるサブクラスターがクォーラムを保持します。残りの単一ノードのサブクラスターは、操作クォーラムも構成クォーラムも保持しないため、そのノードではどのリソースも開始されません。そのノードで既にクリティカル・リソースが稼働中の場合は、定義されたクリティカル・リソース保護方式がそのノードに適用されます。詳しくは、6 ページの『AIX および Linux 上でのクリティカル・リソース保護方式』を参照してください。

クラスター内のノード数が偶数であり、クラスター分割時に一方のサブクラスターで半数のノードを含んでいる場合、タイ・ブレーカーは、いずれのサブクラスターでクリティカル・リソースを稼働させるかを決定します。タイ・ブレーカーを獲得する競争に負けた、クリティカル・リソースを持つノードは、リソース保護の対象になります。これは、それらが即時に停止されるかリブートされることを意味します。

オペレーターのアクションなしで自動的なタイ・ブレークを実行するには、ディスク・タイ・ブレーカー (Linux on System x または Linux on Power® の場合は SCSI、System z 上の Linux の場合は ECKD、AIX の場合は DISK) を使用します。 ディスク・タイ・ブレーカーは、すべてのクラスター・ノードからアクセス可能な共用ディスクです。

SLES 10 x/Linux システム : 始動時に、RSCT は `i8xx_tco` および `i6300esb` などのウォッチドッグ・モジュール (必要な softdog モジュールをロードするようにプリインストールされる場合があります) を削除します。ハードウェア関連のウォッチドッグ・モジュールを System Automation for Multiplatforms と並列に実行することはできません。

VMTIMEBOMB 機能

z/VM 下で、System z 上の Linux 用に実行されている `vmtimebomb` 機能は、z/VM が保留されるシナリオのデータ保護を確実に行いますが、ゲスト Linux システムは実行を継続します。これは、前のセクションで説

明した保護方法の新しいインプリメンテーションです。トポロジー・サービスの観点から見ると、vmtimebomb 機能は特殊な VM vmwatchdog モジュールを介して間接的にアクセスされるに過ぎません。このモジュールは、ユビキタスの Linux softdog ウォッチドッグ・モジュールに似ています。

ウォッチドッグ・モジュールは、ある種の突発的な障害時にオペレーティング・システムを自動的に再始動することにより、永続的な障害を防ぐことを意図しています。通常、アプリケーションは、アクティブに、かつ定期的にウォッチドッグを「ping」して (通常は特殊なデバイスに書き込むことによって)、これが存続していることを示します。これには、システム全体の正常性を示すという役割があります。アプリケーションが活動状態を示すのに失敗した場合、重大な誤動作があることを意味します。

アプリケーションから「すべて良好」のレポートが届かなくなったときは、ウォッチドッグの状況をモニターしてアクションを実行する必要があります。softdog の場合は、Linux オペレーティング・システムが実行します。特に重大な状況では、オペレーティング・システムに障害が発生し、応答できない場合があります。vmwatchdog の場合は、外部の監視を z/VM 制御プログラムが操作するため、最悪の場合でも仮想マシン上でオペレーティング・システムを再始動できます。vmwatchdog は、z/VM 5.1.0 以降のバージョンで、ゲスト・システムとして稼働するカーネルのバージョン 2.6 に対してのみサポートされます。

クリティカル・リソースの設定

ProtectionMode 永続属性を使用して、リソースがクリティカルであるかどうかを指定 します。リソースがクリティカルである場合、構成 RM (IBM.ConfigRM) が、そのリソースを要求に応じて開始できるかどうかを判断します。属性は、 整数値 0 (非クリティカル) または 1 (クリティカル) です。デフォルトでは、IBM.Application リソースは非クリティカル、IBM.ServiceIP リソースはクリティカルです。リソースがクリティカルに設定されると、モニターが即時に開始されます。

ProtectionMode 属性の値をリストするには、RSCT コマンド **lsrsrc** を発行します。

```
lsrsrc IBM.Application Name NodeNameList ProtectionMode
```

リソースをクリティカルとして定義するには、RSCT コマンド **chrsrc** を発行し、ProtectionMode を 1 に設定します。

```
chrsrc -s "Name='apache1'" IBM.Application ProtectionMode=1
```

リソースを非クリティカルとして定義するには、ProtectionMode を 0 に設定します。

```
chrsrc -s "Name='apache1'" IBM.Application ProtectionMode=0
```

リソース・クラス IBM.Application のクリティカル・リソースがノードで現在アクティブであるかどうかを検証するには、以下のコマンドを発行します。

```
lsrsrc IBM.Application Name NodeNameList OpState ProtectionMode
```

以下の出力結果が得られます。

```
resource 1:
        Name           = "apache1"
        NodeNameList   = {"node1","node2"}
        OpState        = 1
        ProtectionMode = 1
resource 2:
        Name           = "apache1"
        NodeNameList   = {"node1"}
        OpState        = 2
        ProtectionMode = 1
resource 3:
        Name           = "apache1"
        NodeNameList   = {"node2"}
        OpState        = 1
        ProtectionMode = 1
```

クリティカル・リソース apache1 は node2 でアクティブです。

クリティカル・リソースがノードで現在アクティブであるかどうかを検証するには、以下のコマンドを発行します。

```
lsrsrc IBM.PeerNode Name CritRsrcActive
```

出力結果は以下のようになります。

```
Resource Persistent and Dynamic Attributes for IBM.PeerNode
resource 1:
        Name           = "node1"
        CritRsrcActive = 0
resource 2:
        Name           = "node2"
        CritRsrcActive = 1
```

クリティカル・リソースは node2 でアクティブです。

クォーラム情報の取得

現在のクォーラム状態を取得するには、IBM.RecoveryRM デーモンの **lssrc** コマンドを使用します。

```
node02:~/build # lssrc -ls IBM.RecoveryRM
```

以下の出力結果が得られます。

```
Daemon State:
   My Node Name         : node02
   Master Node Name     : node01 (node number = 1)
   Our CVN              : 61035379498
   Total Node Count     : 2
   Joined Member Count  : 2
   Config Quorum Count  : 2
   Startup Quorum Count : 1
   Operational Quorum State : HAS_QUORUM
   In Config Quorum     : TRUE
   In Config State      : TRUE
```

さまざまな属性の意味は以下の とおりです。

Total Node Count
クラスター内で定義されているノードの数です。

Joined Member Count
クラスター内で実行されている IBM.RecoveryRM デーモンの数です。これは、(サブ) クラスター内のアクティブ・ノードの数と等しくなります。

Config Quorum Count
System Automation for Multiplatforms のコマンドを使用して構成変更を実行するためにアクティブでなければならない IBM.RecoveryRM デーモンの数です。

Startup Quorum Count
System Automation for Multiplatforms の自動化エンジンがアクティブになる前にアクティブである 必要がある IBM.RecoveryRM デーモンの数です。

操作可能クォーラムの状況
サブクラスター内のノード (1 つ以上) でクリティカル・リソースが稼働している 場合に、このサブクラスターが存続できるか、即時に解除されなければならないかを (サブ) クラスター全体に 示します。この操作クォーラムの 状態は、PeerDomain クラスの動的属性 OpQuorumState によって提供 されます。OpQuorumState の値は以下のいずれかです。

0 - HAS_QUORUM
System Automation for Multiplatforms はリソースを開始できます。

1 - PENDING_QUORUM
まだ解決されていないタイ状態が発生していることを示します。System Automation for Multiplatforms は リソースを開始しません。

2 - NO_QUORUM
System Automation for Multiplatforms はリソースを開始できません。

In Config Quorum
System Automation for Multiplatforms コマンドによる構成変更を受け入れるために、IBM.RecoveryRM デーモンをホスティングする十分な数のノードが アクティブであるかどうかを示します。クラスター

内の「結合された」IBM.RecoveryRM デーモン・グループ・メンバーの 合計数が Config Quorum Count と等しいか、またはこれより多い場合は、「TRUE」が表示されます。

In Config State
起動時に、マスター IBM.RecoveryRM デーモンがシステム・レジストリーの内容の検証を 完了したかどうかを示します。この状態が FALSE の場合は、すべての System Automation for Multiplatforms コマンドが 拒否されます。

OpQuorumState をリストするには、以下のように入力します。

```
lsrsrc IBM.PeerDomain Name OpQuorumState
```

以下の出力結果が得られます。

```
Resource Persistent and Dynamic Attributes for:IBM.PeerDomain
resource 1:
        Name = "myCluster"
        OpQuorumState = 0
```

ポリシー・ベースの自動化

System Automation for Multiplatforms では、自動化ポリシーを作成および処理することにより、高可用性システムを構成できます。自動化ポリシーは、さまざまなシステム・コンポーネント間の関係を定義します。

System Automation for Multiplatforms では、わずかな変更のみで自動化ポリシーを既存のアプリケーションに適用できます。 関係が設定されると、System Automation for Multiplatforms は、指定されたノード上のアプリケーションを 構成されたとおりに確実に管理します。これにより、インプリメンテーション時間およびアプリケーションの複雑なコーディングの必要性が削減されます。既存のスクリプトを変更せずに、リソースとシステムをご使用の環境に簡単に追加できます。

次の IBM Tivoli Open Process Automation Library から、サンプル自動化ポリシーをダウンロードできます。

```
http://catalog.lotus.com/wps/portal/topal
```

サンプルにアクセスするには、「**Browse by**」ウィンドウで 「**Tivoli 製品**」をクリックし、次に、リストから「**Tivoli System Automation**」を選択します。

リソース

リソースを作成および使用することによって、環境内のハードウェアとソフトウェアのコンポーネントを管理できます。

以下のテーブル・リストに、管理対象リソースに対して使用する System Automation for Multiplatforms コマンドを示します。

表 *4.* 管理対象リソースとともに使用する *System Automation for Multiplatforms* コマンド

コマンド	説明
addrgmbr	1 つ以上のリソースをリソース・グループに追加する
chrgmbr	リソース・グループ内の管理対象リソースの永続属性値を変更する
lsrg	既に定義されているリソース・グループまたはそのメンバー・リソースをリストする
rmrgmbr	リソース・グループから 1 つ以上のリソースを除去する

リソース・コマンドについての詳しい説明は、*Tivoli System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

リソースとは

リソースとは、89 ページの『Resource』で説明されている任意の方法を使用して RMC に対して定義された、1 つのハードウェアまたはソフトウェアです。

1 ページの『高信頼性スケーラブル・クラスター・ テクノロジー (RSCT)』で説明しているように、System Automation for Multiplatforms では、その基礎として RMC の機能が 使用されます。このため、リソースは RMC リソース と 呼ばれることがあります。

リソース (アダプター、プログラム、サービス、ディスクなど) は、リソース・マネージャー (RM) によって制御されます。

リソース・クラスとは

リソース・クラスは、同じタイプのリソースのセットです。例えば、リソースは 特定のファイルシステムや特定のホスト・マシンなどですが、リソース・クラスはファイルシステムのセット、またはホスト・マシンのセットです。リソース・クラスは、リソース・クラスのインスタンスが保持できる 共通の特性を定義します (例えば、すべてのファイルシステムは、識別特性 (名前など) および可変特性 (マウントされているかどうか、など) を 保持します)。また、リソース・クラスの個々のリソース・インスタンスは、その特定の特性の値を定義します (例えば、このファイルシステムは「/var」という名前である、これは現在マウントされている ファイルシステムである、など)。

リソース属性とは

リソース属性とは、リソースの一部の特性を意味します。リソースが ホスト・マシンを表す場合、その属性は、ホスト名、物理メモリーのサイズ、マシン・タイプなどを示します。

リソース属性には、永続属性と動的属性の 2 種類があります。

永続属性
前述のホスト・マシンの属性 (ホスト名、物理メモリーのサイズ、およびマシン・タイプ) は、永続属性の例です。 永続属性は、リソースの永続的な特性を示します。 ホスト名を変更したり、その物理メモリーのサイズを増やしたりすることはできますが、これらの特性は、一般的に継続的で不変です。

動的属性
動的属性はリソースの変化する特性を示します。例えば、ホスト・リソースの動的属性は、実行キューで待機中のプロセスの平均数、プロセッサーのアイドル時間、現在ログオン中のユーザー数などを 示します。

管理対象リソース

リソースがリソース・グループに挿入されるとすぐに、そのリソースは管理対象リソースになります。

これは、System Automation for Multiplatforms **addrgmbr** コマンドを使用して行われます (*IBM Tivoli System Automation for Multiplatforms* リファレンス を参照)。この時点以降、リソースは System Automation for Multiplatforms のコマンドおよびプログラムを使用して 管理できます。

管理対象リソースは、System Automation for Multiplatforms のリソース・クラス `IBM.ManagedResource` で提供されます。

リソースによって使用される属性

リソースは、以下の属性を保持できます。

NodeNameList
リソースが、どのノードで稼働可能であるかを示します。NodeNameList は RSCT リソースの永続属性です。

SelectFromPolicy
リソースが使用可能なノードのリストを含みます。SelectFromPolicy は管理対象リソースの永続属性です。

ResourceType
リソースが、固定リソース、浮動リソース、または並行リソースの いずれであるのかを示します。固定リソースは、単一ノード上でのみ実行できます。浮動リソースは、複数ノード上で 実行できますが、同時に実行できるのは単一ノードでのみです。並行リソースは、 複数ノード上で実行でき、同時に複数ノードで実行できます。ResourceType は RSCT リソースの永続属性です。

注 : 並行リソースは、リソース・クラス IBM.Application および IBM.Test のリソースでのみ 使用可能です。

OpState
リソースまたはリソース・グループの操作状態を示します。OpState は RSCT リソースの動的属性です。

NodeNameList 属性

NodeNameList 永続属性は、リソースが稼働できるノードのセットを表します。

System Automation for Multiplatforms によってリソースが開始されるノードは、リソースの NodeNameList 内のノードの順序によって制御されます。デフォルトでは、各浮動リソースが そのノード・リストの最初のノードに置かれます。そのノードで、他のリソースに対する そのすべての関係を考慮して、そのリソースを開始させることができます。この動作は、後述の SelectFromPolicy 属性を使用して変更できます。連結する 必要のある浮動リソースは、独立した浮動リソース (他のリソースに対する関係を 保持していないリソース) および非連結浮動リソースの 前に置かれます。

注 : SelectFromPolicy 属性は、取得済みの IBM.AgFileSystem リソースについては無視されます。

各並行リソースは、すべての適格ノードで開始されます。 並行リソースには、位置制約に関する複数の特別ルールが 適用されます。これらのルールについての詳細は、66 ページの『並行リソースの自動化動作』を 参照してください。

連結リソースが異なる複数のノード・リストにある場合、リソースは、大多数のリソースによって選択されるノードに置かれます。タイ状態 では、1 つのノードがランダムに選択されます。

SelectFromPolicy 属性

SelectFromPolicy 属性は、いずれのノードでリソースを開始するのが望ましいのかを定義します。

注 : SelectFromPolicy 属性は、取得済みの IBM.AgFileSystem リソースについては無視されます。

以下に示す 3 つの設定が使用可能です。

Any
System Automation for Multiplatforms はノードをランダムに選出します。

Ordered
System Automation for Multiplatforms は、次に選択可能なノードを決定する際に、必ず NodeNameList の最初から開始します。

Ordered, Failback
「Ordered,Failback」は、「Ordered」と同等ですが、ホーム・ノードへのフェイルバック機能が実装されます。ホーム・ノードへの フェイルバック機能について詳しくは、62 ページの『ホーム・ノードへのフェイルバック機能』を 参照してください。

ResourceType 属性

ResourceType 属性は、リソース・マネージャーによって定義されるか、リソースの作成時に定義されます。ResourceType 永続属性は、リソースが以下のいずれであるかを指定します。

- シリアル固定 (その NodeNameList 属性に単一の ノード・エントリーが含まれる)。

  固定リソース

  固定リソースは、クラスター内に単一のインスタンスしか持たない リソースです。そのリソースは、単一のノードで定義され、そのノードで 稼働します。これは、プロセス、マウント・ポイント、またはネットワーク・アダプター などの 1 つのエンティティーを表します。

クラスター内の複数のノードで同時に稼働するアプリケーションを定義する場合は、ノードごとに 1 つの固定リソースを定義し、かつ固定リソースごとに、同じ属性 (例えば、名前、start/stop/monitor コマンド)、しかし異なるノード名を指定する必要があります。

- シリアル浮動 (その NodeNameList 属性に 1 つ以上のエントリーが含まれる)。考えられる使用のために複数のノードを定義できますが、アクティブにすることができるのは常にリソースの 1 つのインスタンスのみです。例えば、1 つのマシンから別のマシンに移動できる IP アドレスは浮動リソースです。IP アドレスは、任意の時点で複数のマシンが使用できますが、一度に 1 つのマシンのみこれを使用できます。

浮動リソース

浮動リソースは、クラスター内の複数のノードで稼働可能な リソースです。浮動リソースは、集合リソースを 1 つ、集合リソースに属するノードごとに構成要素リソースを 1 つという方法で RMC 内に表します。

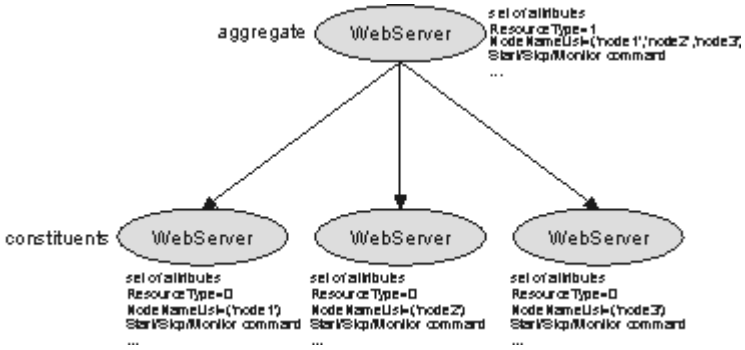


図 7. 浮動リソースの例

集合リソースの ResourceType 属性値は 1 です。リソースが稼働できるノードのセットは、集合リソースの NodeNameList 属性で定義されます。このリソースのその他の属性は、リソース・マネージャー およびそのクラス定義によって定義されます。

浮動リソースを 作成する場合は、集合リソースを作成します。このタイプのリソースを 担当するリソース・マネージャーは、そのリソースが稼働する予定の各ノード上に 構成要素リソースを作成します。構成要素リソースは、独自の属性値を持ちます。構成要素リソースの ResourceType 属性は 0 (固定リソース) で、NodeNameList には 1 つのノードしか含まれません。作成時は、他の属性の値は集合リソースと同じです。

浮動リソースの属性を変更すると、以下のことが発生します。

- 集合リソースの NodeNameList を変更すると、構成要素リソースが削除または作成されます。
- 集合リソースの属性を変更すると、すべての構成要素リソースの 同じ属性が変更されます。
- 構成要素リソースの属性を変更すると、その影響を受けるのは その構成要素リソースのみで、その他の構成要素リソースまたは集合リソースには 影響はありません。

浮動リソースは、複数のノードで稼働可能なアプリケーションやサービス IP アドレス などの自動化可能なエンティティーを表します。

注 : 浮動リソースは、System Automation Application Manager オペレーション・コンソールでは移動グループというラベルが付けられます。

- シリアル並行 (その NodeNameList 属性に *1* つ以上の エントリーが含まれる)。

並行リソース

並行リソースは、浮動リソースとよく似ています。 浮動リソースと異なり、並行リソースでは、常に、すべての構成要素を同時に 開始しようとします。さらに、多数の関係では、並行リソースが関係のソースまたは ターゲットである場合、実装される動作が異なっています。

注 : 並行リソースは、System Automation Application Manager オペレーション・コンソールでは並行グループというラベルが付けられます。

OpState 属性

RMC では、OpState 動的属性を使用して、リソースの操作状態を 指定します。これはリソース・グループに追加されたリソースの場合は必須です。

OpState 属性に指定可能な値を以下に示します。

オフライン
リソースは開始されていません。

オンラインの保留中
リソースは開始されていますが、まだ作業を行う準備ができていません。

オンライン
リソースは作業を行う準備ができています。

オフラインの保留中
リソースは停止プロセス中です。

操作状態の中には、問題を示すものがあります。

オフラインに失敗
リソースは中断され、使用できません。修正時にリソースを リセットする必要があります。

オンライン中
リソースが開始されていましたが、期待された時間間隔内に作業を行う 準備ができず、オフラインにすることができません。または、リソースがオンラインになっていましたが、オフライン要求によってそれをオフラインにすることができませんでした。

Unknown
System Automation for Multiplatforms は、リソースを管理する RMC から信頼できる状態情報を 入手できません。

不適格
リソースはオフラインであり、自動化オーダーを一時的に受け入れることができません。

注 :「オフラインに失敗」状態のリソースは、リセットする 必要があります。これを実行するには、RMC コマンド **resetrsrc** を使用します。詳しくは、「Reliable Scalable Cluster Technology」を参照してください。

リソースのノードが「オフライン」のときは、その時点で操作状態が「不明」となっていても、リソースは「オフラインに失敗」であると見なされます。System Automation for Multiplatforms には リソース・ノードに関する独立した状態データがあるため、これを実行できます。

NominalState はリソース・グループにのみ設定できます。

NominalState 属性はリソース・グループにのみ設定できます。リソース・グループの NominalState の値 (「オンライン」または「オフライン」) によって、そのメンバーのすべてが自動的にオンラインになるかオフラインになるかが決まります。個別のリソースは NominalState 属性を持ちません。この事実は、すべてのリソースがリソース・グループのメンバーでなければならない 1 つの理由になっています。

リソース・グループ

以下のトピックで、リソース・グループの使用法について説明します。

- 18 ページの『リソース・グループとは』
- 20 ページの『リソース・グループによって使用される属性』
- 132 ページの『リソース・グループの作成 』
- 134 ページの『リソース・グループの属性の変更 』
- 135 ページの『リソース・グループの除去』
- 133 ページの『リソース・グループとそのメンバーのリスト』
- 132 ページの『リソース・グループへのメンバー・リソースの追加』
- 134 ページの『リソース・グループからのメンバー・リソースの除去』
- 134 ページの『リソース・グループ・メンバーの属性の変更』

関連セクション:

- 59 ページの『可用性に関連するイベント』

18 ページの表 5 は、リソース・グループで使用する System Automation for Multiplatforms コマンドのリストです。詳細については、「*IBM Tivoli System Automation for Multiplatforms* リファレンス」のコマンドの説明を参照してください。

表 *5.* リソース・グループとともに使用する *System Automation for Multiplatforms* コマンド

コマンド	説明
mkrg	リソース・グループを作成する
rmrg	リソース・グループを除去する
chrg	リソース・グループの永続属性を変更する (リソース・グループの 開始および停止を含む)
lsrg	1 つ以上のリソース・グループをリストする
addrgmbr	リソース・グループにメンバー・リソースを追加する
rmrgmbr	リソース・グループからメンバー・リソースを除去する
chrgmbr	リソース・グループのメンバー・リソースの属性を 変更する
rgreq	リソース・グループを開始、停止、キャンセル、または移動する
rgmbrreq	管理対象リソースの開始、停止、またはキャンセルを要求する
lsrgreq	リソース・グループまたは管理対象リソースに対して適用される未解決の要求をリストする

リソース・グループとは

リソース・グループは、System Automation for Multiplatforms における中心単位です。リソース・グループは、1 つの論理インスタンスとして処理できるリソースの集合の論理コンテナーです。

- リソース・グループを使用して、それらのすべてのメンバーを集合的に制御できます。例えば、リソース・グループの NominalState を「オンライン」に 設定すると、すべてのメンバーが開始され、オンライン状態が維持されます。NominalState を「オフライン」に設定すると、すべてのメンバーが停止され、オフライン状態が維持されます。
- それらの OpState をモニターできます。これにより、個々のリソース・グループ・メンバーの OpState を統合できます。

リソース・グループのメンバーにすることができるのは、以下のタイプのリソースです。

- シリアル固定
- シリアル浮動
- シリアル並行
- リソース・グループそのもの。これは、ネストされたグループを定義できることを 意味します。

固定リソースを含むリソース・グループの例として、シリアル固定リソースを 含むリソース・グループ RG_Fix を挙げます。リソースは、node1 でのみ稼働可能な Web サーバー FixWebServer、および node2 にあるデータベース・リソース FixDB2 です。

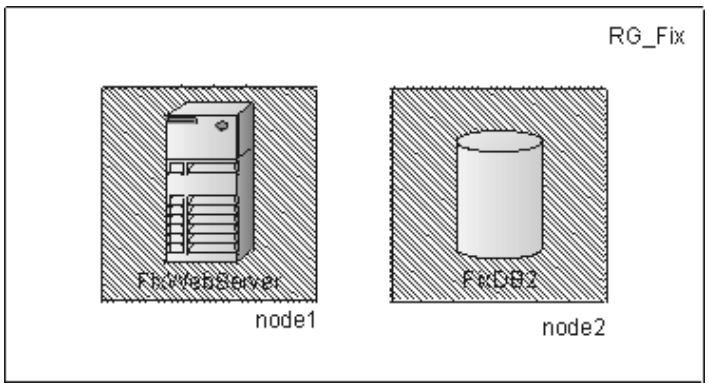


図 *8.* 固定リソースを含むリソース・グループの例

FixWebServer および FixDB2 の両方のリソースを開始するために、RG_Fix の NominalState を「オンライン」に 設定します。この例は、System Automation for Multiplatforms が、クラスター内の異なるノードに分散されたリソース・グループ・メンバーを処理できることも示しています。

浮動リソース・グループ・メンバーの例は以下のとおりです。Web サーバー apache1 は、node1、node2、または node3 のいずれかで稼働可能です。リソース・グループ RG_WebApp は、Web サーバーが 3 つのノードのいずれかで開始できることを除いて、非常によく似ています。

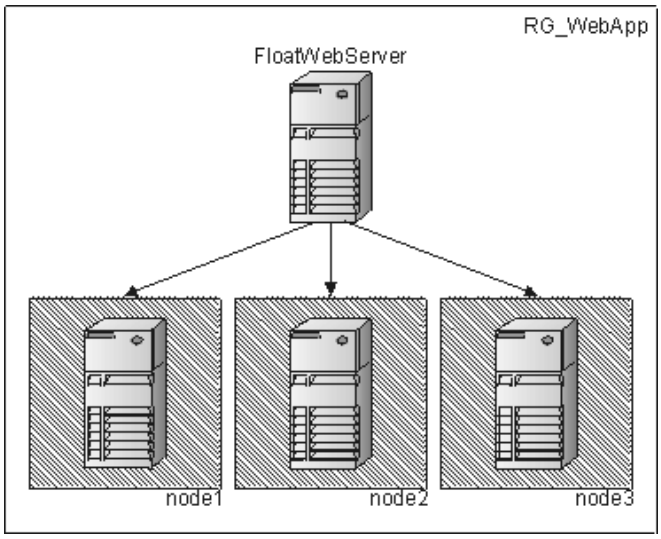


図 *9.* 異なるリソース・タイプのメンバーが混在しているリソース・グループの例

この例は、リソース・グループに異なるリソース・タイプのメンバーが混在可能であることを 示しています。

リソース・グループの概念は非常に強力で、リソース・グループを他のリソース・グループの メンバーとして定義できます。メンバーとして 固定リソースであるリソース A を持つリソース・グループ RG_A と、前の例の リソース・グループ RG_WebApp の例が挙げられます。ネストされたリソース・グループにより、複雑な環境を複数の層に構造化できます。ネスト・レベルは 50 です。

リソース・グループの機能のもう 1 つの柔軟性は、開始または停止関係および位置制約関係など すべての種類の関係を、リソース・グループを使用してソース・リソースまたは ターゲット・リソースとして定義できる点です。さらに、リソース・グループ・メンバーをソース・リソースまたはターゲット・リソースとしてそのような関係の 一部にすることができます。

リソース・グループは、System Automation for Multiplatforms のリソース・クラス `IBM.ResourceGroup` で定義されます。

リソース・グループの使用に関するルール

以下に、リソース・グループを使用するためのルールを示します。

1. リソース・グループには同値を含めることができません。逆もまた 同様です。
2. リソースは 1 つのグループにのみ属すことができます。
3. 1 つのメンバーはグループおよび同値の両方に存在できません。
4. リソース・グループのネスト・レベルは 50 に制限されています。
5. グループまたは関係によってリンク可能なリソース数は 100 に制限 されます。

リソース・グループによって使用される属性

リソース・グループは、ユーザーが定義できる以下の永続 RSCT 属性を 提供します。

ActivePeerDomain
グループが定義されている対象ピア・ドメインの名前。 これはユーザーが設定することはできません。

AllowedNode
リソース・グループ・メンバーの開始が許可されるノードを 制限します。

説明
リソース・グループに関する説明的なテキストを含みます。

ExcludedList
グループのノード・リストから一時的に除外するノードの リストを定義します。

InfoLink
オペレーターがリソースの追加情報を参照できる HTML ページの URL を指定します。

MemberLocation
リソース・グループのすべてのメンバーを連結する (Collocated) 必要があるかどうかを 定義します。「Collocated」は、すべてのメンバーが同じノード上で稼働しなければならないことを意味します。「None」は、メンバーが相互に独立しており、それらが定義されているノード上で 任意に稼働できることを意味します。

注 : 並行リソースを含むことができるようにするには、グループの memberLocation 設定が「None」である必要があります。

名前
リソース・グループの固有の名前を定義します。

NominalState
リソース・グループの本来あるべき状態。System Automation for Multiplatforms は、リソース・グループを この状態にし、保持しようとします。

Owner
リソース・グループの所有者に関する情報 (所有者の名前と電話番号など) を提供します。

Priority
競合状態におけるリソース・グループの重要度を定義します。

Requests
要求永続性を使用可能に設定して、このリソースに対して実行依頼された要求を定義します。

Subscription
エンドツーエンド管理に関するサブスクリプション情報を含みます。

注 : このセクションで説明する永続属性は、それらを含む リソース・グループが「**オフライン**」の場合にのみ 変更できます。例外は、属性 NominalState および情報属性 Description、InfoLink、そして Owner です。リソース・グループが 「**Online**」のときも変更できます。

リソース・グループは、以下の永続属性を提供します。

SelectFromPolicy
このフィールドは、集合リソースの構成要素からの選択時に使用されるポリシーを定義します。現在サポートされている値は以下のとおりです。

- 「Any」または 0: System Automation for Multiplatforms は構成要素をランダムに選択します。
- 「Ordered」または 1: System Automation for Multiplatforms では、集合体の NodeNameList によって推測される順序に基づいて構成要素を選択します。

- 「Ordered,Failback」または 2: フェイルバックをアクティブ化。System Automation for Multiplatforms では、 集合体の NodeNameList の最初の適格エントリーであるホーム・ノードに、このリソースを 戻そうとします。

Instances
将来の利用のために予約されています。

Ordering
将来の利用のために予約されています。

リソース・グループは、以下の動的属性を提供します。

OpState
管理対象リソースのコレクションの 集約的な操作状態を指定します。

WinSource
リソースの OpState を処理するイニシエーターを表示します。 表示される値を次に示します。

Nominal

Location
リソースが現在バインドされているノードを表示します。

LockState
ロック要求が投入された後であり、リソース・グループが現在ロック状態であることを 示します。グループがロックされている間は、グループのメンバーの自動化は すべてサスペンドされます。

TopGroup
リソース・グループの最上位リソース・グループ名を示します。

AutomationDetails
リソース・グループの System Automation for Multiplatforms 内部状態を示します。

MoveStatus
rgreq コマンドにより開始されたリソース・グループの移動の進行状況を示します。

ConfigValidity
ポリシーが無効になったかどうかを表示します。

AllowedNode 属性

AllowedNode 属性を使用して、リソース・グループのメンバーの稼働が制限されるクラスター内のノードのセットを定義します。

以下のパラメーターから選択できます。

すべて
デフォルト値です。これは、リソース・グループによって制限が作成されないことを 意味します。クラスター内のすべてのノードで稼働可能です。

One node (1 つのノード)
すべてのリソース・グループ・メンバーを稼働させる必要のある特定のノードを 定義します。指定されたノードが後でクラスターから除去されると、AllowedNode にはデフォルト値 All が設定されます。

Equivalency of nodes (ノードの同値)
リソース・グループ・メンバーの稼働を制限するノードのセットを 含みます。静的同値のみ許可されます。52 ページの『同値』 も参照してください。

AllowedNode パラメーターのノード制限の特徴

特定のケースにおいては、ノードのセット上で稼働させるリソース・グループのメンバーを制限する必要がある場合があります。例えば、浮動リソースが定義されている場合は、NodeNameList が指定されている必要があり、これは一般的に System Automation for Multiplatforms とは別個の使用方法です。浮動リソースの NodeNameList は、浮動リソースを所有する リソース・マネージャー (例えば GblResRM) が使用します。NodeNameList には、リソース・マネージャーに対し、浮動リソースの構成要素が潜在的にどのノードで稼働できるかを 定義します。その一方で、AllowedNode 属性は、浮動リソースであるさまざまなリソース・グループ・メンバー を持つリソース・グループ・パラメーターに属します。ここでは、AllowedNode

属性により、リソース・グループ・メンバーが稼働可能なノードを 制限できます。これは、各タイプのグループ・メンバーについて、以下を意味します。

- **固定リソース**の場合、固定リソースが存在する 1 つのノードのみが含まれる NodeNameList が、AllowedNode パラメーターの一部になって いなければなりません。
- **浮動並行リソース**の場合、NodeNameList および AllowedNode パラメーター の論理積により、System Automation for Multiplatforms によって浮動リソースが開始され、制御されるノードの セットが定義されます。
- **リソース・グループ**の場合、内部グループおよび外部グループの AllowedNode パラメーターの論理積により、内部グループの許可されたノードの結果リストが 導き出されます。結果リストでは、内部リソース・グループのメンバーの 稼働が許可されるノードが定義されます。

以下の例で、ノード制限における AllowedNode パラメーターの動作を説明します。NodeNameList = {node1} のメンバー FixA、NodeNameList = {node1, node2, node3} の浮動メンバー FloatB、および AllowedNode = {node2, node3, node4} のリソース・グループ RG_B を持つ外部リソース・グループ RG_A があります。RG_A の AllowedNode リストでは、{node1, node2, node 4} が定義されています。RG_B には、NodeNameList が {node1, node2, node3, node4} の FloatC、および NodeNameList が {node3, node4} の FloatD が含まれています。

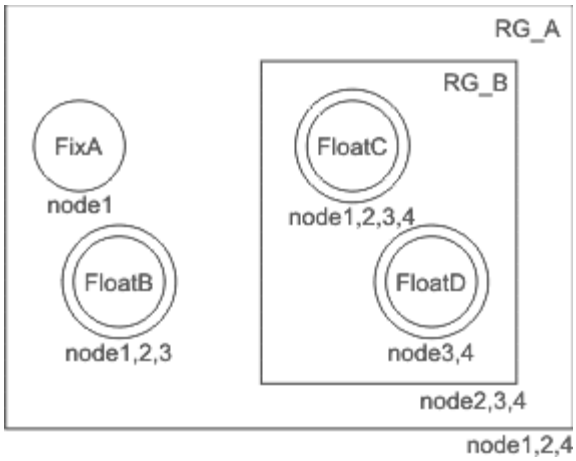


図 *10.* ノード制限における *AllowedNode* パラメーターの動作の例

このシナリオの結果は、以下のとおりです。

- FixA は、System Automation for Multiplatforms によって node1 上でのみ始動可能です。
- FloatB は、System Automation for Multiplatforms によって node1 および node2 上でのみ始動可能です。
- RG_B のメンバーは、node2 および node4 上での稼働に限定されます。
- FloatC は、System Automation for Multiplatforms によって node2 および node4 上でのみ始動可能です。
- FloatD は、System Automation for Multiplatforms によって node4 上でのみ始動可能です。

内部グループの AllowedNode リストは、その外部グループにノード制限がない場合、独自の AllowedNode パラメーターとの論理積、およびその外部グループの AllowedNode リスト との論理積のために影響を受けません。

MemberLocation 属性

`MemberLocation` 永続属性は、リソース・グループ内のリソース間のデフォルトの位置を指定するために使用します。有効値は以下のいずれかです。

Collocated (連結) (デフォルト)
「Collocated」は、すべてのメンバーが同じノード上で稼働しなければならないことを意味します。`Collocated` を指定した場合は、リソース・グループ内のリソース間に `Affinity`、`AntiAffinity`、および `AntiCollocated` 管理対象関係を適用して、メンバー・リソースがどのノードに配置されるべきかを指定することはできません。関係についての詳細は、29 ページの『管理対象関係』を参照してください。

memberLocation 設定に「Collocated」が指定されているグループのメンバーとして並行リソースを定義することはできません。並行リソースをグループに追加するには、グループの MemberLocation 設定が「None」でなければなりません。

なし
None は、リソース・グループにおいてそのメンバーの位置に関する制限がないことが 暗黙指定されることを意味します。

MemberLocation 属性は、リソース・グループのすべてのメンバー・リソースに適用されます。

リソース・グループがネストされる場合、外部リソース・グループの MemberLocation 属性の値により、内部グループ (1 つ以上) の指定が許可されなければなりません。したがって、リソース・グループの MemberLocation 属性が Collocated の場合、連結された複数のリソース・グループのみがメンバーとして許可されます。

Name 属性
リソース・グループの Name (名前) は、クラスター内で固有でなければなりません。

以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

NominalState 属性

NominalState 永続属性は、リソース・グループを制御するために使用します。リソース・グループの NominalState をオンラインに設定することにより、そのすべての メンバーが開始され、オンライン状態が保持されます。

NominalState 属性は、オンラインまたはオフラインのいずれかにすることができます。NominalState をオフラインにすると、すべてのリソース・グループ・メンバーが停止され、オフライン状態が保持されます。以下の例外があります。

1. リソース・グループがネストされている場合、外部グループの NominalState のオンラインにより、内部グループの NominalState のオフラインが否定され、内部グループが開始されます。このような内部グループを個別に停止することはできません。
2. リソース・グループがネストされている場合、外部グループの NominalState のオフラインにより、内部グループの NominalState のオンラインは否定されません。内部グループの NominalState のオンラインにより、このグループおよびこの内部グループに 含まれるグループが開始されます。外部グループの OpState が変更されますが、その他のグループ・メンバーには影響はありません。
3. グループ間の依存関係により、個々のグループ・メンバーを強制的に開始させる場合があります。31 ページの『開始または停止動作の関係』を参照してください。

リソース・グループの NominalState のデフォルト値は、オフラインです。

この属性はいつでも変更可能です。

注 : この属性は、**chrg -o** コマンドを使用して変更できます。詳しくは、*System Automation for Multiplatforms* リファレンス・ガイドの **chrg** コマンドを参照してください。この属性の数値は以下のいずれかです。

1
オフライン

0
オンライン

Priority 属性

Priority 永続属性は、このリソース・グループの他のリソース・グループとの関係における 相対的な優先順位を指定するために使用します。

Priority 属性は、リソース・グループが開始され、他の開始済みリソースまたはオンライン・リソースに対して競合する管理対象関係 (29 ページの『管理対象関係』で説明しています) がある場合に、競合を解決するために使用されます。こうした競合は、リソース・グループまたはそのリソース・グループに含まれる管理対象リソースと、開始済みまたは実行中のリソースとの間に発生する可能性があります。

実際の優先順位が同じリソース・グループでは、非強制メンバーの数によって順番が決まります。非強制メンバーの数が少ないグループが最初に犠牲になります。実際の優先順位が同じであり、非強制メンバーの数が同じグループの場合、このアルゴリズムでは、バインディングの問題から除去するグループが偶然にまかせて決定されます。これにより、制約充足問題に対する過剰な規定 が緩和されます。このように除去されることにより、グループおよびそのメンバー は本来あるべき状態を問わず停止される (オフラインにとどまる) ため、この除去 を sacrificing (犠牲) と呼びます。

例えば、1 つのノードがオンラインになっているクラスターと、互いに AntiCollocated 関係にある 2 つのリソース・グループがあるとします。これは、リソース・グループを同一ノード上で同時に開始してはいけないことを意味します。System Automation for Multiplatforms は、リソース・グループの Priority 属性の値を使用して、オンラインにするべきリソース・グループと、AntiCollocated 関係があるためオンラインにできないリソース・グループを判別します。

競合関係があるため、より低い優先順位のアクティブ・リソース・グループが、より高い優先順位のリソース・グループの活動化を妨げる場合、より高い優先順位を持つリソース・グループの活動化を進めるために、より低い優先順位のリソース・グループが停止されます。

リソース・グループがネストされている場合、外部リソース・グループは内部リソース・グループ以上の優先順位を持つ必要があります。

Priority 属性のデフォルト値は 0 で、これは最も低い 値です。最大値は 200 です。

リソース・グループの実際の優先順位は、優先順位属性の値、およびグループの現在の監視状態によって決定される調整値から計算されます (26 ページの『AutomationDetails 属性』を参照)。 オンラインのグループの調整値は、開始リソースの放棄を困難にする +10 であり、オフラインに失敗するグループの場合は -10 です。 実際の優先順位は、-10 から +210 の値と想定できます。 優先順位の意味について詳しくは、56 ページの『ノード位置の割り当て』を参照してください。

ヒント:

- System Automation for Multiplatforms は、管理対象リソースの Mandatory 属性 (27 ページの『リソース・グループ・メンバーに対して使用される属性』で説明されています) も使用して、競合状態のときに開始されるリソースを判別します。ネストされたリソース・グループの場合、非必須メンバーのリソース・グループは、必須メンバーよりも優先順位が低くなければなりません。この条件が満たされていないと、必須メンバーが廃棄されることがあります。
- グループに「アンバインド」または「オフライン」状態のメンバーがある場合、+10 の調整値はありません。

ExcludedList 属性

ExcludedList 属性を使用して、グループのノード・リストから 1 つのノードまたは複数のノードを一時的に除外します。ノードを除外する場合、除外されるノード上にある リソースが自動的に強制的にオフラインにされることはありません。**rgreq** コマンドを使用することにより、移動を起動する必要があります。

これは、除外リストにノードを置くことによって、System Automation for Multiplatforms がそのノードをリソースをホスティングする 潜在的な候補と認識しなくなることを意味します。この属性は、後の時点における除外 (**samctrl** コマンドを使用) への準備として、ノードから徐々に、混乱を生じさせることなくリソースを移動するために使用できます。

以下のルールが適用されます。

1. 固定リソース・グループ・メンバーの場合、ノードの除外は、それ以降リソースを開始できないことを意味します。

2. 浮動リソース・グループ・メンバーの場合、ノードの除外は、それ以降そのノード上における 構成要素を開始できないことを意味します。

ActivePeerDomain

この属性により、グループが定義されている RSCT ピア・ドメインの名前が 表示されます。

Description 属性

この属性には、リソース・グループに関する説明的なテキストを含むことができます。この 属性は情報を提供する目的にのみ使用でき、自動化の機能には影響しません。

InfoLink 属性

この属性は、オペレーターがリソースの追加情報を参照できる HTML ページの URL を指定するために使用できます。この 属性は情報を提供する目的にのみ使用でき、自動化の機能には影響しません。

所有者属性

この属性は、リソース・グループの所有者に関する情報 (所有者の名前と電話番号など) を提供します。この 属性は情報を提供する目的にのみ使用でき、自動化の機能には影響しません。

Subscription 属性

この属性は、エンドツーエンド管理からのサブスクリプション情報を含みます。

OpState 属性

System Automation for Multiplatforms では、OpState 動的属性を使用して、(管理対象リソースの) コレクションの 集約的な操作状態を明示します。これは、リソース・グループの メンバー・リソースの個々の操作状態から決定されます。

OpState 属性に指定可能な値を以下に示します。

表 *6. OpState* 属性の値および状況

状況	値	説明
不明	0	
オンライン	1	すべての必須メンバー・リソースがオンラインであることを 明示します。停止状態にある非必須メンバーであるリソースは無視されます。
オフライン	2	すべてのメンバー・リソースがオフラインであることを 明示します。
オフラインに失敗	3	FailedOffline リソース・グループに含まれる 1 つ以上のメンバー・リソースが 「FailedOffline」であることを明示します。この場合、リソース・グループに含まれる すべてのリソースが「オフライン」に設定されます。 リソース・グループは、リソース・グループ・メンバーが配置できなかった場合、バインド・プログラムの実行結果として「オフラインに失敗」状態になることがあります。56 ページの『ノード位置の割り当て』 も参照してください。この場合、自動化詳細は、Sacrificed の BindingState を示します。26 ページの『AutomationDetails 属性』 も参照してください。
オンライン中	4	メンバー・リソースがオンライン中であることを 明示します。
オンラインの保留中	5	開始コマンドが実行されることを明示します (リソース・グループの NominalState 属性はオンラインに設定されます)。リソース・グループは、オンライン・アクションの処理を開始する必要があります。
オフラインの保留中	6	オフライン・アクションが開始されたことを 明示します。

表 6. *OpState* 属性の値および状況 (続き)		
状況	**値**	**説明**
Ineligible	8	リソース・グループはオフラインであり、自動化オーダーを一時的に処理できません。通常、これはノード隔離が進行中であるか失敗した場合に発生します。ノード隔離について詳しくは、RSCT または DB2 pureScale の資料を参照してください。

TopGroup 属性

この属性は、リソース・グループの最上位リソース・グループを示します。

System Automation for Multiplatforms では、リソース・グループを別のリソース・グループのメンバーにすることができ、これは、リソース・グループを ネストできることを意味します。TopGroup 属性は、現行グループの最上位 リソース・グループの名前を表示します。この属性は、*IBM Tivoli System Automation for Multiplatforms* リファレンス の説明ならびに以下に示すように、**lsrg** コマンドを使用して表示できます。

注 : lsrg -g コマンドを使用してリソース・グループを照会する場合は、 ユーザーが便利なように、TopGroup 属性とともに最上位グループの NominalState も出力に表示されます。出力結果は以下のようになります。

```
TopGroup             = apacherg
TopGroupNominalState = Offline
```

AutomationDetails 属性

この属性は、グループの System Automation for Multiplatforms の内部状態を示します。以下の状態 があります。

CompoundState
グループの依存関係を含むリソース・グループの全体的な状況。例:「Satisfactory」- リソース/グループが要求されたユーザー状況に 達しています。

DesiredState
ユーザーが要求したリソース・グループの状況。例:「online」- ユーザーはリソース・グループがオンラインであることを要求しました。

ObservedState
自動化の観点から見たリソース・グループの実際の状況。例:「online」- リソース・グループは現在オンラインです。

BindingState
リソース・グループが特定のシステムにバインドされているかどうかを示す状況。例:「bound」- リソース・グループは現在特定のシステムに バインドされています。

AutomationState
リソース・グループが現在自動化されているかどうかを示す状況。例:「Idle」- System Automation for Multiplatforms は現在リソース・グループを開始または停止しようとしていません。

ControlState
リソース・グループを自動化により制御できるかどうかを示す状況。例:「startable」- 現在このリソース・グループを開始できます。

HealthState
現在は使用されていません。将来のリリースのために予約されています。

前述の自動化の詳細を 表示するには、**lsrg** コマンドを -A d および -V オプションを 指定して使用します。「apacherg」という名前のリソース・グループの 自動化の詳細を表示するには、以下のコマンドを使用します。

```
lsrg -A d -V -g apacherg
```

MoveStatus 属性

この動的属性は、**rgreq** コマンドにより開始されたリソース・グループの移動の進行状況を示します。

移動状況を取得するには、**lsrg** コマンドを -A d および -V オプションを指定して使用します。

例:

「apacherg」というリソース・グループの移動状況を取得するには、次のコマンドを使用します。

```
lsrg -A d -V -g apacherg
```

表 *7. MoveStatus* の値

MoveStatus の値	**説明**
未定義	初期化のときの状態
サポートされない	移動はサポートされない
なし	移動のプロセスにない
準備中	移動操作が始まった
Stopping	オフラインのフェーズ 1: リソースが停止されている
取り消し	移動が取り消された (使用不可)
Starting	オンラインのフェーズ: リソースが再始動された
範囲外	移動は成功しなかった。オリジナル・ノードのグループを再始動
完了	移動が完了した

ConfigValidity 属性

ポリシーの設定後、ポリシーを無効にするいくつかの事項が発生する ことがあります。この属性は、無効の理由を示します。

例えば、連結リソース・グループのメンバーの唯一の共通ノードであった ノードがピア・ドメインから除去された場合、以後リソースを開始することはできません。このことが、ConfigValidity 属性によって示されます。

リソース・グループ・メンバーに対して使用される属性

各リソース・グループ・メンバーに加えて、ユーザーは以下の永続属性を定義する必要があります。

Mandatory
各リソース・グループ・メンバーに対して定義され、そのメンバーがグループに必須であるかどうかを指定します。または、メンバーを非必須にすることもできます。

MemberOf
リソースがメンバーであるリソース・グループの名前。

SelectFromPolicy
リソース・グループ・メンバーごとに定義され、いずれのノードでリソースを開始するのが望ましいのかを指定します。

ConfigValidity
この属性は、将来の利用のために予約されています。

RecoveryPolicy
障害に対応する方法を指定します。

Mandatory 属性
Mandatory 永続属性は、管理対象リソースが必須または非必須の いずれであるかを明示するために使用されます。

リソース・グループが開始されると、グループ内の必須であるすべての管理対象リソースも開始されなければなりません。非必須の (Mandatory 属性が False に設定されている) 管理対象リソースは、競合が存在する場合は開始されないことがあります。必須の管理対象リソースに障害が発生した場合、リソース・グループ全体が停止し、別のノード上で開始されますが、リソース・グループの非必須メンバーに障害が発生した場合、リソース・グループはそのノード上でオンライン状態を保ちます。

リソース・グループのメンバーであるリソースは、この属性値が明示的に False に 設定されていなければ、暗黙的に必須です。Mandatory 管理対象リソース属性が False であるメンバー・リソースは、リソース・グループをアクティブにするために犠牲になることがあります。

SelectFromPolicy に「**Ordered,Failback**」を設定する には、リソースがグループの必須メンバーでなければなりません。

MemberOf 属性
リソースがリソース・グループ・リソースに含まれていることを示します。

MemberOf 永続属性は、リソース (ネストされたリソース・グループを含む) が リソース・グループに追加されると暗黙的に生成されます。MemberOf 属性は、(停止命令によって明示的に、またはリカバリー不能メンバー・リソースの障害によって 暗黙的に) リソース・グループがアクティブまたは非アクティブにされたときに開始 および停止されるリソースのセットを決定するために使用します。リソースは、1 つのリソース・グループのみのメンバーにすることができます。

SelectFromPolicy 属性
SelectFromPolicy 属性は、いずれのノードでリソースを開始するのが 望ましいのかを定義します。

注 : SelectFromPolicy 属性は、取得済みの IBM.AgFileSystem リソースについては無視されます。

以下に示す 3 つの設定が使用可能です。

Any
System Automation for Multiplatforms はノードをランダムに選出します。

Ordered
System Automation for Multiplatforms は、次に選択可能なノードを決定する際に、必ずリストの最初から 開始します。

Ordered,Failback
「Ordered,Failback」は、「Ordered」と同等ですが、ホーム・ノードへのフェイルバック機能が実装されます。ホーム・ノードへの フェイルバック機能について詳しくは、62 ページの『ホーム・ノードへのフェイルバック機能』を 参照してください。

RecoveryPolicy 属性
RecoveryPolicy 属性は、System Automation for Multiplatforms による障害、自動回復または手動回復への対応方法を指定するために使用します。

次の 3 つの設定が可能です。

AutomaticRecovery
障害から自動的に回復します。これはデフォルトです。

LockOnResFailure
グループのメンバーに障害が発生した場合に、そのリソース・グループに対するロック要求を実行依頼します。 メンバーをホスティングするノードに障害が発生した場合は、そのメンバーを自動的に回復します。

LockOnAnyFailure
メンバーまたはホスティング・ノードに障害が発生した場合に、リソース・グループに対するロック要求を実行依頼します。

この属性は、メンバーまたはホスティング・ノードに障害が発生した場合に、System Automation for Multiplatforms によって最初に評価される属性です。 System Automation for Multiplatforms は、障害のタ

イプと RecoveryPolicy の設定に基づいて、 自動的に回復するか、または自動回復アクションを回避するために、グループに対するロック要求を実行依頼します。

ロック要求を使用すると、System Automation for Multiplatforms が障害を解決できない場合、または再始動の前に診断データを保存する必要がある場合などに、オペレーターが障害を手動で回復させることができます。 ロック要求は、オペレーターに代わって System Automation for Multiplatforms が自動的に処理し、すべてのグループ・メンバーを対象とするロック要求として、関係に沿って伝搬されます。ロックが使用されると、従属リソースまたは他のグループ・メンバーを強制的に停止するようなすべての回復アクションを、System Automation for Multiplatforms が実行できなくなります。このようなアクションは、オペレーターが手動で実行する必要があり、オペレーターがロック要求を除去しても、System Automation for Multiplatforms では処理されません。

障害の状態は、グループおよび障害が発生したメンバーについての操作状態 LockedByFailure によって示されます。ロック要求のコメント・セクションに、そのロック要求を処理する障害が発生しているリソースまたはノードがリストされます。ロック要求をリストするには、コマンド `lsrgreq –L <group name>` を使用します。オペレーターは、手動で回復アクションを実行すると、コマンド `rgreq –o unlock <group>` を使用してグループからロック要求を削除することで、自動化を再開できるようになります。

注 : 手動による回復ステップを実行せずに、オペレーターがロック要求を除去した場合でかつ、その障害が発生したリソースにクリーンアップが定義されている場合にのみ、System Automation for Multiplatform は、リソースの自動的な再始動の前にクリーンアップを処理します。

ネストされたグループに RecoveryPolicy を設定することにより、リソースの手動回復の有効範囲を広げることができます。障害の発生時にロックを起動するすべてのネストされたグループまたはリソースが、そのロックの有効範囲内で同じ RecoveryPolicy の定義値を持っている必要があります。

リソースのグループ化

System Automation for Multiplatforms は、リソースをグループ化することにより、複数のリソースを単一ユニットとして管理します。

System Automation for Multiplatforms では、リソースをグループ化できます。リソースをグループ化すると、ロケーション関係や開始および停止関係などの、グループのメンバー間のすべての関係を設定できます。構成が完了すると、単一ユニットとしてのグループに対して操作を実行できるようになります。 リソース・グループを操作することにより、オペレーターがアプリケーション・コンポーネントとそれらの関係を記憶する必要がなくなるため、オペレーター・エラーのリスクが削減されます。

管理対象関係

以下のトピックで、管理対象関係とその使用法について説明します。

29 ページの表 8 は、管理対象関係に使用する System Automation for Multiplatforms コマンドのリストです。 詳しくは、「*Tivoli System Automation for Multiplatforms* リファレンス・ガイド」のを参照してください。

関係の作成および管理方法を学習するには、136 ページの『関係の管理』 を参照してください。

表 *8.* 管理対象関係に使用する *System Automation for Multiplatforms* コマンド

コマンド	説明
mkrel	管理対象関係を作成する
lsrel	管理対象関係をリストする
rmrel	管理対象関係を除去する
chrel	管理対象関係を変更する

注 : System Automation の決定は、グループとリソース間の関係の組み合わせなどのさまざまな要因、属性設定、およびグループの公称状態に基づいて行われます。これらの要因のそれぞれが、自動化動作に影響する可能性があります。 以下のトピックの例は、関係の使用時に起こる代表的な自動化動作の説明に過ぎず、実際の動作は、ここに記載されたものとは異なることがあります。

管理対象関係

管理対象関係は、1 つのソース・リソースと、1 つ以上のターゲット・リソースの 間に存在します。

最初の例として、リソース・グループの開始時に、ソース・リソースが、ターゲット・リソースがオンラインになった後に開始されなければならない場合を挙げます。管理対象関係のこの例では、Relationship 属性の値 StartAfter を 使用します。

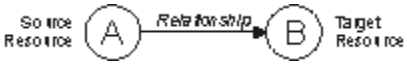


図 *11.* 管理対象関係の例

例に示すように、関係は常に方向を示します。 関係は、ノードの境界を超えて成立します。

2 番目の例として、可能な場合はソース・リソースがターゲット・リソースと同じ ノード上で開始する必要がある場合を挙げます。管理対象関係のこの例では、Relationship 属性の値 Affinity を使用します (31 ページの『Relationship 属性』で説明します)。

Relationship 属性は、31 ページの『Condition 属性』とともに使用できます。3 番目の例として、ターゲット・リソースが オンラインの場合に限り、可能な場合は、ソース・リソースが ターゲット・リソースと同じノード上で開始される必要がある場合を挙げます。管理対象関係のこの例では、Relationship 属性の値 Affinity を、Condition 属性の値 IfOnline と ともに使用します。

Relationship 属性については 31 ページの『Relationship 属性』で、Condition 属性については 31 ページの『Condition 属性』で説明します。

管理対象関係を組み合わせて使用することにより、複雑な自動化のシナリオを定義できます。

管理対象関係のターゲットとすることができるのは、Name および OpState を備えた任意のタイプのリソースです。ただし、リソースの目標が競合しないようにするために、集合リソースの 1 つの構成要素および同値のメンバーはどちらも、管理対象関係のターゲットとすることはできません。

関係は、1 つのソース・リソースと 1 つ以上のターゲット・リソースの間で定義されます。 関係のソース・リソース (リソース・グループ・メンバー またはリソース・グループの場合もあります) を除去した場合、関係は 削除されます。ターゲット・リソースのリストから最後のターゲット・リソース (リソース・グループ・メンバー、リソース・グループ、または基礎となる RMC リソースの場合があります) を除去した場合、その関係も同様に削除されます。

管理対象関係は System Automation for Multiplatforms リソース・クラス `IBM.ManagedRelationship` で提供されます。

管理対象関係によって使用される属性

以下の図は、管理対象関係の別の例を示しています。

図 *12.* 管理対象関係の例

管理対象関係には、以下のセクションで説明する 属性が保持されます。

Name 属性

Name 永続属性を使用して、管理対象関係について使用する 名前を指定します。この属性はオプションです。これにより、関係をより簡単に変更または削除できます。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **`sampolicy`**
- System Automation for Multiplatforms への接続。

Source 属性
Source 永続属性を使用して、管理対象関係の ソース・リソースを指定します。

Target 属性
Target 永続属性を使用して、管理対象関係のターゲット・リソース・ リストを指定します。

Relationship 属性
Relationship 永続属性を使用して、ソース・リソースとターゲット・リソースとの間に 適用する関係を指定します。

関係には、開始/停止依存関係 およびロケーション依存関係の 2 種類があります。

- 開始または停止依存関係は、開始または停止動作を定義するために使用します。
  - StartAfter
  - StopAfter
  - 依存
  - DependsOnAny
  - ForcedDownBy
- ロケーション依存関係は、ノード上にリソースを配置するために使用します。
  - Collocated
  - AntiCollocated
  - Affinity
  - AntiAffinity
  - IsStartable

Condition 属性
Condition 永続属性は、IsStartable 管理対象関係を除くすべてのロケーション関係とともに 使用する条件を指定します。StartAfter 関係と組み合わせて使用できるのは、IfPossible 条件のみです。

Condition 永続属性 は、関係がいつ適用可能と見なされるかを定義します。以下に、適用可能な条件を示します。

- IfOnline
- IfNotOnline
- IfOffline
- IfNotOffline
- IfPossible
- IfIWasOnline
- IfIWasNotOnline
- なし

開始または停止動作の関係

System Automation for Multiplatforms では、開始/停止動作を定義するために使用できる 以下の関係が提供されています。

- **StartAfter**
- **StopAfter**
- **DependsOn**
- **DependsOnAny**
- **ForcedDownBy**

開始/停止関係のソースは、リソース・グループのメンバーまたはリソース・グループ のいずれかです。リソース・グループについての詳細は、18 ページの『リソース・グループとは』を 参照してください。

開始/停止関係のターゲットは、以下のいずれかです。

- リソース・グループのメンバーまたはリソース・グループ
- 同値
- OpState 属性を提供する必要の ある (管理対象リソースでない) RSCT リソース

DependsOn 関係で、ソース・リソースまたはターゲット・リソース、またはその両方がグループの場合は、これらのグループのメンバー・ロケーションは「Collocated」でなければならないことに注意してください。

開始コマンドは、リソースに対して直接発行できません。したがって、リソースがそのメンバーであるリソース・グループの公称状態をオンラインに設定することにより、リソースを開始します。

並行リソースは DependsOnAny 関係のソースとしても、ターゲットとしても 許可されません。このような関係の結果は、 定義されていない自動化動作になります。

StartAfter 関係

StartAfter 関係を使用して、ターゲット・リソース (1 つ以上) がオンラインの場合にのみ、ソース・リソースが開始されるようにします。

StartAfter 関係により、以下の動作方式が提供されます。

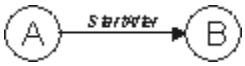


図 *13. StartAfter* 関係の例

- 開始動作で、StartAfter は、次のようにリソース A および B の 開始シーケンスを定義します。ソース・リソース A を開始する必要があるときは、 最初にターゲット・リソース B が開始されます。リソース B がオンラインになった後、リソース A が開始されます。

  リソース A およびリソース B は、別のノードで開始できることに 注意してください。

  2 つのリソース・グループ間の StartAfter 関係と同時に IfPossible 条件を使用する場合、ターゲット・リソース・グループ B が結合できずに Sacrificed バインド状態なるとき、そのターゲット・リソース・グループ B は迂回されるため、この動作は異なったものとなります (56 ページの『ノード位置の割り当て』を参照)。 この場合、関係のソースは関係を無視して開始されます。

StartAfter 関係では、強制停止動作 (38 ページの『DependsOn 関係』を参照) は提 供されません。

ターゲットが同値の場合、並行リソースは StartAfter 関係のソースとして 許可されません。このような関係の結果は、 定義されていない自動化動作になります。

StartAfter 関係の開始動作
開始動作は、ターゲット・リソースの操作状態 (OpState) を介して 制御されます。リソース B の操作状態がオンラインになると、リソース A が開始されます。

多くの場合、リソース A およびリソース B は、同じリソース・グループのメンバーです。

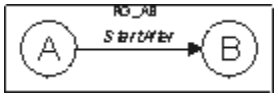


図 *14. StartAfter* 関係の開始動作

それらのリソース・グループの公称状態をオンラインに設定すると、A および B の両方のメンバーが開始されます。 A から B に対する StartAfter 関係 により、リソース B が最初に開始されます。リソース B の操作状態が オンラインになると、リソース A が開始されます。

リソース A がリソース・グループ RG_A のメンバーで、リソース B がリソース・グループ RG_BC のメンバーであり、A と B の間で StartAfter 関係が定義 されているとします。この場合、RG_A の公称状態をオンラインに設定することにより、StartAfter 関係の開始動作が起動されます。

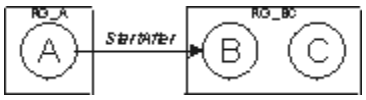


図 *15.* リソース・グループが異なる *2* つのリソース間における *StartAfter* 関係の例

StartAfter 関係の開始シーケンスにより、リソース B が最初に開始される必要があります。 RG_BC の公称状態をオフラインに設定すると、以下の競合が 発生します。RG_BC ではリソース B がオフラインであることが要求されますが、StartAfter 関係により B が 強制的に開始されます。System Automation for Multiplatforms では、オンライン要求は常に オフライン要求より重要であるとしてこの競合を解決します。このため、RG_BC の他のグループ・メンバーが、グループの公称状態が オフラインであるために開始されなくても、リソース B は開始されます。リソース B がオンラインになった後、System Automation for Multiplatforms がリソース A を開始します。リソース C は 開始されません。

リソース A が開始されていて、リソース B が StartAfter 関係によって開始されるが、構成グループの公称状態が「オフライン」のままである場合、リソース B はリカバリーされます。すなわち、B は、障害が発生した場合に再始動されます。この拡張動作は、System Automation for Multiplatforms のリリース 2.3 で導入されました。以前のリリースでは、B は障害が発生した場合でも再始動されませんでした。

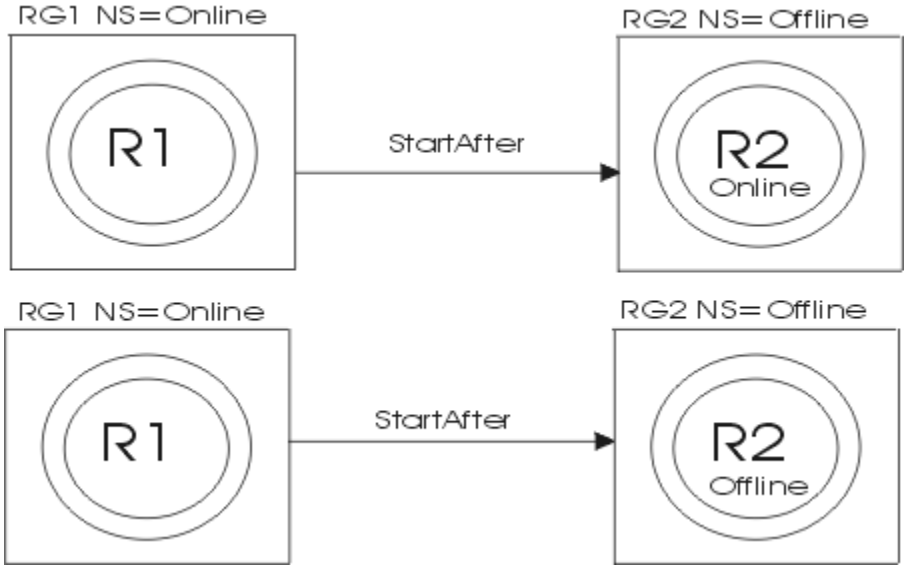


図 *16. 2* つのリソース間における *StartAfter* 関係の例

開始順序は、関係の順方向にのみ作用します。リソース A およびリソース B が異なるリソース・グループに属す (A が RG_A、B が RG_B に属す) 場合は、RG_B の公称状態をオンラインに 設定しても、リソース B はリソース A に対して順方向の関係を持っていないため、リソース A に対するアクションは実行されません。

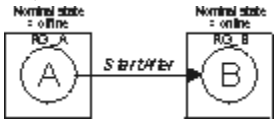


図 *17.* ターゲット・リソースが既にオンラインである場合の *StartAfter* 関係の例

RG_A の公称状態がオンラインに設定されると、リソース B が既にオンラインであるため、リソース A はただちに開始できます。

別のシナリオとして、リソース A がリソース B およびリソース C に対して、StartAfter 関係を保持しているとします。

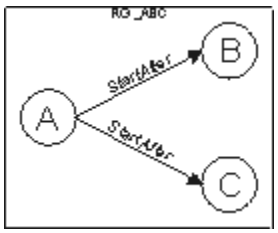


図 *18.* 単一リソース・グループ内に *StartAfter* 関係を *2* つ持つリソースの例

この場合、A を開始するには、System Automation for Multiplatforms がリソース A を開始する前に、リソース B および C の両方がオンラインになっている必要があります。例えば、A、B、および C はリソース・グループ RG_ABC のメンバーです。RG_ABC の公称状態をオンラインに設定すると、まずリソース B および C が 並行して開始されます。両方のリソースの操作状態が オンラインになると、リソース A が開始されます。

別の例として、リソース A がリソース・グループ RG_A の、リソース B が リソース・グループ RG_B の、リソース C がリソース・グループ RG_C のそれぞれメンバーであることも考えられます。

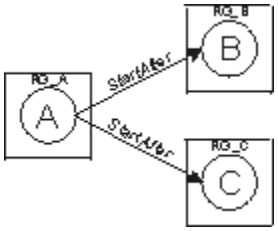


図 *19.* 異なるリソース・グループの リソースに対する *2* つの *StartAfter* 関係を持つリソースの例

A には、B および C の両方に対する StartAfter 関係があります。RG_A の公称状態をオンラインに設定すると、StartAfter 関係によって、リソース C およびリソース B が開始されます。 リソース B および C の 両方がオンラインになった後、A が開始されます。

以下の図の例で、2 つのリソース・グループ (「RG1」と「RG2」) が StartAfter/IfPossible 関係によって接続され、ターゲット・リソース・グループ (「RG2」) が結合できないときに起こる開始の動作を示します。

- 関係のソース (「RG1」) が開始されます。
- ターゲット・リソース・グループ (「RG2」) との関係は有効と見なされず、その bindingState は Sacrificed に設定されます。両方のリソース・グループが別のグループのメンバーの場合は、異なる動作が起こることに注意してください。

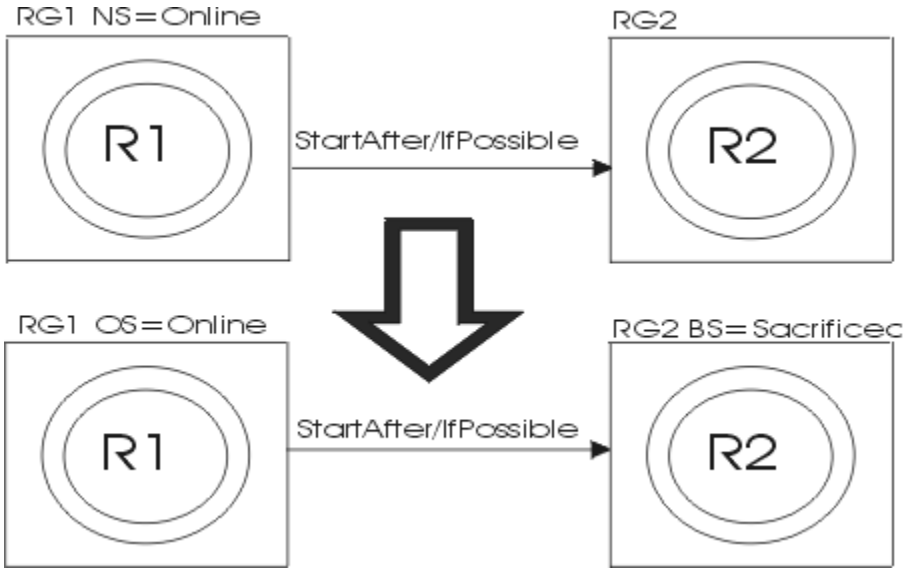


図 *20. StartAfter/IfPossible* 関係の開始動作

StartAfter 関係での IfPossible 条件の使用

StartAfter 関係で IfPossible 条件を指定することができます。この条件は、ターゲット・リソース・グループが結合できない場合、このグループをバイパスできることを指定します。この場合、ターゲット・リソース・グループは「Sacrificed」状態になり、StartAfter 関係は無視されます。StartAfter/IfPossible は、グループ間にのみ使用してください。サポート・リソースとして管理対象リソースを指定した場合の結果は未定義です。

StartAfter 関係の停止動作

ソース・リソース A がオンラインの間は、ターゲット・リソース B は停止できません。ソース・リソース A の NominalState 属性がオフラインに変更されると、ターゲット・リソース B は自動的に停止します。両方のリソースを同時に停止させることが できます。

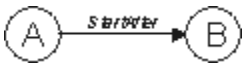
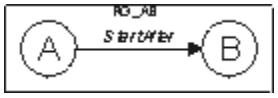


図 *21. StartAfter* 関係の例

多くの場合、ソース・リソース A およびターゲット・リソース B は、同じリソース・グループの メンバーです。このため、これらの NominalState 値は同じです。

図 *22.* 同じリソース・グループのリソース間における *StartAfter* 関係



RG_AB の NominalState 属性をオフラインに設定して、メンバー A および B の両方を停止します。StartAfter 関係に停止シーケンスは不要なため、リソース A および B を同時に停止できます。

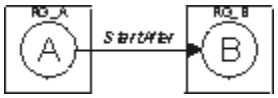


図 *23.* 異なるリソース・グループのリソース間における *StartAfter* 関係

RG_B の NominalState がオンラインであるとすると、リソース・グループ RG_B に影響を 与えることなく RG_A を開始および停止できます。RG_B はオンラインのままです。

RG_B の公称状態をオフラインに、RG_A の NominalState をオンラインに設定すると、ターゲット・リソース B は、ソース・リソース A の前に開始されます。 RG_A の NominalState をオフラインに設定すると、リソース A および B が同時に 停止されます。

以下の停止動作を考えてみましょう。

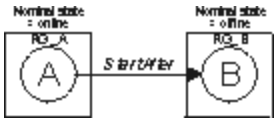


図 *24.* ターゲット・リソースの公称状態がオフラインである *StartAfter* 関係

RG_A の NominalState が「オンライン」、かつ RG_B の NominalState が「オフライン」ならば、リソース A およびリソース B はオンラインです。 ここで、RG_A の NominalState をオフラインに 設定します。リソース A およびリソース B は同時に停止します。このような動作に なるのは、StartAfter 関係を介して渡されたリソース・グループ RG_A についての 開始要求によってリソース B が開始されたためです。RG_A をオフラインに 設定することにより開始要求が除去され、リソース・グループ RG_B の NominalState がオフラインで あることにより、リソース B が停止されます。

StartAfter 関係により 典型的な停止動作が発生します。リソース A および B は同時に停止させることができます。

リソース A、B、および C は、それぞれリソース・グループ RG_A、RG_B、および RG_C の メンバーです。

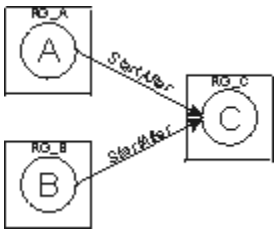


図 *25.* 異なるリソース・グループの同じリソースに対する *2* つの独立したリソースの *StartAfter* 関係

A および B の両リソースをサポートするには、リソース C はオンラインでなければなりません。RG_A か RG_B のいずれか、または両方の NominalState が「オンライン」である限り、RG_C の NominalState が「オフライン」であっても、リソース C が「オンライン」であり続ける必要があります。 RG_A および RG_B の両方の NominalState がオフラインである場合にのみ、リソース C は 停止可能です。これは、RG_C の NominalState がオフラインである場合も同様です。

StartAfter 関係の使用に関するルール

1. StartAfter 関係は、既存の DependsOn 関係と競合しては なりません。
2. StartAfter 関係は、管理対象リソース間に存在するロケーション関係を 前提とするものではありません。ロケーション関係を定義する場合は (44 ページの『ロケーション関係』を参照)、この目的で追加の関係を作成する 必要があります。
3. System Automation for Multiplatforms は、ソース・リソースの開始要求を受け取ると、必ず最初に ターゲット・リソースの開始を試行します。
4. ターゲット・リソースが障害になった場合、ソース・リソースが停止される わけではありません。

StopAfter 関係

StopAfter 関係を使用して、ターゲット・リソースが停止済みの場合にのみ、ソース・リソースを停止できるようにします。

StopAfter 関係により、以下の動作方式が提供されます。

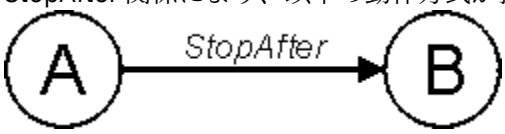


図 *26. StopAfter* 関係の例

- ターゲット・リソースが「オフライン」(「オフラインに失敗」を含む) にならないかぎり、リソース A は停止されません。

StopAfter 関係では、開始動作と強制停止動作は提供されません (32 ページの『StartAfter 関係』および 38 ページの『DependsOn 関係』を参照)。

ターゲットが同値の場合、並行リソースは StopAfter 関係のソースとして 許可されません。このような関係の結果は、 定義されていない自動化動作になります。

StopAfter 関係の停止動作

ターゲット・リソース B が「オンライン」の間は、ソース・リソース A は停止 できません。ターゲット・リソース B の OpState 属性が「オフライン」または「オフラインに失敗」に変更されると、ソース・リソース A は自動的に停止します。

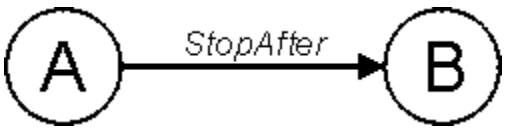


図 *27. StopAfter* 関係の例

多くの場合、ソース・リソース A およびターゲット・リソース B は、同じリソース・グループの メンバーです。RG_AB の NominalState 属性を「オンライン」に設定すると、メンバー A および B の両方 が開始します。StopAfter 関係に開始シーケンスは不要なため、リソース A お よび B を同時に開始できます。リソース・グループの NominalState 属性を「オフライン」に設定すると、メンバーが停止します。A から B に対する関係 によって、リソース B が最初に停止されます。リソース B の操作状態が「オフライン」になると、リソース A が停止します。

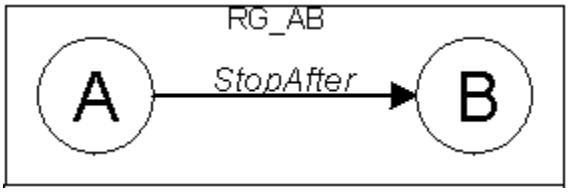


図 *28.* 同じリソース・グループのリソース間における *StopAfter* 関係

リソース A と B がそれぞれ異なるリソース・グループに属していて (A が RG_A、B が RG_B に属している)、RG_B の NominalState が「オフライン」の場合、リソー ス・グループ RG_B へ依存せずに RG_A を開始および停止できます。RG_B の NominalState を「オンライン」に設定し、RG_A の NominalState を「オフライン」に設定する と、ターゲット・リソース B が「オンライン」である間はソース・リソース A を停止する ことはできません。

図 *29.* 異なるリソース・グループのリソース間における *StopAfter* 関係

RG_A の NominalState が「オフライン」の場合は、リソース A に依存せずに RG_B を開始したり停止したりできます。リソース A がリソース・グループ RG_A のメンバー、リソース B がリソース・グループ RG_B のメンバーであり、リソース C がリソース・グループ RG_C のメンバーである場合もあります。A は、B と C の両方に対し StopAfter 関係を持っています。

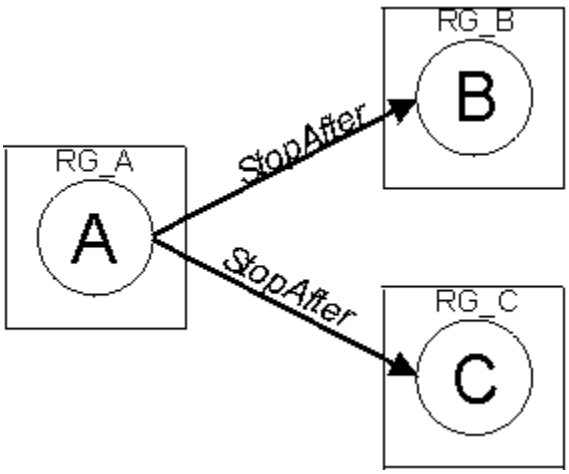


図 *30.* 異なるリソース・グループの リソースに対する *2* つの *StopAfter* 関係を持つリソースの例

RG_A の NominalState が「オンライン」であるときに RG_A を停止する場合、RG_B と RG_C の両方の NominalState が「オンライン」である間は RG_A を停止することはできません。 RG_B および RG_C の両方の NominalState が「オフライン」または「オフラインに失敗」で ある場合にのみ、リソース A は停止可能です。

DependsOn 関係

System Automation for Multiplatforms では、DependsOn 関係を使用して、ターゲット・リソース (1 つ以上) がオンラインの 場合にのみ、ソース・リソースを開始できるようにします。

これは、以下の点を 除いて StartAfter 関係と同様に使用します。

- DependsOn 関係には、ソース・リソースおよびターゲット・リソース間の 暗黙的なコロケーション (45 ページの『Collocated 関係 』で説明) も含まれます。
- ターゲット・リソースで障害が発生すると、ソース・リソースも停止します。

DependsOn 関係により、以下の 3 つの動作方式が提供されます。

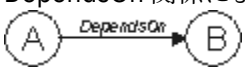


図 *31. DependsOn* 関係*:* 動作方式

リソース A はリソース B の機能に依存し、これはリソース A がリソース B なしでは機能できないことを意味します。したがって、強制停止動作が導入されます (以下のリストの項目 38 ページの『3』 を参照)。

1. 開始動作で、DependsOn は暗黙的なコロケーションにより、次のように リソース A および B の開始シーケンスを定義します。リソース A (ソース) を開始する必要があるときは、最初にターゲット・リソース B が開始されます。リソース B がオンラインになった後、同じノード上でリソース A (ソース) が開始されます。
2. 停止動作に関して、DependsOn は次のようにリソース A および B の 停止シーケンスを定義します。リソース B (ターゲット) を停止する必要があるときは、最初にソース・リソース A が停止されます。リソース A がオフラインになった後、リソース B (ターゲット) が 停止されます。
3. ターゲット・リソースの障害時における強制停止動作: ターゲット・リソース B が 障害になると、リソース A も停止されます。続いて、38 ページの『1』 で 説明している開始動作に従って再始動が起動されます。

DependsOn 関係の開始動作

DependsOn 関係の開始順序付けは、ターゲット・リソースの 操作状態 (OpState) を介して制御されます。リソース B の操作状態がオンラインになると、リソース A が開始されます。開始シーケンスに 加えて、DependsOn により、リソース A はリソース B が開始されたノードと 同じノード上で開始されなければならないという連結制約が提供されます。このため、リソース B は、後でリソース A を開始できるノード上で 既に開始されています。DependsOn 関係の一部である連結制約は、Collocated 関係の動作に相当します。この 動作について詳細は、45 ページの『Collocated 関係 』を参照してください。

多くの場合、リソース A およびリソース B は、同じリソース・グループの メンバーです。

図 *32. DependsOn* 関係*:* 開始動作パート *I*

それらのリソース・グループの公称状態をオンラインに設定すると、A および B の両方のメンバーが開始されます。 A から B に対する DependsOn 関係 によって、リソース B が最初に開始されます。リソース B の操作状態が オンラインになると、リソース A が同じノード上で開始されます。

リソース A がリソース・グループ RG_A のメンバーで、リソース B が リソース・グループ RG_BC のメンバーであり、A から B に対して DependsOn 関係が定義 されているとします。この場合、RG_A の公称状態をオンラインに設定することにより、DependsOn 関係の開始動作が起動されます。

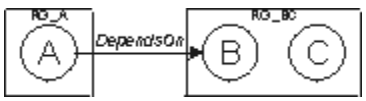


図 *33. DependsOn* 関係*:* 開始動作パート *II*

DependsOn 関係の開始シーケンスにより、リソース B が最初に開始される必要があります。 RG_BC の公称状態をオフラインに設定すると、以下の競合が 発生します。RG_BC ではリソース B がオフラインであることが要求されますが、DependsOn 関係により B が 強制的に開始されます。System Automation for

Multiplatforms では、オンライン要求は常にオフライン要求より 重要であるとしてこの競合を解決します。このため、RG_BC の他のグループ・メンバーが、グループの公称状態がオフラインで あるために開始されなくても、リソース B は開始されます。リソース B が オンラインになった後、System Automation for Multiplatforms がリソース A を開始します。当然ながらリソース A および B は 同じノード上で開始されます。リソース C は開始されません。

開始順序は、関係の順方向にのみ作用します。リソース A およびリソース B が異なるリソース・グループに属す (A が RG_A、B が RG_B に属す) 場合は、

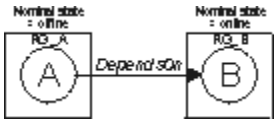


図 *34. DependsOn* 関係*:* 開始動作パート *III*

RG_B の公称状態をオンラインに設定しても、リソース B からリソース A への順方向の関係はないため、リソース A について何のアクションも実行されません。続いて RG_A の公称状態もオンラインに設定すると、リソース B が既にオンラインになっている場合と同じノード上でリソース A をただちに開始させることができます。

別のシナリオとして、リソース A がリソース B およびリソース C に対して、DependsOn 関係を保持しているとします。

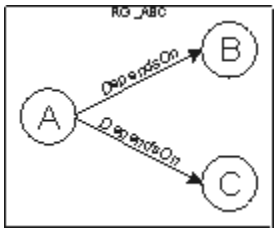


図 *35. DependsOn* 関係*:* 開始動作パート *IV*

この場合、A を開始するには、System Automation for Multiplatforms がリソース A を開始する前に、リソース B および C の両方がオンラインになっている必要があります。例えば、A、B、および C はリソース・グループ RG_ABC のメンバーです。RG_ABC の公称状態をオンラインに設定すると、まずリソース B および C が 並行して開始されます。両方のリソースの操作状態がオンラインになると、リソース A が開始されます。A は、B および C が稼働しているノードと同じ ノード上で開始されなければならないため、3 つのすべてのリソースが同じノード上で開始 されます。

別の例として、リソース A がリソース・グループ RG_A の、リソース B が リソース・グループ RG_B の、リソース C がリソース・グループ RG_C のそれぞれメンバーである場合もあります。

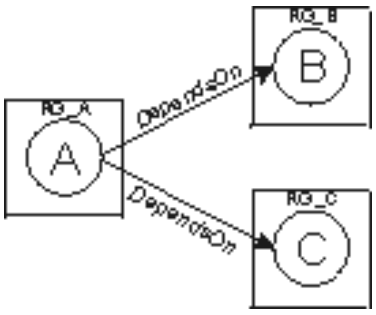


図 *36. DependsOn* 関係*:* 開始動作パート *V*

A には、B および C の両方に対する DependsOn 関係があります。RG_A の公称状態をオンラインに設定すると、リソース B およびリソース C が開始されます。 リソース B および C の両方がオンラインになった後、同じノード上で A が開始されます。

DependsOn 関係の停止動作

DependsOn 関係の停止順序付けは、ソース・リソースの 操作状態 (OpState) を介して制御できます。

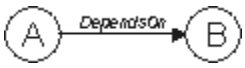


図 *37. DependsOn* 関係*:* 停止動作パート *I*

リソース A の OpState がオフラインになると、リソース B を停止できます。

多くの場合、リソース A およびリソース B は、同じリソース・グループの メンバーです。

図 *38. DependsOn* 関係*:* 停止動作パート *II*

A および B の両メンバーを停止させるには、リソース・グループの公称状態属性をオフラインに設定します。DependsOn 関係により、リソース A が最初に停止します。 リソース A が オフラインになると、リソース B が停止します。

リソース A がリソース・グループ RG_A のメンバーで、リソース B が リソース・グループ RG_B のメンバーであり、A から B に対して DependsOn 関係が定義 されているとします。この場合、RG_B の公称状態をオフラインに設定することにより、DependsOn 関係の停止動作を起動できます (System Automation for Multiplatforms では、リソースを直接停止 させることはできません)。DependsOn 関係によって、リソース A が最初に 停止します。リソース・グループ A の公称状態をオンラインに設定すると、競合が発生します。RG_A ではリソース A がオンラインであることが要求されますが、DependsOn 関係 により A が停止されます。

図 *39. DependsOn* 関係*:* 停止動作パート *III*

この競合は、System Automation for Multiplatforms では、オンライン要求は常にオフライン要求より重要であるという方式で解決されます。 このため、リソース A はオンラインのまま保持され、リソース B は 停止できません。RG_A の公称状態をオフラインに設定した場合にのみ、リソース A を停止できます。リソース A がオフラインになると、それに続いてリソース B が停止します。

また、考慮すべき暗黙的な停止動作もあります。

図 *40. DependsOn* 関係*:* 停止動作パート *IV*

RG_A の公称状態がオンラインで、リソース・グループ RG_B の公称状態がオフラインに設定されると、前述の開始のシナリオで示したように、リソース A および B がオンラインです。 ここで RG_A の公称状態はオフラインに設定されています。これにより、リソース A が停止します。さらに、リソース B が 停止します。リソース B が停止するのは、リソース・グループ RG_A に 対する開始要求が、DependsOn 関係を介してリソース B に伝搬された ためです。このリソース・グループ RG_A がオフラインに設定されているため、開始要求が除去され、リソース・グループ RG_B の公称状態のオフラインにより B が停止されます。DependsOn 関係により、典型的な停止動作が 発生します。リソース A の停止前にリソース B を停止させることはできません。このため、最初にリソース A が停止します。A がオフラインになると、それに続いてリソース B が停止します。

別のシナリオとして、リソース A および B が C に対して DependsOn 関係を保持している場合を例に挙げます。リソース C を停止するには、最初にリソース A および B の両方をオフラインにすることが 必要です。

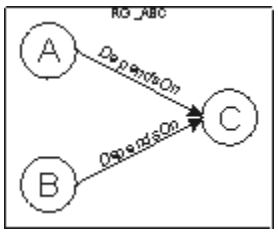


図 *41. DependsOn* 関係*:* 停止動作パート *V*

例えば、A、B、および C が、同じリソース・グループ RG_ABC のメンバーです。 RG_ABC の公称状態をオフラインに 設定すると、リソース A および B が最初に停止します。両方のリソースの 操作状態がオフラインになると、リソース C が停止します。もう 1 つの例として、リソース A、B、および C が、それぞれリソース・グループ RG_A、RG_B、および RG_C の メンバーである場合を挙げます。

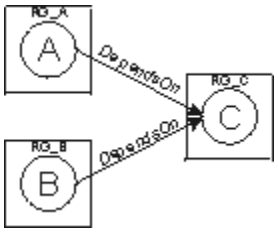


図 *42. DependsOn* 関係*:* 停止動作パート *VI*

RG_C の公称状態をオフラインに設定すると、DependsOn 関係の停止動作が起動されます。 ここで、リソース・グループ RG_A および RG_B の公称状態が 停止動作に優先することがあります。RG_A または RG_B の公称状態がオンラインで ある限り、リソース C は停止できません。これは、競合状態においては、オンライン要求が常にオフライン要求より優先されるためです。このため、DependsOn 関係の停止動作は、RG_A および RG_B の公称状態が オフラインに設定されるまで延期されます。これらのメンバー A および B が オフラインになると、それに続いてリソース C も停止します。

DependsOn 関係の強制停止動作

DependsOn 関係の基本原則は、ソース A はターゲット・リソース B の 機能に依存するということです。ターゲット・リソース B が障害になると、ソース・リソース A はそれ以上機能できません。このため、B を 再始動するのみでは十分ではありません。B の障害のためにリソース A も強制的に 停止されます。その後、開始動作に従って、最初に B、続いて A の順で 両方のリソースが再始動されます。

例として、リソース B に対して DependsOn 関係を保持するリソース A を 定義したとします。

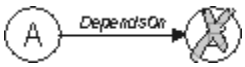


図 *43. DependsOn* 関係*:* 強制停止動作パート *I*

両方のリソースが「オンライン」です。リソース B が障害に なった場合、リソース A が停止し、続いて通常の開始動作が実行されます。リソース B が 再始動され、それに続いてリソース A が再始動されます。

リソース A およびリソース B は、同じリソース・グループ RG_AB のメンバーです。

図 *44. DependsOn* 関係*:* 強制停止動作パート *II*

さらに、リソース A からリソース B に対する DependsOn 関係が定義されています。 RG_AB がオンラインに設定されると、リソース B が最初に開始され、続いてリソース A が開始されます。リソース B が障害またはオフラインになった場合は、リソース A も 停止します。その後、DependsOn の開始シーケンスにより正常再始動が 実行されます。つまり、A が開始される前に B が開始されます。

別の例として、リソース A がリソース・グループ RG_A のメンバーで、リソース B が リソース・グループ RG_B のメンバーであり、A は B との間に DependsOn 関係を保持していると します。

図 *45. DependsOn* 関係*:* 強制停止動作パート *III*

リソース・グループ RG_A がオンラインに、RG_B の公称状態がオフラインに設定されると、リソース B が最初に開始され、その後にリソース A が開始されます。 DependsOn 関係の強制停止動作が、リソース B の障害により起動されます。これにより、リソース A も 停止します。RG_A の公称状態が「オンライン」である場合も、上記の状態になります。System Automation for Multiplatforms では、強制停止動作は常に リソース・グループのオンライン要求より重要であるとして この競合を解決します。

強制停止動作は、DependsOn 関係のチェーンを介して伝搬します。以下のシナリオを例に挙げます。リソース A がリソース・グループ RG_A の、リソース B が リソース・グループ RG_B の、リソース C がリソース・グループ RG_C のそれぞれメンバーであり、A から B に対して、および B から C に対して DependsOn 関係があるとします。

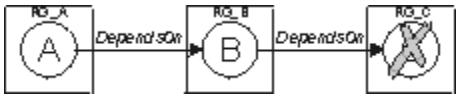


図 *46. DependsOn* 関係*:* 強制停止動作パート *IV*

リソース・グループ RG_A がオンラインに設定され、これにより 3 つのリソース C、B、および A が順次開始されて、オンライン状態にあるとします。 ここでリソース C が障害になります。これにより、リソース A および B が強制的に 停止されます。最初にリソース A が停止し、続いてリソース B が停止します。これは、強制停止動作の方が通常のオンライン要求より重要度が高い ためです。

DependsOn 関係の使用に関するルール

DependsOn 関係を使用するためのさまざまなルールがあります。

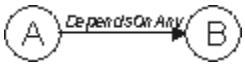
1. ソース・リソースまたはターゲット・リソースがグループの場合、グループのすべてのメンバーは 連結されていなければなりません。
2. 並行リソースのソースと並行リソースまたは同値のターゲットの 間に dependsOn 関係を定義できます。この場合は、特別な動作が実装されます。 ソース・リソースの各固定リソースでは、ターゲット・リソースにある一致する固定リソースとの間に、独自の dependsOn 関係を形成します。 この動作について詳しくは、 66 ページの『並行リソースの自動化動作』を参照してください。

DependsOnAny 関係

DependsOnAny 関係の動作は、開始シーケンスの 連結制約が提供されない点を除いて DependsOn 関係と同じです。このため、ソース・リソースとターゲット・リソースは、同一ノード上または 異なるノード上のいずれかで開始できます。

DependsOnAny 関係により、以下の 3 つの動作方式が提供されます。

図 *47. DependsOnAny* 関係パート *I*



1. 開始動作で、DependsOnAny はロケーション関係なしで、リソース A および B の開始シーケンスを定義します。リソース A (ソース) を開始する必要があるときは、最初にターゲット・リソース B が開始されます。リソース B がオンラインになった後、リソース A (ソース) が開始されます。DependsOn 関係との唯一の違いは、リソース A およびリソース B を 異なるノード上で開始できる点であることに注意してください。
2. 停止動作に関して、DependsOnAny は次のようにリソース A および B の 停止シーケンスを定義します。リソース B (ターゲット) を停止する必要があるときは、最初にソース・リソース A が停止されます。リソース A がオフラインになった後、リソース B (ターゲット) が 停止されます。

3. ターゲット・リソースの障害時における強制停止動作: ターゲット・リソース B が 障害になると、リソース A も停止されます。続いて、42 ページの『1』 で 説明している開始動作に従って再始動が起動されます。

DependsOnAny 関係についての詳細は、DependsOn 関係を参照してください。

注 : シナリオ ***A ---> DependsOn ----->B*** は、シナリオ ***A ---> DependsOnAny ---> B*** および ***A----> Collocated---->B*** に対応しています。

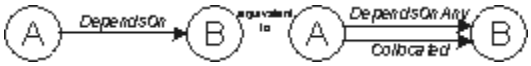


図 *48. DependsOnAny* 関係パート *II*

ForcedDownBy 関係

ForcedDownBy 関係を使用して、ターゲット・リソースがオフラインにな ったときにソース・リソースが停止されるようにします。

ForcedDownBy 関係により、以下の動作方式が提供されます。

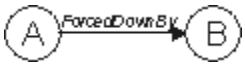


図 *49. ForcedDownBy* 関係

- ターゲット・リソースが予期せずにオフラインになるか、またはターゲット・ リソース自体が強制的にオフラインになる場合は、リソース A を強制的にオフラ インにする必要があります。リソース A と B が同時に停止することがあります。リソース A の強制停止は、その以 前の状態に関係なく、リソース B が先にオンライン状態または何らかの停止状態 (オフラ インに失敗) になった後、通常の停止状態 (オフライン) に入ったときに起動さ れます。

ForcedDownBy 関係では、開始動作と停止動作は提供されません (32 ページの『StartAfter 関係』、36 ページの『StopAfter 関係』、および 38 ページの『DependsOn 関係』 を参照してください)。

ターゲットが同値の場合、並行リソースは ForcedDownBy 関係のソースとして 許可されません。このような関係の結果は、 定義されていない自動化動作になります。

ForcedDownBy 関係の強制停止動作

ForcedDownBy 関係の基本原則は、ターゲット・リソース B がオフラインになるか、ま たは失敗した場合は、ソース A が必ず強制的にオフラインにされることです。

例として、リソース B に対して ForcedDownBy 関係を保持するリソース A を定義し たとします。

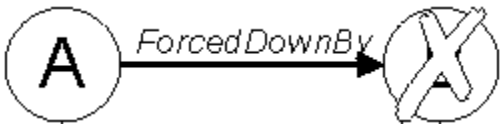


図 *50. ForcedDownBy* 関係*:* 強制停止動作パート *I*

両方のリソースが「オンライン」です。 リソース B が停止ま たは失敗する場合、リソース A が強制停止されます。

別の例として、リソース A がリソース・グループ RG_A のメンバーであり、 リソース B がリソース・グループ RG_B のメンバーであり、A はリソース B との間 に ForcedDownBy 関係を保持しているとします。リソース・グループ RG_A および RG_B の NominalState 属性に「オンライン」が設定されている場合、リソース A および B は互いに対する一切の依存関係なしで開始されます。ForcedDownBy 関係の強制停止動作は、以下のいずれかにより起動されます。

1. リソース B の障害。これにより、リソース A は停止します。RG_A の公称状態が「オンライン」である場合も、上記の状態になります。ただし、この場合 RG_A の公称状態がまだ「オンライン」であるため、リソース A は System Automation for Multiplatforms により再始動されます。

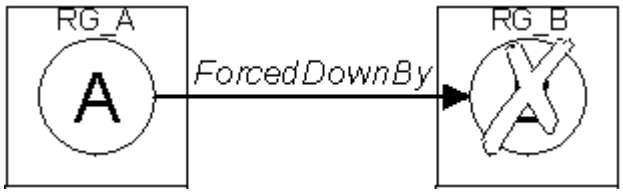


図 *51. ForcedDownBy* 関係*:* 強制停止動作パート *II*

2. リソース B の停止。

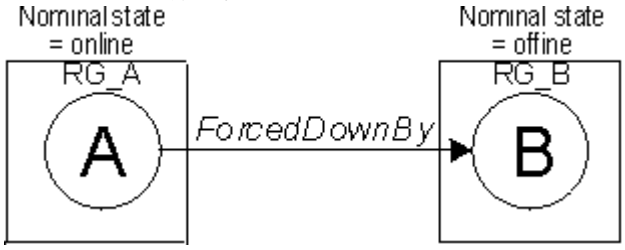


図 *52. ForcedDownBy* 関係*:* 強制停止動作パート *III*

RG_B の公称状態を「オフライン」に設定すると、リソース A も停止します。 RG_A の公称状態が「オンライン」である場合も、上記の状態になります。ただし、この場合 RG_A の公称状態がまだ「オンライン」であるため、リソース A は System Automation for Multiplatforms により再始動されます。

ロケーション関係

System Automation for Multiplatforms では、ロケーション関係を定義するために使用できる 以下の関係が提供されています。

- Collocated
- AntiCollocated
- Affinity
- AntiAffinity
- IsStartable

例えば、リソース A および B は、node1、node2、および node3 で 始動可能な浮動リソースです。

図 *53.* ロケーション関係

これらの関係の背後にあるのは、リソース間のロケーション制約を定義するという考えです。浮動リソースなどのリソース・タイプ、およびグループは、これらを開始できるノードのリストを提供します。リソース A およびリソース B は、node1、node2、および node3 で始動可能な浮動リソースです。

要件としては、リソース A は必ずリソース B が既に稼働中または稼働することに なっているノード上で開始されなければなりません。この動作は、A から B に対する Collocated 関係を定義することによって指定できます。

リソース B が既に稼働中であるノード上でリソース A が開始されてはならない ということを必要とする逆の動作については、AntiCollocated 関係を 定義することにより指定できます。

リソース A は、可能な場合は リソース B が稼働中のノードで開始される必要があり、そうでない場合は Affinity 関係が使用される任意の場所で始動可能です。Collocated 関係と比較すると、Affinity 関係は、「柔らかい」ロケーション関係を持っています。

AntiAffinity 関係は、可能な場合は、リソース A は B が既に稼働中のノードで 開始されてはならないことを定義するために使用します。この要件を満たすことができない場合にのみ、プロセス A は B が存在するノード上で始動可能です。Affinity 関係と同様に、AntiAffinity 関係にも AntiCollocated 関係と比較して 「柔らかい」位置制約があります。

IsStartable 関係は、ソース・リソース A は、ターゲット・リソース B が 始動可能なノードにのみ配置可能であることを定義します。この関係は、ソース・リソースおよびターゲット・リソースの公称状態がオンラインの場合にのみ考慮されます。1 つのリソース (ソースまたはターゲット) の公称状態がオンラインでない 場合、IsStartable 関係は、公称状態がオフラインの リソースとともに廃棄されます。

条件 *IfOnline*、*IfOffline*、*IfNotOnline*、*IfNotOffline*、*IfIWasOnline*、および *IfIWasNotOnline*

IsStartable を除くすべてのロケーション関係とともに以下の条件を 指定できます。

以下にこれらの条件を示します。

IfOnline
IfOnline は、ターゲット・リソースの OpState がオンラインの場合にのみロケーション関係を 評価することを定義します。そうでない場合、位置は無視されます。IfOnline には、「オンラインの保留中」および「オフラインの保留中」などの状態は含まれません。

IfOffline
IfOffline は、ターゲット・リソースの OpState が「オフライン」、「オフラインに失敗」または「不明」のいずれかの場合にのみロケーション関係が評価されることを意味します。そうでない場合、ロケーション関係は無視されます。

IfNotOnline
IfNotOnline は、ターゲット・リソースが「オンライン」状態でない場合にのみロケーション関係が評価されることを意味します。IfNotOnline には、「オンラインの保留中」および「オンライン中」などの状態が含まれます。そうでない場合、ロケーション関係は無視されます。

IfNotOffline
IfNotOffline は、ターゲット・リソースが「オフライン」、「オフラインに失敗」、または「不明」状態でない場合にのみロケーション関係が評価されることを意味します。そうでない場合、ロケーション関係は無視されます。

IfIWasOnline
IfIWasOnline は、ターゲット・リソースがオンラインであり、ソース・リソースが前にオンラインであったがオフラインになった場合にのみロケーション関係が評価されることを意味します。

IfIWasNotOnline
IfIWasNotOnline は、ターゲット・リソースがオンラインであり、ソース・リソースが前にオンラインでなかった (例えば、初期始動された) 場合にのみロケーション関係が評価されることを意味します。

ロケーション関係の使用に関するルール

1. ロケーション関係のソースは、リソース・グループのメンバーまたはリソース・グループの いずれかです。リソース・グループについての詳細は、18 ページの『リソース・グループとは』を 参照してください。
2. ロケーション関係のターゲットは、以下のいずれかです。
   - リソース・グループのメンバーまたはリソース・グループ
   - 開始/停止方法および OpState 属性を提供する必要の ある (管理対象リソースでない) RMC リソース
3. ソース・リソースまたはターゲット・リソースがグループの場合、グループのすべてのメンバーは 連結されていなければなりません。
4. Collocated 関係は、常にフェイルバック設定より 優先されます。

Collocated 関係

System Automation for Multiplatforms では、Collocated 関係を使用して、ソース・リソースおよびそのターゲット・リソースが 同じ ノード上に配置されるようにします。Collocated 関係により、以下の動作方式が提供されます。

図 *54. Collocated* 関係

- Collocated 関係は、リソース A の開始時に、リソース B が既に稼働中のノード上でのみ始動可能であることを定義します。

Collocated 関係は、47 ページの『ケース IV』で説明しているように、Condition 属性と ともに使用できます。

Collocated 関係の動作
このトピックでは、Collocated 関係の 4 つの状態について詳しく説明します。

ケース I

リソース A の開始時には、リソース B が既に稼働中の同じノードにこれを配置します。稼働中とは、リソース B の OpState が、「オンライン」、「オンラインの保留中」、「オンライン中」、または「オフラインの保留中」のいずれかであることを意味します。

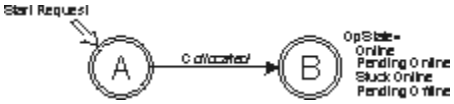


図 *55. Collocated* 関係、ケース *I*

この動作は、標準状態を表します。

Collocated 関係では、将来の状態の予測に基づいて、ノード選択の 最適化が試行されます。以下のケースが考えられます。

ケース II

リソース B が開始され、リソース A が「オフライン」、「オフラインに失敗」、または「不明」状態の いずれかです。

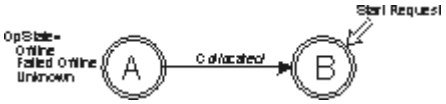


図 *56. Collocated* 関係、ケース *II*

動作としては、リソース B のノード選択がリソース A から独立していることが予想されます。しかし、System Automation for Multiplatforms によるリソース B のノード選択時には、将来、リソース A も開始できるようなノードが選択されます。 この予測アプローチは、後におけるリソース A の 開始動作を単純化するためのものです。エラー状態が発生しない場合に、リソース B が開始された後、リソース A をリソース B が稼働しているノードと 同じノードで開始できるようにします。

ケース III

リソース A が開始され、リソース B が「オフライン」状態です。

図 *57. Collocated* 関係、ケース *III*

理論上、リソース A は B が稼働中のノードにはバインド できないため、現在そのノード・リストの任意のノードに配置 可能です。ここで、再度予測アプローチにより、将来リソース B も始動可能である リソース A のノード位置の検索が試行されます。このため、System Automation for Multiplatforms では、リソース A のみを開始させる場合であっても、リソース A および B の両方について同じノード位置を決定します。System Automation for Multiplatforms の内部の動作は以下のようになります。リソース A を開始する必要がある場合、System Automation for Multiplatforms ではリソース A および B の両方のノード位置を決定し、それからリソース A を開始します。

注 : リソース B の開始は、Collocated 関係によって駆動されません。 これは、他の開始/停止関係またはグループの動作によって実行されます。

予測アプローチを要約すると以下のようになります。リソース A またはリソース B の いずれかが開始され、他のリソースがオフライン状態である場合、System Automation for Multiplatforms では、そのいずれかのリソースを開始する前に、論理的にリソース A および B の両方がバインドされる ノード位置を決定します。

ノード位置の最適化は、現在の環境に基づく予測に過ぎないことに 注意してください。ノード選択決定のベースとなった前提条件は、時間の経過によって変更されることがあります。

ノード選択の誤った予測のシナリオは以下のとおりです。リソース A および B は 浮動リソースでノード 1、2、および 3 に配置させることができます。関係 A -- Collocated ---> B が定義されています。ここで、リソース B を開始する必要があります。 Collocated 関係に より、System Automation for Multiplatforms では、 リソース A および B に対して node1 が選択されます。 その後、リソース B が開始されます。しばらくすると、管理者の使用方法エラーにより、リソース A がノード 1 で開始されなくなります。ノード 1 上のリソースの OpState は FailedOffline です。 その後、要求によりリソース A の開始要求がトリガーされます。リソース A はもうノード 1 上で開始できないため、競合状態が発生します。この状態は後述する方法で解決する必要があります。

ケース IV

考えられる他の状態として、リソース B が開始されるときに、リソース A が既に 実行状態にある (OpState が、「オンライン」、「オンラインの保留中」、「オンライン中」、または「オフラインの保留中」のいずれか) とします。

図 *58. Collocated* 関係、ケース *IV*

リソース A が開始された時点で、既にリソース B についても同じノードが選択されています。何もエラーが発生しなければ、リソース B をそこで開始させることができます。先に 選択されていたノード上にリソース B の開始を妨げる問題が存在した場合は、リソースはアンバインドされ、リソース B の開始時に新しいノード位置が 検索する必要があります。これは、リソース B が別のノードで始動可能であることを意味します。

以下の条件付き関係を定義できます。

- **Collocated/IfOnline**

  関係 A ---> Collocated/IfOnline -----> B は、リソース B がオンライン状態である場合にのみロケーション関係が考慮されることを 意味します。そうでない場合、ロケーション関係は無視されます。IfOnline には、「オンラインの保留中」および「オフラインの保留中」などの状態は含まれません。

- **Collocated/IfOffline**

  関係 A ---> Collocated/IfOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態 である場合にのみロケーション関係が有効であることを意味します。

- **Collocated/IfNotOnline**

  関係 A ---> Collocated/IfNotOnline -----> B は、リソース B がオンライン状態でない場合にのみロケーション関係が 有効であることを意味します。

- **Collocated/IfNotOffline**

  関係 A ---> Collocated/IfNotOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態 でない場合にのみロケーション関係が有効であることを意味します。

- **Collocated/IfIWasOnline**

  関係 A --->Collocated/IfIWasOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインであったがオフライン状態になった場合にのみロケーション関係が有効であることを意味します。

- **Collocated/IfIWasNotOnline**

関係 A ---> Collocated/IfIWasNotOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインになっていない場合にのみロケーション関係が有効であることを意味します。

AntiCollocated 関係

System Automation for Multiplatforms では、AntiCollocated 関係を使用して、ソース・リソースおよび そのターゲット・リソースが別の ノード上に 配置されるようにします。AntiCollocated 関係により、以下の動作方式が提供されます。

図 *59. AntiCollocated* 関係

- AntiCollocated 関係は、リソース A の開始時には、リソース B が既に稼働されているノードとは別のノード上でのみリソース A を始動可能であることを定義します。

AntiCollocated 関係は、Condition 属性とともに使用できます。

ターゲットが同値の場合、並行リソースは AntiCollocated 関係のソースとして 許可されません。このような関係の結果は、 定義されていない自動化動作になります。

AntiCollocated 関係の動作
AntiCollocated 関係には、4 つの異なる状態があります。

ケース I

リソース A の開始時には、リソース B が現在稼働中のノードとは別のノードにこれを配置します。稼働中とは、リソース B の OpState が、「オンライン」、「オンラインの保留中」、「オンライン中」、または「オフラインの保留中」のいずれかであることを意味します。

図 *60. AntiCollocated* 関係、ケース *I*

この動作は、標準状態を表します。

AntiCollocated 関係では、将来の状態の予測に基づいて、ノード選択の 最適化が試行されます。以下のケースが考えられます。

ケース II

リソース B が開始され、リソース A が「オフライン」、「オフラインに失敗」、または「不明」状態の いずれかです。

図 *61. AntiCollocated* 関係、ケース *II*

一般的に、リソース B のノード選択は リソース A とは独立していることが予想されます。しかし、System Automation for Multiplatforms による リソース B のノードの選択時には、将来リソース A を別のノードで開始できるような ノードが選択されます。この予測アプローチは、将来のリソース A の 開始動作を単純化するためのものです。エラー状態が発生しない場合に、リソース B が開始された後、リソース A をリソース B が稼働していない 別のノードで開始できるようにします。これは、**ケース I** の説明に対応しています。

ケース III

リソース A が開始され、リソース B がオフライン状態 (「オフライン」、「オフラインに失敗」) です。

図 *62. AntiCollocated* 関係、ケース *III*

理論上、リソース A は現在そのノード・リストの いずれかのノードに配置可能です。ここで、再度予測アプローチにより、将来リソース B を別のノードで開始できるような リソース A のノード位置の検索が試行されます。このため、System Automation for Multiplatforms では、リソース A のみを開始させる場合でも、リソース B のノード位置を決定します。

予測アプローチを要約すると以下のようになります。リソース A が オフライン状態であり、リソース A またはリソース B のいずれかが開始される場合 (ケース II および ケース III を参照)、System Automation for Multiplatforms では、リソース A および B のいずれかを開始させる前に、この両リソースについて異なるノード位置を決定します。

Collocated 関係の説明で既に述べたように、現在の環境に基づく予測が、時間の経過によって誤りとなることがあります。それでもやはり、ほとんどの場合において予測アプローチにより自動化動作が単純化されます。

ケース IV

リソース B が開始されるときに、リソース A が既に実行状態にある (OpState が、「オンライン」、「オンラインの保留中」、「オンライン中」、または「オフラインの保留中」のいずれか) とします。

図 *63. AntiCollocated* 関係、ケース *IV*

リソース A が開始された時点で (ケース III を参照)、既にリソース B について別のノードが選択されています。何もエラーが発生しなければ、リソース B をそこで開始させることができます。前に選択されていたノード上でリソース B を開始できなくなるような問題が 発生した場合は、リソース B の開始時に新しいノード位置が 検索されます。これは、リソース B を、リソース A が既に稼働中の場所も含め、任意の場所で開始できることを意味します。

以下の条件付き関係を定義できます。

- **AntiCollocated/IfOnline**

  関係 A ---> AntiCollocated/IfOnline -----> B は、リソース B がオンライン状態である場合にのみロケーション関係が 有効であることを意味します。そうでない場合、ロケーション関係は無視されます。IfOnline には、「オンラインの保留中」および「オフラインの保留中」などの状態は含まれません。

- **AntiCollocated/IfOffline**

  関係 A ---> AntiCollocated/IfOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態 である場合にのみロケーション関係が有効であることを意味します。

- **AntiCollocated/IfNotOnline**

  関係 A ---> AntiCollocated/IfNotOnline -----> B は、リソース B がオンライン状態でない場合にのみ ロケーション関係が有効であることを意味します。

- **AntiCollocated/IfNotOffline**

  関係 A ---> AntiCollocated/IfNotOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態でない場合にのみロケーション関係が有効であることを 意味します。

- **AntiCollocated/IfIWasOnline**

  関係 A ---> AntiCollocated/IfIWasOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインであったがオフライン状態になった場合にのみロケーション関係が有効であることを意味します。

- **AntiCollocated/IfIWasNotOnline**

関係 A ---> AntiCollocated/IfIWasNotOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインになっていない場合にのみロケーション関係が有効であることを意味します。

Affinity 関係

Affinity 関係により、以下の動作方式が提供されます。

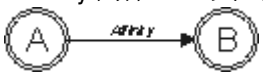


図 *64. Affinity* 関係

- Affinity 関係は、リソース A の開始時には、可能な場合はリソース B が既に稼働中のノードが選択されることを定義します。他の ロケーション関係によりこれが禁止されている場合、リソース A は別のノードでも 稼働可能です。

Affinity 関係は、Collocated 関係と非常によく似ています。したがって、Affinity 関係は柔らかいロケーション関係を、その一方で Collocated 関係は 強固なロケーション関係を定義します。

Affinity 関係は、31 ページの『Condition 属性』とともに使用できます。

以下の条件付き関係を定義できます。

- **Affinity/IfOnline**

  関係 A ---> Affinity/IfOnline -----> B は、リソース B がオンライン状態である場合にのみロケーション関係が考慮されることを 意味します。そうでない場合、ロケーション関係は無視されます。IfOnline には、「オンラインの保留中」および「オフラインの保留中」などの状態は含まれません。

- **Affinity/IfOffline**

  関係 A ---> Affinity/IfOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態である場合にのみロケーション関係が有効であることを意味します。

- **Affinity/IfNotOnline**

  関係 A ---> Affinity/IfNotOnline -----> B は、リソース B がオンライン状態でない場合にのみロケーション関係が 有効であることを意味します。

- **Affinity/IfNotOffline**

  関係 A ---> Affinity/IfNotOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、または「不明」状態でない場合にのみロケーション関係が有効であることを意味します。

- **Affinity/IfIWasOnline**

  関係 A --->Affinity/IfIWasOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインであったがオフライン状態になった場合にのみロケーション関係が有効であることを意味します。

- **Affinity/IfIWasNotOnline**

  関係 A --->Affinity/IfIWasNotOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインになっていない場合にのみロケーション関係が有効であることを意味します。

AntiAffinity 関係

AntiAffinity 関係により、以下の動作方式が提供されます。

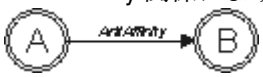


図 *65. AntiAffinity* 関係

- AntiAffinity 関係は、リソース A の開始時には、可能な場合はリソース B が既に 稼働中のノードとは別のノードが選択されることを定義します。他のロケーション関係によりこれが禁止されている場合、リソース A は同じノードでも 稼働可能です。

AntiAffinity 関係は、AntiCollocated 関係と非常によく似ています。したがって、AntiAffinity 関係は柔らかいロケーション関係を、その一方で AntiCollocated 関係は強固なロケーション関係を定義します。

AntiAffinity 関係は、31 ページの『Condition 属性』とともに使用できます。

44 ページの『ロケーション関係』 も参照してください。

以下の条件付き関係を定義できます。

- **AntiAffinity/IfOnline**

  関係 A ---> AntiAffinity/IfOnline -----> B は、リソース B がオンライン状態である場合にのみ ロケーション関係が有効であることを意味します。そうでない場合、ロケーション関係は無視されます。IfOnline には、「オンラインの保留中」および「オフラインの保留中」などの状態は含まれません。

- **AntiAffinity/IfOffline**

  関係 A ---> AntiAffinity/IfOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、 または「不明」状態である場合にのみロケーション関係が有効であることを意味します。

- **AntiAffinity/IfNotOnline**

  関係 A ---> AntiAffinity/IfNotOnline -----> B は、リソース B がオンライン状態でない場合にのみ ロケーション関係が有効であることを意味します。

- **AntiAffinity/IfNotOffline**

  関係 A ---> AntiAffinity/IfNotOffline -----> B は、リソース B が「オフライン」、「オフラインに失敗」、 または「不明」状態でない場合にのみロケーション関係が有効であることを意味します。

- **AntiAffinity/IfIWasOnline**

  関係 A --->AntiAffinity/IfIWasOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインであったがオフライン状態になった場合にのみロケーション関係が有効であることを意味します。

- **AntiAffinity/IfIWasNotOnline**

  関係 A --->AntiAffinity/IfIWasNotOnline -----> B は、リソース B がオンライン状態であり、リソース A が前にオンラインになっていない場合にのみロケーション関係が有効であることを意味します。

IsStartable 関係

IsStartable 関係により、以下の動作方式が提供されます。

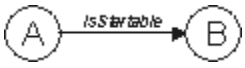


図 *66. IsStartable* 関係

- IsStartable 関係は、リソース A および B の公称状態がオンラインの場合に、リソース B が始動可能なノードにのみリソース A を配置できることを 定義します。

IsStartable は、後でターゲット・リソースが実際に始動可能であることを意味しません。これは、後の時点において、リソースの障害により そのすべての関係が解決できなくなる可能性があるためです。

44 ページの『ロケーション関係』 も参照してください。

IsStartable 関係の動作

IsStartable 関係があると、以下の動作が行われます。 IsStartable 関係は、ターゲット・リソースが始動可能な ノードにのみソース・リソースが配置可能であることを定義します。この関係は、ソース・リソースおよびターゲット・リソースの公称状態がオンラインの場合にのみ考慮されます。1 つのリソース (ソースまたはターゲット) の公称状態がオンラインでない 場合、IsStartable 関係は、公称状態がオフラインの リソースとともに廃棄されます。

以下の例で、IsStartable 関係の動作を説明します。リソース A およびリソース B は浮動リソースで、同じリソース・グループ RG_A の メンバーです。リソース A は node1 および node2 で、リソース B は node2 および node3 で 稼働可能です。リソース A からリソース B に対して、IsStartable 関係が定義されて います。

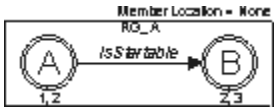


図 *67. IsStartable* 関係、詳細パート *I*

リソース・グループの公称状態がオンラインに設定されると、両方のメンバーが開始されます。IsStartable 関係に基づいて、リソース A およびリソース B は、両リソースに共通のノード node2 で開始されます。リソース B がノード 2 で「オフラインに失敗」状態になると、リソース・グループ RG_A を開始してもリソース A は開始されません。これは、リソース A およびリソース B の両方を開始できるノードが存在しないためです。

以下の例で、IsStartable 関係の詳細情報を示します。このシナリオにおいては、リソース A は node1、node2、および node3 で稼働可能であり、リソース・グループ RG_A のメンバーです。リソース B は node1 および node2 で稼働可能であり、リソース・グループ RG_B のメンバーです。リソース A からリソース B に対する IsStartable 関係が 定義されています。

図 *68. IsStartable* 関係、詳細パート *II*

以下に、この例において考えられる状態を説明します。

- RG_A の公称状態がオンラインに設定され、RG_B はオフラインです。IsStartable 関係は、ソース・リソースおよびターゲット・リソースの公称状態が オンラインである場合 (ここでは RG_A および RG_B) にのみ考慮され、この場合 RG_B の公称状態はオフラインであるため、関係は無視されます。したがって、リソース A は node1、node2、または node3 のいずれでも開始できます。
- RG_A の公称状態がオンラインに設定され、RG_B は既にオンラインです。 この場合、IsStartable 関係は考慮に入れられ、System Automation for Multiplatforms は、リソース B を始動可能な ノード (node1 または node2) でリソース A を開始します。
- 問題のためにリソース B を node1 および node2 で開始できず、RG_B の 公称状態がオンラインです。リソース・グループ RG_A を開始すると、両リソースに共通の node1 および node2 でリソース B を開始できないため、リソース A はオンラインになることができません。
- 問題のためにリソース B を node1 および node2 で開始できず、RG_B の 公称状態がオフラインです。リソース・グループ RG_A の公称状態が オンラインに設定されると、リソース・グループ RG_B の本来あるべき状態はオフライン であるため、System Automation for Multiplatforms はリソース B を廃棄し、IsStartable 関係は無視されます。

同値

以下のトピックで、同値とその使用法について説明します。

52 ページの表 9 は、同値に使用する System Automation for Multiplatforms コマンドのリストです。 詳しくは、「*Tivoli System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

表 *9.* 同値とともに使用する *System Automation for Multiplatforms* コマンド

コマンド	説明
mkequ	同値リソースを作成する
rmequ	同値リソースを除去する
chequ	同値リソースを変更する
lsequ	同値リソースをリストする

同値とは

同値は、同じ機能を提供するリソースの集合です。同値は、同じリソース・クラスの固定リソースのセットで 構成されます。

例えば、ネットワーク・アダプターは同値として定義できます。1 つの ネットワーク・アダプターがオフラインになると、別のネットワーク・アダプターがオフラインの アダプターから処理を引き継ぐことができます。

同値はまた、リソース・グループのノード・リストを設定するためにも 使用されます。この同値から、1 つ以上のリソースを選択して、管理対象関係を 満たすことができます。ただし、管理対象関係を満たすためにノード上で開始できるのは、1 人のメンバーのみです。管理対象関係について詳細は、29 ページの『管理対象関係』を参照してください。

同値には、2 つのタイプがあります。

1. 静的メンバーシップ・リスト付きの同値。 このタイプの同値には、ユーザーが同値に明示的に追加したリソースの固定セットが含まれます。
2. SelectString リスト付き同値。 同値内に含まれるリソースを、実行時にこのタイプが動的に判別する場合の同値。 動的な選択文字列と一致する RMC リソースが作成されると、そのリソースは自動的に同値に含まれます。リソースが順序付けられないため、このタイプの同値に関してポリシーの指定は意味がありません。

以下のタイプの同値を使用して、特定の自動化動作を有効にできます:

Failback
順序付けられたポリシーと組み合わせてのみ使用されます (例えば、順序付けられたビット・セットを備えた SelectFromPolicy 属性)。このポリシーの同値は、以下の動作を示します。

System Automation for Multiplatforms の場合は、同値のすべてのリソースが、必ず同値の最初のメンバーをホスティングするノードで実行します。 このメンバーが「オフライン」になると、System Automation for Multiplatforms は、同値の 2 番目のメンバーをホスティングするノード上のリソースを開始します。 最初のメンバーが「オンライン」に戻ると、System Automation for Multiplatforms は、すべての従属リソースを停止し、それをもう一度、同値の最初のメンバーをホスティングするノードで再始動します。

タイプ Failback の同値と関係を持つリソース・グループに対する移動要求の結果は、予期した動作にはならないことに注意してください。移動は起こりますが、移動直後に完了し、リソース・グループは元のノードにフェイルバックします。

NoFailure
このタイプの同値は、以下の動作を示します。

同値への依存関係を持つリソースは、タイムアウト間隔内に「オンライン」操作状態に到達しない場合は、「オフラインに失敗」状態に設定されません。 これは、開始に時間のかかるリソースの場合は役立ちます。 その意味は、リソースの StartCommandTimeout が無限大に設定された場合に等しく、しかし StartCommandTimeout 値が非常に高いリソースは、待ち状態にあるために、StartCommandTimeout に到達するまで自動化できないのに対し、これらのリソースは「オンライン」タイムアウトの到達後もう一度自動化できる点が異なります。

Shadow Resources/Shadow 同値
シャドー・リソースは、別のリソースの OpState をモニターします。 System Automation for Multiplatforms のみが、シャドー・リソースの OpState を評価しますが、その開始も停止も行いません。 シャドー・リソースは、浮動リソースから、異なるノードで実行する 2 つ以上の固定リソースのいずれかへの関係を定義する際に使用されます。 詳しくは、143 ページの『シャドー・リソースの使用』を参照してください。

同値は、System Automation for Multiplatforms のリソース・クラス IBM.Equivalency で提供されます。

同値の使用に関するルール

以下に、同値を使用するためのルールを示します。

1. リソースは、同値またはリソース・グループのいずれかのメンバーにすることができますが、両方のメンバーにすることは できません。
2. 1 つのリソースを複数の同値に含めることができます。
3. 指定するリソースは、すべて同じリソース・クラスのものでなければなりません。
4. 同値を同値のメンバーにすることはできません。

5. リソース・グループを同値のメンバーにすることはできません。
6. 同値をリソース・グループのメンバーにすることはできません。
7. アクティブ・リソースの関係を満たす同値を変更することは できません。
8. 同値は、管理対象関係のターゲット にのみすることができます (管理対象関係のソース にすることは できません)。
9. 同値のメンバーは、固定リソースでなければなりません。浮動リソースは許可 されません。

同値によって使用される属性

名前

Equivalency の名前。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

MinimumNecessary

同値を使用可能にするために必要な最小メンバー数を指定します。 選択文字列が一致すると、メンバーシップが動的に追加されます。

Subscription

System Automation Application Manager 用のプラグインである System Automation for Multiplatforms アダプターをサポートします。任意の同値に対するサブスクリプションのマッピングが許可されます。

MemberClass

System Automation for Multiplatforms では、MemberClass 永続属性を使用して、すべてのメンバー・リソースの リソース・クラスを決定します。

Membership

System Automation for Multiplatforms は、Membership 永続属性を使用して、同値に含まれる リソースのセットを決定します。Membership 属性が指定されている場合は、SelectString 属性は指定できません。

SelectString

System Automation for Multiplatforms では、SelectString 永続属性を使用して、同値に含める リソースを動的に決定します。選択文字列と一致するリソースが システムに対して挿入または、システムから除去されると、System Automation for Multiplatforms によって同値が 自動的に変更されます。SelectString 属性が指定されている場合は、Membership 属性は指定できません。

SelectFromPolicy

System Automation for Multiplatforms では、SelectFromPolicy 永続属性を使用して、同値からの選択を作成する際に使用するポリシーを決定します。この属性は、同値を参照するリソースがオフラインのときに変更できます。 可能性があるポリシー:

Any (デフォルト)
この場合、同値に含まれるリソースが障害になると、System Automation for Multiplatforms は、事前指定された 選択の順序を参照することなく任意のリソースを選択します。

Ordered
この場合、同値に含まれるリソースが障害になると、System Automation for Multiplatforms は常に 選択リストの先頭から開始します。

注 : 「Ordered」ポリシーは、動的な SelectString が使用される場合は使用できません。

SelectFromPolicy 永続属性には追加の設定値があります。これは、Any または Ordered 設定値と結合して、特定の自動化動作を有効にできます。

Failback
Ordered ポリシーに指定できるのは Failback オプションのみです。 このオプションが指定されると、System Automation for Multiplatforms は、最初のメンバーが再度使用可能になったときに、リソースを、同値の最初のメンバーをホスティングするノードにフェイルバックします。

この機能はリソース・グループに対する移動要求に影響することに注意してください (167 ページの『リソース・グループの移動』も参照)。Failback ポリシーによる同値が移動要求に含まれている場合、リソース・グループの移動は行われますが、リソースがターゲット・ノードで「オンライン」になった直後に、同値の Failback オプションによって、リソース・グループは元のノードに戻されます。

NoFailure
NoFailure オプションが指定されると、同値への DependsOn 関係を持つリソースは、これらのリソースに対する「オンライン」要求が「オンライン」タイムアウト間隔内に成功しない場合は、「オフラインに失敗」に設定されません。このことは、これらのリソースが別のノードにフェイルオーバーされるのは、これらのリソースの MonitorCommand が「オフラインに失敗」に戻る場合のみであることを暗黙に示しています。

NoControl
NoControl オプションは、同値のメンバーをシャドー・リソースと識別します。 この場合、System Automation for Multiplatforms は、同値のメンバーを開始または停止しませんが、同値リソース・メンバーの OpState の変更に対してのみ反応します。デフォルトでは、System Automation for Multiplatforms は、例えばクラス IBM.Application の同値内の制御可能なリソースの開始および停止を行います。

同値のメンバーによって使用される属性

- 同値に追加するリソースは、以下の属性を保持している必要があります。
  - OpState
- 同値に追加するリソースは、以下の属性を保持することができます。
  - NodeNameList
  - ResourceType

これらの属性についての詳細は、14 ページの『リソースによって使用される属性』を参照してください。

リソースの自動化

System Automation for Multiplatforms は目標駆動型自動化の概念を提供します。自動化ポリシーの中で目標を定義します。

このタスクについて

自動化ポリシーを作成することで、クリティカル・リソースの目標を設定します。目標は、例えば「リソース A は常にオンラインでなければならない」などです。

以下の用語を使用してリソースの状態を指し示します。

NominalState
リソースの本来あるべき状態を規定します。

OpState
リソースの実際の状態を提供します。MonitorCommand の戻りコードによって戻されます。

System Automation for Multiplatforms は、IBM.Application クラスのリソースを管理するコマンドと連携して作業を行います。例えば、StartCommand はリソースを開始し、StopCommand はリソースを停止し 、MonitorCommand はリソースの状況をモニターします。リソースは、IBM.Application、IBM.ServiceIP、IBM.AgFileSystem など、別のクラスのものであっても構いません。このトピックでは、IBM.Application リソース・クラスを対象とします。

MonitorCommand は System Automation for Multiplatforms によって定期的に実行されます。リソースの状況は MonitorCommand の戻りコードに反映され、System Automation for Multiplatforms により、そのアプリケーションの OpState にマップされます。NominalState と OpState は、互いに比較されます。それらが同一であるか否かに応じて、OpState と NominalState の値が同一になるように、アプリケーションに対して StartCommand または StopCommand が実行されます。

ノード位置の割り当て

このタスクについて

バインド・プログラム は、System Automation for Multiplatforms がまだノード位置を割り当てていないリソースを 開始することが必要になった場合に常に呼び出されます。ノード位置が割り当てられているリソースは、バインド済み とも呼ばれます。例として、複数のノードで稼働できる浮動リソースのバインディングについて説明します。ここでは、バインディング・アルゴリズムはこの浮動リソースの すべてのロケーション関係を考慮して、そのノード位置を決定 (バインド) する 必要があります。リソースを既にバインドした、前回のバインディング・アルゴリズム実行に基づいて、リソース A のロケーションの制限を含めて、新しいソリューションが検索される必要があります。バインディング・ソリューションは、必ず明瞭である必要があるわけではありません。さまざまな代替ソリューションがあり、System Automation for Multiplatforms に任意に選択されます。

あいまいなシナリオの例として、node1 および node2 で稼働可能な 2 つの 浮動リソース A および B が含まれる Collocated ロケーション関係を持つ リソース・グループを挙げます。グループが開始されると、以下の 2 つの代替ソリューションが 選択できます。A および B が node1 にバインドされるか、両方が node2 にバインドされるか、のいずれかです。

バインディング・アルゴリズムが、関連するすべてのリソースのノード配置についてソリューションを 見つけることができると、リソース (1 つ以上) が開始されます。ただし、当然ながら、ロケーション関係は解決する必要のある競合になる場合があります。

競合解決の例として、2 つの浮動リソース A および B は、node1 および node2 に配置できるが、パフォーマンス上の制約のために同じノードで並行して実行することができないことがあります。 これを実行するには、A から B および B から A への AntiCollocated 関係を指定します。ここでは、ノード 2 が失敗したときに、リソース A は既にノード 1 で稼働中であると想定します。ここでユーザーがリソース B を開始すると、複数のリソースを同一ノード上で開始できないため、ロケーション関係の競合が発生します。この状況では、両方のリソースの実行を可能にするような、この問題への完璧な解決策はありません。このような競合を解決するために、System Automation for Multiplatforms はいわゆる「廃棄ステップ」を実行します。

既にオンラインであるリソースが、問題の一部となっている可能性があります。これらのリソースについては、リソース・グループによって、その優先順位として追加の優先順位ボーナス 10 が設定されます。Priority 属性とその影響の詳細については、20 ページの『リソース・グループによって使用される属性』 および 24 ページの『Priority 属性』 を参照してください。

以下のセクションで、System Automation for Multiplatforms によるロケーション関係の検索とその競合の解決処理のソリューションについて詳細に説明します。この処理全体を バインディング・アルゴリズムと呼びます。

バインディング・アルゴリズムは、以下に示す複数のステップで構成されます。

1. **ディスカバリー・ステップ: バインディング問題を独立して解決できる 構成サブセットを決定する。**

   ディスカバリー・アルゴリズムは、以下に示す複数のサブステップで構成されます。

   a. **すべての関連リソースを検索する (構成サブセット)**

      ロケーション関係により、利用者の構成は、独立して解決可能な複数の 構成サブセットに分類されます。これは、多くの場合、ロケーション関係は構成のリソースのサブセットにのみ影響を 与えるためです。一例は、A --> Collocated --> B、B --> Collocated --> C、および D --> Collocated --> E という構成です。ここで、A、B、および C のロケーション関係は、D および E から独立して解決できます。これらの 2 つのサブセットの場合、以下のすべてのステップが個別に実行されます。

   b. **OpState が「オフラインに失敗」のすべてのリソースを無視する**

OpState が「オフラインに失敗」であるすべてのインスタンスは、バインディング・ソリューションに寄与しないことは明白です。これらのインスタンスは、バインディング・ソリューションを 検索するために使用される構成サブセットから除去されます。この例として、node1 および node2 で稼働可能な 2 つの浮動リソース A および B が含まれる リソース・グループ R1 を挙げます。このグループには、Collocated パラメーターが設定されて います。これはリソース A および B が同じノード上で開始される必要があることを意味します。node2 が中断し、これが原因で node2 上の浮動リソース A および B の構成要素が 「オフラインに失敗」状態になると想定します。このため、node2 上のインスタンスはバインディングの問題の解決に役立たないことから、これらが 構成サブセットから除去されます。

c. **開始できないリソース・グループを クリーンアップする**

必須リソース・グループ・メンバーが「オフラインに失敗」状態である 場合、リソース・グループの動作に従って、リソース・グループを開始することは できません。このため、このようなリソース・グループを除いたすべてのリソース・グループ・メンバーを 停止する必要があります。

例として、上記で説明した、浮動リソース A および B を含む リソース・グループ R1 を挙げます。アプリケーション・エラーのために、いずれかのノードで浮動リソース A を開始できない場合で、この開始できない浮動リソース A が必須リソース・グループ・メンバーである場合は、 浮動リソース B も停止します。詳しくは、27 ページの『リソース・グループ・メンバーに対して使用される属性』を参照してください。

2. **完全なソリューション・ステップ:「完全な」ソリューションの検索を試行する**

まず、リカバリー・リソース・マネージャーは、構成サブセットの 関連するすべてのロケーション関係の完全なソリューションを検索しようとします。この ステップでは、44 ページの『ロケーション関係』で説明するようなバインディングの検索を試みます。この 最初のステップの目的は完全なソリューションを検索することであるため、すべての Affinity 関係 および AntiAffinity 関係は、純粋な Collocated 関係および AntiCollocated 関係であるかのように 扱われます。また、「オフライン」であり開始対象でないリソースさえも、必要に応じてバインドが試行されます。ロケーション関係の 競合が発生しない場合、必要なリソースがバインドされ、バインディング・アルゴリズムが完了します。次のステップとして、System Automation for Multiplatforms は、開始する必要のあるこれらのリソースを開始できます。

このバインディング・ステップが、解決できない矛盾する制約を持つ競合状態を引き起こす 場合があります。これを解決するために System Automation for Multiplatforms は、以下で説明する複数の サブステップで構成される廃棄ステップを提供します。

3. **廃棄ステップ: 競合するロケーション関係が存在する状態を 解決する**

廃棄ステップは、以下に示す複数のサブステップで構成されます。

a. **すべての Affinity 関係および AntiAffinity 関係を 無視する**

競合状態を解決するための最初のアプローチは、すべての Affinity 関係および AntiAffinity 関係を無視することです。これらは「柔らかい」ロケーション関係 であるためです。System Automation for Multiplatforms は、前のバインディングに基づいて、バインドする必要がある リソースのソリューションの検索を試みます。すべての Affinity 関係 および AntiAffinity 関係が無視されるため、ロケーション関係が単純化され、バインディング・ソリューションが見つかる可能性が高くなります。ソリューションが見つかった場合、廃棄ステップは終了します。しかし、まだ 競合状態が解決できない可能性があります。その場合、廃棄ステップの次のレベルに到達します。

b. **OpState が「オフライン」で、開始する必要のない すべてのリソースを無視する**

すべての Affinity 関係および AntiAffinity 関係を 無視してもバインディングの問題のソリューションが見つからなかった場合 (ステップ 57 ページの『3.a』 を参照)、次のレベルとして、「オフライン」であり現在 開始する対象でないすべてのリソースをバインディングの評価において 無視します。これにより、バインディング・ソリューションが見つかる可能性が 高くなります。

例として、浮動リソース A を含むリソース・グループ R1、浮動リソース B を 含むリソース・グループ R2 があり、A から B に対して AntiCollocated 関係が定義されていると します。浮動リソース A および B は node1 および node2 で稼働可能ですが、node2 が 中断されました。

図 69. 浮動リソースを含むリソース・グループの例

ここで、R1 の公称状態がオンラインに 設定され、それが原因でリソース A を開始するには、その前にバインドすることが必要になります。System Automation for Multiplatforms は、まず完全なソリューションを見つけようとします。このため、A および B のバインドが 試行されます。ここでは、ソリューションは見つかりません。次に System Automation for Multiplatforms は、すべての Affinity 関係 および AntiAffinity 関係を無視しますが、これでもソリューションは見つかりません。続いて「オフライン」状態で、開始する必要のないすべてのリソースが 無視されます。これにより、リソース B が評価において無視されます。これで、リソース A を node1 にバインドすることが可能になります。

使用可能なバインディング・ソリューションが見つかった場合、 廃棄 ステップは終了します。見つからない場合は、廃棄プロセスの次のステップに到達します。

c. **最も重要でないリソース・グループ・メンバーを停止する**

廃棄ステップの次のレベルは、リソース・グループ・メンバーを停止し、それらのメンバーをバインディングの評価時に無視することです。各リソース・グループには、優先順位の値が割り当てられているので、まず優先順位の値の最も低いグループのリソース・グループ・メンバーが停止され、それらを除いてバインディング・ソリューションの検索が試行されます。これによりバインディングの制約が満たされない場合は、次のレベルの優先順位を持つグループのリソース・グループ・メンバーが選択されます。

非必須リソースは、以下の順序で、バインディング・セットから削除されます。

i) 開始できないリソースが、最初に削除されます。

ii) 削除できるリソースがない場合、次に削除できるのは、現在オフラインのリソースであると見なされます。

iii) それでも削除できるリソースがない場合は、非必須リソースがすべて削除されます。

iv) リソース・グループ自体が非必須メンバーである場合は、必須リソースを含むすべてのリソースが削除されます。

優先順位のスキーマに加えて、リソース・グループを停止および除去するために、以下の 2 つのサブステップが実行されます。

i) 特定の優先順位のグループのすべての非必須メンバーが停止され、バインディング・ソリューションで無視されます。グループが別のグループの非必須メンバーである場合は、必須メンバーは明示的に破棄されます。この追加ステップは、SAP ポリシーのようなより複雑なポリシーを編成するのに役立ちます。また、バインディングの問題のソリューションが見つかるまでリソースを停止する必要がある状況 (例えば、SAP エンキュー・サーバーで障害が起こった場合) に対処できるようにします。

ii) これでもまだ競合が存在する場合、前述のように、次に優先順位の低いグループのメンバーが停止されます。

これらの 2 つのサブステップは、バインディング・ソリューションが見つかるまで、またはバインディングの問題が解決不能として見なされるまで繰り返し実行されます。

4. **すべてのリソースの停止**

バインディングの問題がまだ未解決の場合は、バインディングの問題に関係するリソースはすべて停止され、これらのリソースの停止中に新たに解決が試みられます。

ヒント:

- 外部グループおよび内部グループは、サブグループ化に関して定義されます。内部グループは外部グループに含まれます。外部グループの優先順位は、内部グループと同じか、高くなければなりません。 この条件が満たされていないと、内部グループより前に外部グループが廃棄されます。ただし、外部グループが廃棄された場合、内部グループは自動的に停止します。

- 別のリソース・グループの非必須メンバーであるリソース・グループの優先順位は、同じリソース・グループの必須メンバーであるリソース・グループより低くなければなりません。 この条件が満たされていないと、必須メンバーが廃棄されることがあります。
- 優先順位の処理の詳細については、20 ページの『リソース・グループによって使用される属性』および24 ページの『Priority 属性』を参照してください。
- 同値に依存しており、始動可能メンバーも既に開始されたメンバーも存在しないリソースまたはリソース・グループの場合、本来あるべき状態はオフラインです。 公称状態や開始要求とは関係ありません。したがって、これらの リソースおよびリソース・グループは、57 ページの『3.b』で廃棄されます。
- 単一の集約点および制御点を提供するためにネスト・グループを採用する場合は、サブグループを非必須にすることを検討してください。 これにより、バインディング・アルゴリズムによる各サブグループの廃棄が許可され、バインディングの問題の制約が除去されます。アルゴリズムのこの部分は、緩和フェーズと呼ばれます。グループは廃棄されると、開始しないか、または明示的に停止します。
- 同値に依存しており、始動可能メンバーも既に開始されたメンバーも存在しないリソースまたはリソース・グループの場合、本来あるべき状態はオフラインです (公称状態や開始要求とは関係ありません)。 したがって、これらの リソースおよびリソース・グループは、57 ページの『3.b』において廃棄されます。
- メンバー・リソースではなく、リソース・グループ間で依存関係が確立される場合は、開始要求の伝搬によって、従属するリソース・グループの本来あるべき状態がオンラインになります。
  - これにより、上記で説明した廃棄に関する動作が変わるため、場合によっては関係のチェーン内でリソースの障害が回復されません。
  - この依存関係のチェーンのために、障害を回復するには、リソース・グループ間の関係においてリソースの優先順位を調整して、リソースを廃棄するための適切な順序を設定する必要があります。
  - 経験法則からすると、関係のチェーン (A->B->C) を使用する場合、従属するリソース・グループには、サポート・グループより 11 高い優先順位を割り当てる必要があります。155 ページの『開始要求および停止要求のスコープ』も参照してください。

自動化動作

このセクションのトピックでは、System Automation for Multiplatforms の自動化動作について説明します。

このタスクについて

可用性に関連するイベント

NominalState 属性がオンラインであるすべての最上位リソース・グループが自動化 されます。これは、リソース・グループ内の管理対象リソースの管理対象関係を満たすことができる場合に、これらのグループ内のそのような最上位リソース・グループおよび管理対象リソースの開始が試行されることを意味します。

リソース・グループまたは管理対象リソースをオンラインにすることができない場合 (その 1 つ以上の メンバー・リソースが必要な状態であるオンラインに到達することに完全に失敗した 場合)、リソース・グループは「オフライン」または「オフラインに失敗」状態になります。System Automation for Multiplatforms に対して リソース・グループの開始の再試行が必要なことを通知するイベントが発生するまで、このリソース・グループはオフラインのままになります。

以下に、オフラインのリソース・グループをオンラインにする可能性がある イベントを示します。

- リソース・グループの `AllowedNodes` 属性 (21 ページの『AllowedNode 属性』で説明) を、例えばリソース・グループを開始できる追加のノードを含めるなどして変更します。詳細については、「*System Automation for Multiplatforms* リファレンス・ガイド」の **chrg** コマンドの説明を参照してください。
- リソース・グループから管理対象リソースを除去します。その結果、開始できないリソースが除去されるために、その他のメンバー・リソースが 始動可能になる可能性があります。詳細については、「*System Automation for Multiplatforms* リファレンス・ガイド」の **rmrgmbr** コマンドの説明を参照してください。
- 管理対象関係のターゲット・リソースであるが、現在リソース・グループの メンバーでないリソース・グループにリソースを追加します。これはその後 System Automation for Multiplatforms によって自動化されます。その結果、管理対象関係が満たされるため、その他のメンバー・リソースが始動可能になる

可能性があります。詳細については、「*System Automation for Multiplatforms* リファレンス・ガイド」の **addrgmbr** コマンドの説明を参照してください。

- System Automation for Multiplatforms が制御できず、リソース・グループ・メンバーに対して管理対象関係を 保持するリソースを開始します。その結果、リソースはオンラインになり、管理対象関係が満たされます。これにより、リソース・グループも オンラインになる可能性があります。リソースを開始するには、RMC の **startrsrc** コマンドを使用します (詳細はこのコマンドのマニュアル・ページを参照してください)。
- 集合体に構成要素を追加して、より多くのノードにおいて集合リソースを 使用可能にし、結果として System Automation for Multiplatforms がすべての管理対象関係の制約を 満たすことができるようにします。構成要素が 1 つの ハードウェアである場合は、ハードウェアをインストールするか、これを正しく定義する必要があります。リソースが浮動リソースの場合は、NodeNameList 属性にノード名を追加する ことにより構成要素を追加します。詳細は RMC の資料およびマニュアル・ページを 参照してください。
- 動的選択文字列を使用する同値によって、新規リソースが検索されます。結果として、このリソースが同値に追加され、この同値に対する 管理対象関係が解決される可能性があります。
- 管理対象リソースを NotMandatory にします。これによりこのリソースを犠牲にする ことができます。結果として、その他の管理対象リソースを開始できます。これは、それ自体がグループ・メンバーであるリソース・グループにも適用されます。詳細については、「*System Automation for Multiplatforms* リファレンス・ガイド」の **chrgmbr** コマンドの説明を参照してください。
- 障害が修正された後、集合またはその構成要素の 1 つについて リセットを実行します。結果として、リソースがオフラインになり、その後 System Automation for Multiplatforms によって 開始できます。詳細は RMC の **resetrsrc** コマンドのマニュアル・ページを参照してください。
- オフライン状態であったノードがオンラインになります。その結果、System Automation for Multiplatforms は リソース・グループをオンラインにすることができる可能性があります。
- リソース・グループの優先順位属性を変更します (24 ページの『Priority 属性』で説明)。他のリソース・グループとの優先順位の競合のために、リソース・グループが始動可能でない可能性があります。この場合、開始するグループの優先順位を高くするか、その他のグループの優先順位を低くします。 この変更は、次にグループを開始すると有効になります。
- 優先順位が低い リソース・グループの開始を抑制する、優先順位が高いリソース・グループを停止します。その結果、管理対象関係の競合が 回避されます。詳細については、「*System Automation for Multiplatforms* リファレンス・ガイド」の **chrg** コマンドの説明を参照してください。

リソース・グループが現在その NominalState 値のとおりである場合、以下のイベントにより追加の自動化アクションが発生する可能性があります。

1. NominalState 属性値をオフラインからオンラインに変更する。
2. NominalState 属性値をオンラインからオフラインに変更する。

リソース・グループの状態サイクル

61 ページの図 70 に、System Automation for Multiplatforms がリソース・グループを正常に開始および停止したときに発生する状態遷移を示します。

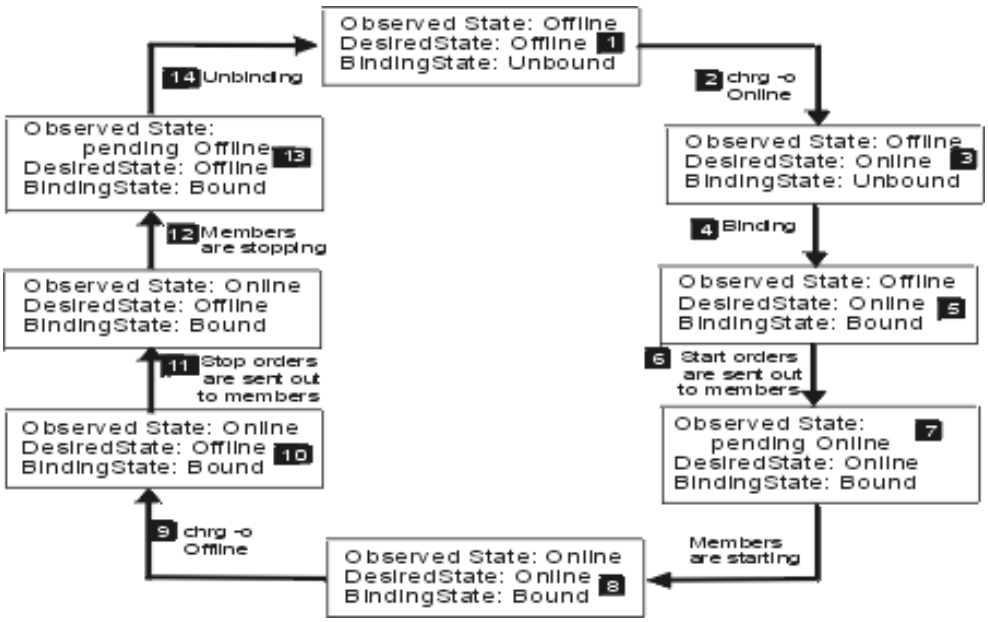


図 *70.* リソース・グループの状態サイクル

サイクルの開始では (1)、リソース・グループの DesiredState は「オフライン」、その ObservedState は「オフライン」、およびリソース・グループのメンバーは結合が解放され、Unbound の BindingState となっています。

リソース・グループの NominalState が「オンライン」に変更されると (2)、リソース・グループの DesiredState も「オンライン」に変わります。 BindingState は依然 Unbound のままです (3)。

このバインディング・ステップ (4) で、バインド・プログラムは、自動化ポリシーで定義されたすべての依存関係を満たすリソース・グループのすべてのメンバーの配置の検索を試みます。 バインディング・ステップが正常な場合、リソース・グループの BindingState は Bound に変わり (5)、そうでない場合は、リソース・グループの BindingState は結局 Sacrificed 状態となり、リソースは開始されません。

すべてのリソース・グループ・メンバーが結合されると、System Automation には、それらを開始するノードが分かります。 論理デックは、リソース・グループのメンバーの開始順序を送出し (6)、自動化ポリシーで定義されたすべての依存関係が満たされているか確認します。 リソースが開始する間に、リソース・グループの ObservedState は「オンラインの保留中」に変わります (7)。

すべてのリソースが「オンライン」のとき、リソース・グループの ObservedState は「オンライン」に変わります (8)。

リソース・グループの NominalState が「オフライン」に変更されると (9)、リソース・グループの DesiredState は「オフライン」に変わります (10)。 この結果 System Automation が起動され、リソース・グループのすべてのメンバーを停止します。 論理デックは、リソースの停止順序を送出し (11)、自動化ポリシーで定義されたすべての依存関係が満たされているか確認します。 リソースの停止中、ObservedState は「オフラインの保留中」に変わります (12)。 すべてのリソースが「オフライン」のとき、リソース・グループの ObservedState は「オフライン」に変わります (13)。

アンバインディング・ステップ (14) において、リソース・グループ・メンバーのノードの配置は内部テーブルから削除され、リソース・グループの BindingState は Unbound に変わります (1)。

引き継ぎ条件

グローバル・リソース・マネージャー

StartCommand の異常終了 (RC=非ゼロ)
オンライン・リソースを停止する StopCommand が発行されると、IBM Tivoli System Automation for Multiplatforms は引き継ぎを実行します。異常終了の直後に MonitorCommand が System Automation for Multiplatforms に RC=1(オンライン) を返すと、引き継ぎは行われません。

オンライン処理のタイムアウト
System Automation for Multiplatforms がリソースの開始を試行し、既に複数回失敗していると、RetryCount の 制限に達します。オンライン発注が再度 タイムアウトになると、引き継ぎが発生します。

StartCommand のタイムアウト
System Automation for Multiplatforms が StartCommand を強制終了すると、 引き継ぎが発生します。

MonitorCommand が RC=3 (オフラインに失敗) を返す
System Automation for Multiplatforms が StartCommand を強制終了すると、 引き継ぎが発生します。

Reliable Scalable Cluster Technology

スタンバイ・ノード上の HATS がハートビートのタイムアウトを検出
以下の障害の可能性があります。

- アクティブ・ノードでのハードウェアまたはソフトウェアの障害
- ネットワークまたはネットワーク・アダプターの障害
- hatsd の障害: Mbuf 不足や IP 通信の障害が、アダプターでの データ・トラフィックの過負荷によって引き起こされます。

アクティブ・ノードの softdog モジュール (デッドマン・スイッチ・タイマー) に よるカーネル・パニック
hatsd の障害には、以下のような原因が考えられます。

- 入力または出力の過負荷が原因で hatsd が一時的に使用不可になっている。 この過負荷は、メモリー不足、多数の割り込み、および不適切な優先順位設定 が原因で発生します。
- 障害が原因で hatsd が終了している。

ホーム・ノードへのフェイルバック機能

ホーム・ノードへのフェイルバック機能がアクティブ化されているリソースでは、そのリソースの NodeNameList で最初にある適格ノードが、そのリソースの ホーム・ノードとして定義されます。フェイルバックが起動されるたびに、リソースは、稼働している構成要素での稼働を停止され、ホーム・ノードで開始されます。 NodeNameList の最初のノードが使用可能でない場合は、 リスト内の次のノードがホーム・ノードの役割を引き継ぎます。

ホーム・ノードへのフェイルバック機能の仕組み

ホームへのフェイルバックの起動には、複数の前提条件があります。

- リソースはオンラインです。
- リソースはホーム・ノードで実行されていません。

以下のイベントにより、ホーム・ノードへのフェイルバックが起動されます。

- ホーム・ノードがオンラインになる
- 除外されたホーム・ノードが再度組み込まれる
- ホーム・ノードの構成要素が適格になる (つまり、オフラインに失敗、 不明、または不適格のいずれでもない)

ホーム・ノードへのフェイルバック機能は、グループ・メンバーの位置制約、関係、および移動要求と競合する場合があります。 そのようなシチュエーションでは、以下のルールが適用されます。

- グループ・メンバーの位置制約と競合するフェイルバック: メンバー・ロケーションが連結されているグループでは、アクティブ・フェイルバックのないリソースがフェイルバックによって停止されることはありません。つまり、すべてのリソースで フェイルバックがアクティブ化されている場合に限り、フェイルバックが実行されます。
- 関係と競合するフェイルバック: フェイルバックでは、すべての関係および 関係による位置制約を受け入れます。つまり、すべてのリソースで同じノードへのフェイルバックが実装されている場合を除き、フェイルバックを実行するためにリソースが停止されることはありません。

- 移動要求と競合するフェイルバック: フェイルバック・リソースを含むグループに対する移動要求を実行した場合、それらのリソースは、リソースのホーム・ノード上に即時にフェイルバックされる可能性があります。 フェイルバックが望ましくない場合には、以下の 2 つのオプションがあります。
  - グループ・メンバーの `SelectFromPolicy` 属性を「order」または「any」に変更して、ホーム・ノードへのフェイルバック機能を非アクティブ化することができます。
  - *System Automation for Multiplatforms* インストールと構成のガイドの説明に従って、ホーム・ノードを除外できます。

ホーム・ノードへのフェイルバック機能の動作の例

次のセクションでは、ホーム・ノードへのフェイルバックのさまざまなシナリオを示します。

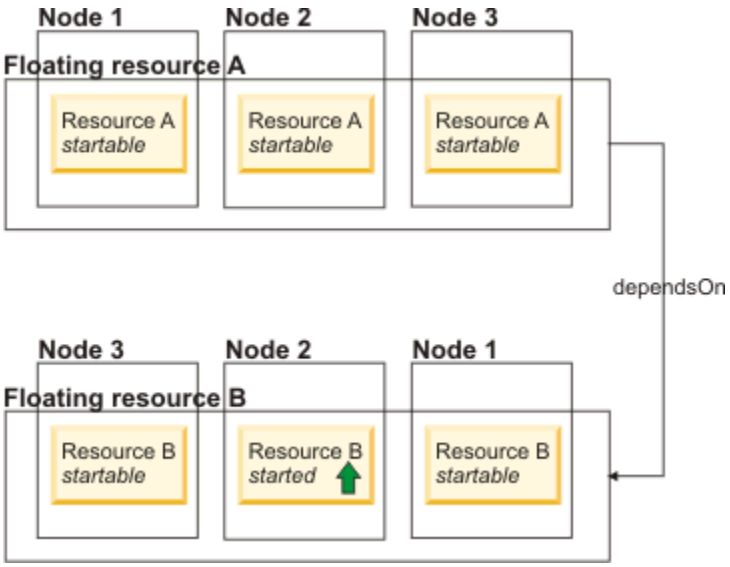


図 71. ホーム・ノードへのフェイルバックがアクティブ化されている 場合の *dependsOn* 関係の例

ホーム・ノードへのフェイルバックがリソース A でアクティブ化されています。リソース A はオフラインであり、 ノード 2 で既に開始されているリソース B に対する dependsOn 関係を持っています。 リソース A が開始命令を受け取ると、リソース A はノード 2 で開始されます。

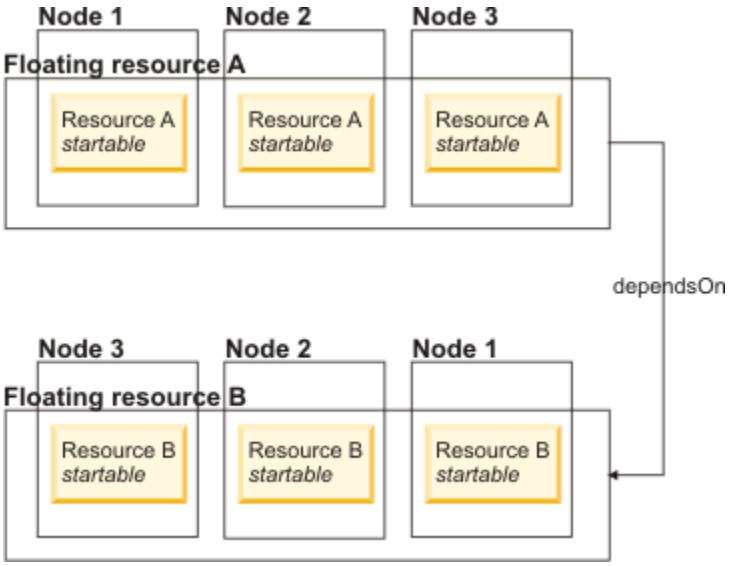


図 72. ホーム・ノードへのフェイルバックがアクティブ化されており 、*NodeNameLists* の順序が異なる場合の *dependsOn* 関係の例

リソース A でホーム・ノードへのフェイルバックがアクティブ化されています。リソース A および B はオフラインで、リソース A からリソース B に対する dependsOn 関係が存在しています。これらのリソースで

NodeNameLists の順序が異なっています。 リソース A が開始命令を受け取った場合は、ノード 1 がリソース A の ホーム・ノードであるため、両方のリソースがノード 1 で開始されます。

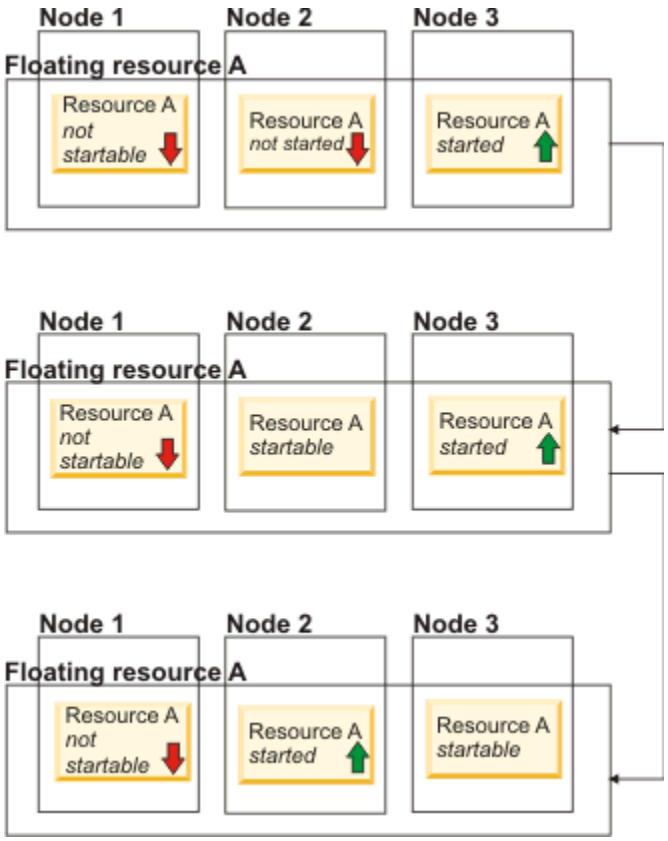


図 *73.* ホーム・ノードへのフェイルバック機能がアクティブ化されている リソースの例

ホーム・ノードへのフェイルバックが、ノード 3 で 稼働しているリソース A でアクティブ化されています。ノード 2 が使用可能になった 場合、ノード 1 はまだ使用不可であるため、ノード 2 がホーム・ノードにもなります。リソース A はノード 2 に移動されます。

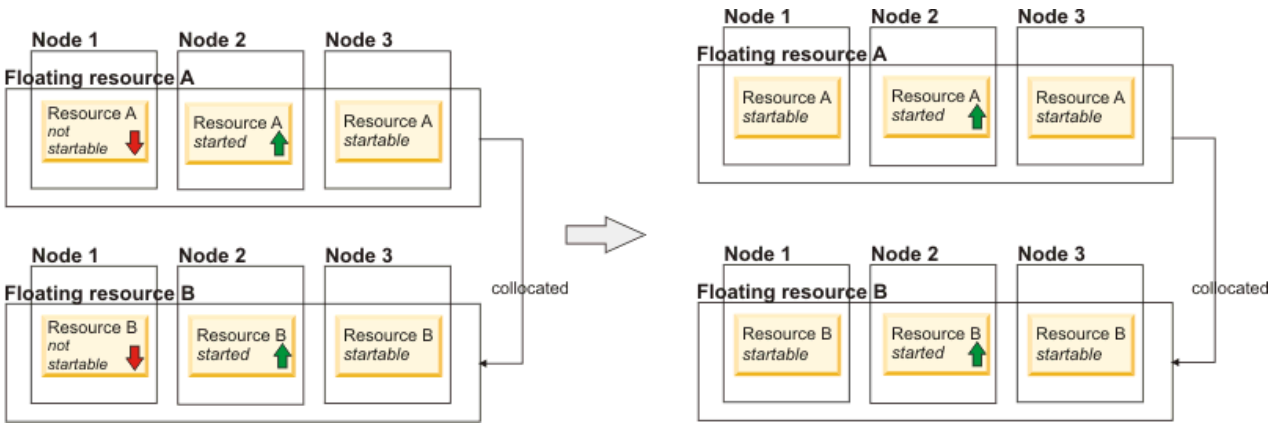


図 *74.* ホーム・ノードへのフェイルバックが *1* つのリソースでのみアクティブ化されている場合の *Collocated* 関係の例

ホーム・ノードへのフェイルバックがリソース A でアクティブ化されています。リソース A は、 リソース B に対する Collocated 関係を持ちます。ノード 1 が使用可能になった場合に、 フェイルバックを実行するために停止されるリソースはありません。代わりに、リソースがノード 2 でオンラインのままになります。

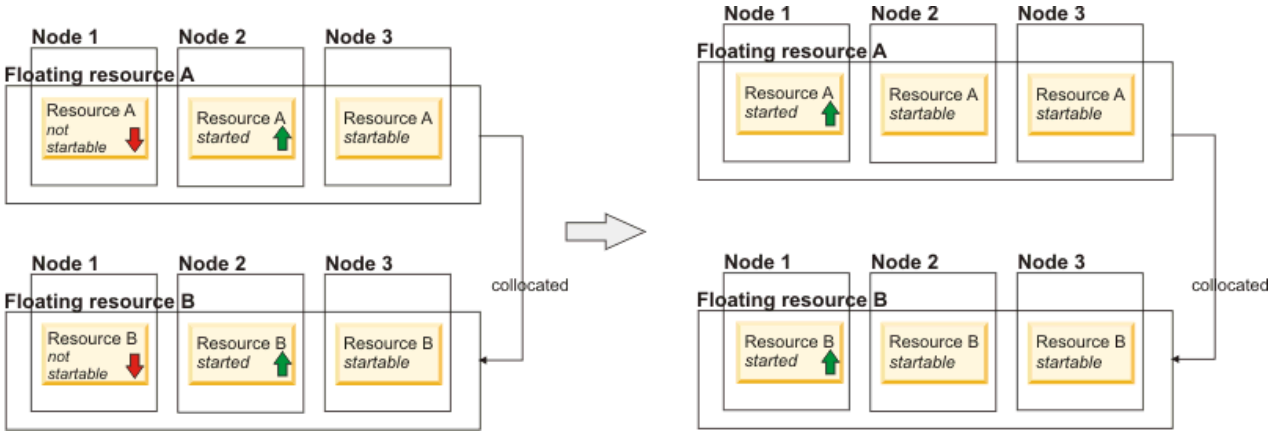


図 75. ホーム・ノードへのフェイルバックが両方のリソースでアクティブ化されている場合の *Collocated* 関係の例

ホーム・ノードへのフェイルバックがリソース A およびリソース B でアクティブ化されています。リソース A は、リソース B に対する Collocated 関係を持ちます。ノード 1 が使用可能になった場合、ノード 1 は両方のリソースのホーム・ノードであるため、両方のリソースがノード 1 に移動されます。

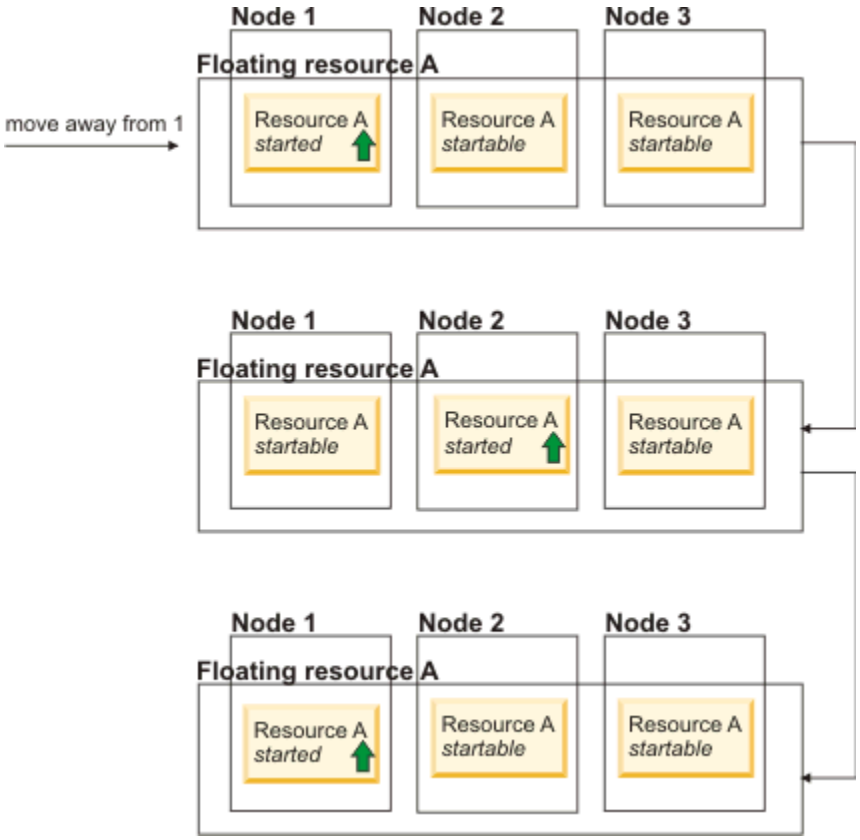


図 76. ホーム・ノードへのフェイルバック機能がアクティブ化されている リソースへの移動要求の例

ホーム・ノードへのフェイルバックがリソース A でアクティブ化されています。リソース A が移動要求を受け取ると、リソース A はその要求を受け入れますが、直後にホーム・ノードを認識し、ホーム・ノードに戻ります。

並行リソースの自動化動作

このタスクについて

並行リソースは、同値と似ていますが、グループ・メンバーシップが 許可されます。つまり、グループの公称状態を介して、アクティブに開始および 停止できます。並行リソースは、浮動リソースと同じく集合リソースですが、 構成要素リソースの 1 つのみではなくすべてを開始します。このセクションのトピックでは、並行リソースを定義および操作する方法について説明します。

並行リソースの定義

このタスクについて

並行リソースは、ResourceType 属性の値に 2 を設定することによって定義されます。例えば、リソース・クラス IBM.Application の並行リソースを定義するには、以下のように入力します。

```
mkrsrc IBM.Application Name="concurrentApp"
StartCommand="/usr/scripts/concurrentControl start"
StopCommand="/usr/scripts/concurrentControl stop"
MonitorCommand="/usr/scripts/concurrentControl monitor"
 user="root" ResourceType=2
```

並行リソースの *OpState* の合成

このタスクについて

並行リソースおよび浮動リソースは集合リソースであるため、このいずれのリソース・タイプについても、OpState は同様な方法で合成されます。 並行リソースの OpState の合成方法について詳しくは、78 ページの『集合リソースの OpState の合成』を参照してください。

並行リソースの制約事項

すべてのリソースを並行リソースとして定義できるわけではありません。さらに、 並行リソースの属性には、一連のオプションの一部に対応していない属性も あります。

- 並行リソースは、リソース・クラス IBM.Application および IBM.Test でのみサポートされます。
- 並行リソースでは、SelectFromPolicy 属性に対する値 Ordered,Failback は許可されません。したがって、ホームへのフェイルバックはサポートされません。
- 並行リソースは、メンバーシップの位置制約を強制するグループのメンバー にすることはできません (つまり、並行リソースの追加先は、-l none オプション を指定して作成されたリソース・グループのみでなければなりません)。
- 並行リソースでは、一部のタイプの関係に対して特別な動作が実装されます。 その他のいくつかの関係については、並行リソースをソースまたはターゲットにする ことはできません。これらの例外ケースについては、66 ページの『並行リソースとの関係を定義する上での例外ケース』で説明します。

並行リソースとの関係を定義する上での例外ケース

並行リソースに対する関係を定義すると、追加の複数のルールが 適用されます。66 ページの表 10 は、通常の動作との差異を示します。空のセルは、差異がないことを 意味します。

表 *10.* 並行リソースに対して定義された関係における 特別な動作

Relationship	**ソースとしての並行リソース**	**ターゲットとしての並行リソース**
Affinity		
AntiAffinity	常に無視される	常に無視される
AntiCollocated	禁止	禁止
Collocated		リソースは同値のように動作する

表 10. 並行リソースに対して定義された関係における 特別な動作 (続き)		
Relationship	**ソースとしての並行リソース**	**ターゲットとしての並行リソース**
依存	他のリソースが並行リソースまたは同値 である場合の特別な動作 [1]	
DependsOnAny	禁止	禁止
ForcedDownBy	ターゲットが同値であれば禁止	
IsStartable		
StartAfter	ターゲットが同値であれば禁止	
StopAfter	ターゲットが同値であれば禁止	

注 : サポートされない関係を並行リソースに対して設定した場合、 結果の自動化動作は未定義です。

1. 並行リソースが、別の並行リソースまたは同値との dependsOn 関係の ソースまたはターゲットである場合、そのリソースは、一致する各構成要素間 に dependsOn 関係が存在するかのように動作します。 ソースの構成要素で、一致する構成要素のないすべての構成要素では、 集合リソースに対する dependsOnAny 関係が実装されます。 それらの構成要素は、その集合が「オンライン」状態を報告するとすぐに開始されます。 次の例で、この動作を説明します。

   並行リソース ResA が ノード 1、2、および 3 に定義されています。別の並行リソース ResB が ノード 1 および 2 に定義されています。ResA をソース、ResB をターゲット として dependsOn 関係が定義されています。両方のリソースがオフライン であるときに、ResA が開始された場合、構成要素 ResA:1 および ResA:2 は、 それぞれ ResB:1 および ResB:2 に対する dependsOn 関係を持つかのように 動作します。ResA:3 は、集合リソース ResB に対する dependsOn 関係を持つ かのように動作し、ResB の少なくとも 1 つの構成要素が開始されるとすぐに 開始されます。

並行リソースでのクリーンアップ処理の使用

並行リソースはクリーンアップ処理に対応しています。ただし、リソースの セットアップ後に、追加の複数の考慮事項を検討する必要があります。 浮動リソースの場合は、すべてのノードでクリーンアップ・コマンドを 実行して、各構成要素リソースを確実にクリーンにすることができます。並行リソース の場合は、他方の構成要素がまだ実行中である可能性があるため、これを 行うことはできません。この問題を解決するには、各構成要素で独自の クリーンアップ・コマンドを定義する必要があります。各構成要素リソースのクリーンアップ・コマンドを個別に設定するには、並行リソースの作成後に chrsrc コマンドを 使用します。次のコードでは、machine1 上のサンプル構成要素リソース myApplication の クリーンアップ・コマンドを変更します。

```
chrsrc -s “Name like 'myApplication' & NodeNameList='{machine1}'”
IBM.Application CleanupCommand='/usr/bin/my_cleanup machine1'
```

リソース・パターン

このセクションでは、リソースとリソース・グループに関連した特定のシチュエーションにおける System Automation for Multiplatforms の動作について説明します。

コマンドの使用

以下のセクションでは、StartCommand、MonitorCommand、StopCommand、および CleanupCommand に関連する問題について説明します。

StartCommand

IBM Tivoli System Automation for Multiplatforms では、リソースの StartCommand 属性に指定されたコマンドを使用して、リソースをオンラインにします。

StartCommand は以下の状況で呼び出されます。

- リソース・グループの NominalState 属性がオンラインに変更され、このリソースのすべての開始依存関係が満たされた直後。

- リソースの障害が原因で、リソースの OpState がオンラインからオフラインに変更された直後。

以下のいずれかのイベントの結果、OpState がオフラインに変更された場合は、StartCommand は呼び出されません。

- リソース・グループの NominalState がオフラインに変更された。
- 別のリソースへの依存関係を満たすために System Automation for Multiplatforms によってリソースが停止または強制終了した場合。

StartCommand は実行されているが、このリソースが、オンライン・タイムアウトになったときに依然オフラインであり、かつ StartCommand 実行試行最大数 (RetryCount 属性に定義されている) に到達していない場合、リソースの「オンライン」タイムアウト (秒単位) は、次の公式を使用して計算されます。

```
(StartCommandTimeout + MonitorCommandPeriod + MonitorCommandTimeout + 5)
```

注 :

1. System Automation for Multiplatforms は実タイマーを使用しないため、+ 5 は絶対値ではありません。実際の値は、System Automation for Multiplatforms デーモンがいつ再度起動されるか (頻繁に起動される) によって、5 から 8 秒の範囲内になることがあります。
2. この「オンライン」タイムアウトは、StartCommand の以前の実行中にリソースの OpState が変更されなかった場合のみ評価されます。OpState が変更された場合は (例えば、「オンラインの保留中」または「オンライン」に)、オンライン・タイムアウト・タイマーはキャンセルされ、リソースの OpState が再度オフラインに変更された直後に、リソースの StartCommand が呼び出されます。

デフォルトでは、System Automation for Multiplatforms は StartCommand を同期的に実行します。つまり、System Automation for Multiplatforms はコマンドの終了を待ち、戻りコードの情報を取得します。

オンライン・タイムアウトに加えて、それぞれのリソースには StartCommandTimeout 属性があります。StartCommandTimeout 属性は、StartCommand の最大実行時間を決定します。StartCommand が StartCommandTimeout の期間内に戻らない場合、System Automation for Multiplatforms は SIGKILL コマンドを使用して StartCommand を強制終了し、ノードのシステム・ログにメッセージを記録します。

StartCommandTimeout の期間内で開始されたアプリケーション・プロセスが System Automation for Multiplatforms に制御を戻さない場合は、これが問題の原因になることがあります。 そのような場合は、StartCommandTimeout に達するたびにアプリケーション・プロセスが強制終了されるため、結果的に System Automation for Multiplatforms はこのアプリケーションをリソースとして開始できません。 この問題を訂正するには、以下のいずれかの方法を使用して、アプリケーション・プロセスを、呼び出し側 StartCommand から切り離す必要があります。

- すべてのファイル・ハンドルをファイルにリダイレクトし、アプリケーション・プロセスをバックグラウンドで開始します。例えば、以下のようにします。

  ```
  /usr/bin/application  >/outputfile  2>&1   &
  ```

- 「setsid()」C 関数を使用する小さいラッパー・アプリケーションを作成して、アプリケーション・プロセスを呼び出し側 StartCommand から切り離します。
- 前述の方式が機能しないか、あるアプリケーションに適さない場合は、同期コマンド・ビットに 0 を設定します。この場合、System Automation for Multiplatforms は、このリソースの StartCommandTimeout 属性を無視し、StartCommand およびそのすべての子プロセスはこのノード上に無期限に留まることができます。

  注: この方法を使用する場合に System Automation for Multiplatforms はこの StartCommand の戻りコードを待たないため、StartCommand が失敗してもそのようなリソースはフェイルオーバーされません。リソースがオンライン・タイムアウト期間中にオンラインにならない場合、System Automation for Multiplatforms は、RetryCount に達するまで StartCommand を再度呼び出します。

MonitorCommand

`ProcessCommandString` が設定されていない場合、System Automation for Multiplatforms は、`IBM.Application` リソースの `MonitorCommand` を使用して、ノード上のこのリソースの `OpState` を判別します。System Automation for Multiplatforms は、リソースがリソース・グループまたは同値に追加さ

れるとすぐにそのモニターを開始します。 リソースは、そのリソースの実行が許可されるすべてのノード上でモニターされます。リソースの NodeNameList 属性で指定されているノードでの実行が許可されます。

最初の呼び出し後、MonitorCommand は MonitorCommandPeriod 属性で定義された頻度で呼び出されます。リソースは、リソースがリソース・グループまたは同値から除去されるまで、リソースが定義されている各ノードで無期限にモニターされます。

MonitorCommand は、リソースの StartCommand または StopCommand が完了した直後にも呼び出されます (デフォルトの場合、すなわちこのリソースの RunCommandsSync 属性が 1 に設定されている場合は、同期コマンドについてのみ)。これは、リソース・グループ全体の開始または停止の性能を強化するために導入されたものであり、StartCommand または StopCommand の完了直後にリソースの OpState が確認されるようになりました。この MonitorCommand の実行後は、再び MonitorCommandPeriod 秒の頻度で実行されます。つまり、次の MonitorCommand は、MonitorCommandPeriod 秒後に実行されます。

MonitorCommandPeriod 属性の値は、指定されたリソースの MonitorCommandTimeout 属性の値を下回ることがあります。この可能性があるのは、MonitorCommandPeriod が、実際には MonitorCommand の最後の実行の終了と次の開始の間の待機期間を指定するためです。結果として、MonitorCommandTimeout は、MonitorCommandPeriod とは独立して指定できるようになり、したがって、頻度の高い MonitorCommand 実行は、実行が長い MonitorCommand スクリプトの場合でも認識できます。しかし、依然として MonitorCommand は可能な限り有効にしておくことをお勧めします。

MonitorCommand に関して、問題または異常な動作を避けるために注意すべきことが 2 つあります。

1. MonitorCommand は、リソースを実行可能なすべてのノード上で実行されます。リソースを停止する (リソース・グループの NominalState がオフライン) 必要があり、オペレーターがこのリソースを手動で開始する場合、System Automation for Multiplatforms は、このことをリソースの MonitorCommand を使用して通知し、リソースに対して StopCommand を実行して、再びオフラインにします。

   個々のリソース・グループ・メンバーをオンラインまたはオフラインにする必要がある場合 (例えばバックアップを実行する場合)、System Automation for Multiplatforms 要求を使用する必要があります (**rgreq** コマンド)。この要求は、リソース・グループの NominalState を却下し、リソースの NominalState がオフラインであっても、リソース・グループ・メンバーが開始されるのを許可します。
2. MonitorCommandTimeout 属性に指定されたタイムアウト期間の有効期限が切れる前に実行中の MonitorCommand が完了しなかった場合は、 System Automation for Multiplatforms は SIGKILL コマンドを使用して MonitorCommand を強制終了し、ノードのシステム・ログにメッセージを記録し、リソースの OpState を「不明」に設定します。その結果、このリソースとすべての従属リソースは、意味のある状態を再度判別できるようになるまで自動化されません。メッセージがシステム・ログ内に頻繁に出現する場合は、MonitorCommandTimeout 属性の値を確認して、必要に応じて調整する必要があります。

ProcessCommandString の使用によるリソースのモニター

リソースをモニターする方法として、2 つのオプションがあります。

1. MonitorCommand によって定義されているスクリプトを介したモニター
2. リソースの ProcessCommandString 属性の設定によるモニター

ProcessCommandString を設定した場合は、定義した値をプロセス・テーブルで スキャンすることにより、リソースの状態が判別されます。値が 検出される場合、リソースはオンラインであると評価されます。値が 検出されない場合、リソースはオフラインであると評価されます。

リソースをモニターするプロセス・テーブル・スキャンのアクティブ化

プロセス・テーブル・スキャンでは、ワイルドカード・パターンを検索して、 最初に一致した時点で検出したと解釈します。POSIX.2 fnmatch() システム・コール のワイルドカード・ルールが適用されます。以下のパターンが使用可能です。

- アスタリスク (*) は 0 から n 個の文字と一致します。

- 疑問符 (?) は任意の単一文字と一致します。
- 大括弧 ([ ]) は、大括弧内の任意の値 1 つと一致します。

fnmatch の資料を参照するには man fnmatch と入力してください。

ProcessCommandString がアクティブであっても、モニター・スクリプトはまだ存在している必要があります。ProcessCommandString が既存のリソースに追加されると、 モニター処理は、モニター・スクリプトの実行からプロセス・テーブルのスキャンに 変更されます。その後で ProcessCommandString を削除すると、モニター処理が変更されて、 モニター・スクリプトの実行に戻されます。サンプル・リソース myApplication に 対するプロセス・テーブル・スキャンを使用可能にするには、次のように入力します。

```
chrsrc -s „Name like 'myApplication'“ IBM.Application
ProcessCommandString=“/usr/sbin/httpd2*“
```

ProcessCommandString および *ProbePID* の使用

RunCommandsSync 属性の第 7 ビットは、ProbePID がアクティブかどうかを 制御します。ProcessCommandString が設定されており、ProbePID がアクティブ である場合は、リソースのモニター動作が変更されます。プロセス・テーブルでの 一致が検出された場合に、リソースは、オンラインであると直接評価されません。 代わりに、MonitorCommand が実行されて、リソースの最終状態が判別 されます。

オフライン・リソースに対してプロセス・テーブル・スキャンがもたらすパフォーマンスの向上を享受する一方で、MonitorCommand を実行して、リソースが実際にオンラインであることを確認する場合に、この動作を使用します。 ProbePID を設定することにより、プロセス・テーブルにあるストリングが同一の可能性 のある 2 つのアプリケーションを区別することもできます。

ProbePID をアクティブ化するには、RunCommandsSync ビット・マスクの第 7 ビット を設定する必要があります。70 ページの表 11 は、RunCommandsSync 属性の値と、 ProbePID に対するそれらの影響を示します。

表 *11. ProbePID* のアクティブおよび非アクティブの状態

RunCommandsSync	**ProbePID**
0 - 63	非アクティブ
64 - 127	アクティブ
128 - 191	非アクティブ
192 - 255	アクティブ

サンプル・リソース myApplication に対して probePID をアクティブ化 するには、次のように入力します。

```
chrsrc -s „Name like 'myApplication'“ IBM.Application
ProcessCommandString=“/usr/sbin/httpd2*“ RunCommandsSync=65
```

リソースの動作のモニター

ProcessCommandString 属性を導入することにより、System Automation for Multiplatforms では、 リソースの状態をモニターする方法を複数提供するようになりました。71 ページの図 77 は、System Automation for Multiplatforms によるリソースの状態の評価方法を示します。

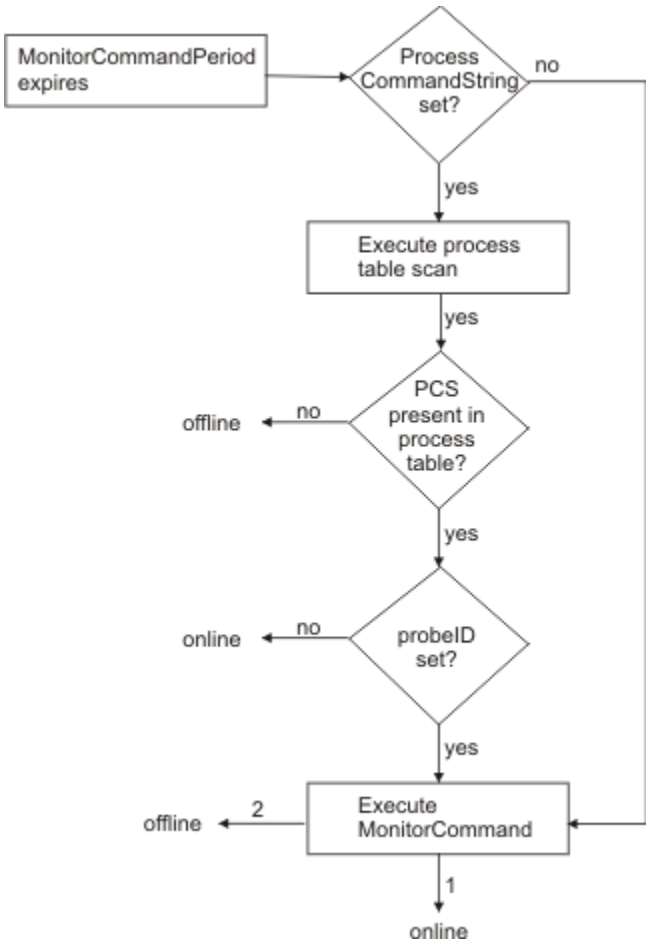


図 77. リソースの状態の評価

MonitorCommandPeriod の有効期限が切れるたびに、System Automation for Multiplatforms はプロセス・テーブル・スキャンを実行して ProcessCommandString を検索します。

- エントリーを検出した場合:
  - ProbePID ビットが設定されていれば System Automation for Multiplatforms は、アプリケーションの状況を判別する MonitorCommand を実行します。
  - ProbePID ビットが設定されていなければアプリケーションの状況はオンラインです。
- エントリーが検出されなかった場合: リソースの状況はオフラインまたはオフラインに失敗 (状況が根拠になる場合) です。

StopCommand

デフォルトの場合、StopCommand は、System Automation for Multiplatforms によって同期的に実行されます。つまり、System Automation for Multiplatforms はコマンドが終了するのを待って、戻りコードの情報を獲得します。 さらに、各リソースには、StopCommand の実行の最長時間を決定する属性 StopCommandTimeout があります。StopCommand が StopCommandTimeout の時間内に戻らない場合、System Automation for Multiplatforms は SIGKILL コマンドを使用して StopCommand を強制終了します。これが発生すると、そのノードのシステム・ログにメッセージが記録されます。

リソースの StopCommand が実行されたが、リソースの OpState が、「オフライン」タイムアウト期間内に「オフライン」に変わらない場合、System Automation for Multiplatforms は、そのリソースに対してリセット操作を実行します。 リソースの「オフライン」タイムアウト期間 (秒単位) は、以下の公式を使用して計算されます。

```
MAX(StopCommandTimeout, MonitorCommandPeriod, MonitorCommandTimeout) + 5
```

System Automation for Multiplatforms は実タイマーを使用しないため、+ 5 は絶対値ではないことに注意してください。 System Automation for Multiplatforms デーモンは頻繁に起動され、その結果、値が追加されて、5 から 8 秒の範囲内になることがあります。

リソースに対してリセット操作を実行すると、StopCommand の 2 番目の実行が行われますが、このときは、特殊な環境変数 (SA_RESET) が 1 に設定されます。これは、リソースの StopCommand 内で活用され、例えば以下のような拡張強制動作を実現できます。

```
#/bin/sh
# A sample stop/reset automation script for the lpd application
if [ $SA_RESET -eq 1 ]; then
  killall -9 lpd
  exit $?
else
  /etc/init.d/lpd stop
  exit $?
fi
```

StopCommand の 2 番目を実行する必要がない場合、例えば次のようにして、スクリプトをただちに終了できます。

```
#/bin/sh
# A sample stop/reset automation script for the lpd application
if [ $SA_RESET == 1 ]; then
  exit 0
else
  /etc/init.d/lpd stop
  exit $?
fi
```

リソースが StopCommand の 2 番目の実行後も依然停止しない場合、リソースの OpState は「オンライン中」に設定されます。つまり、ここでリソースの停止には手操作による介入が必要です。 System Automation for Multiplatforms がリソースの自動化を再開するのは、オペレーターによって問題が解決した後のみです。

手動でリセットすると、指定されたすべてのリソースについて停止コマンドが実行されるようになりました。選択文字列内に NodeNameList 属性を指定せずに **resetrsrc** コマンドを実行すると、停止コマンドはすべてのリソースに対して実行されます。リソース名のみを浮動リソースの選択文字列に指定すると、停止コマンドは、構成要素ごとと集合リソースに対して実行されます。これは、選択文字列が集合リソースと構成要素リソースから構成されるためです。 集合リソース自体を処理する停止コマンドがすべての構成要素リソースに渡されるため、すべての構成要素リソースは、停止コマンドを 2 度実行します。このため、コマンドの選択文字列内で resetrsrc を実行するたびに NodeNameList 属性を指定することをお勧めします。

CleanupCommand

クリーンアップ処理により、リソースの失敗後に、環境を復元し、リソースを始動可能に戻すことができます。

CleanupCommand 要件

System Automation for Multiplatforms 3.2.1 では、IBM.Application リソース・クラスのみが、クリーンアップ処理をサポートしています。さらに、クリーンアップ処理は、浮動リソース および並行リソースでのみサポートされています。クリーンアップ処理の実行には、 さらに以下の要件があります。

- リソースが IBM.Application リソース・クラスの集合リソースである
- リソースはリソース・グループのメンバーである
- クォーラムが設定されている
- クラスターが手動モードではない
- アクティブなロック要求がない
- リソースが除外ノード上にない
- クリーンアップ・フラグにダーティーが設定されている

クリーンアップ処理の起動

クリーンアップ処理をアクティブ化するには、任意指定の属性である CleanupCommand に 値を指定します。さらに、CleanupNodeList と CleanupCommandTimeout の 2 つの任意指定の属性を設定できます。この 2 つの属性を設定しなかった場合、System Automation for Multiplatforms では、それらに対するデフォルト値を 使用します。CleanupCommandTimeout のデフォルト値は 5 秒です。 CleanupNodeList のデフォルト値は、NodeNameList のミラーリング です。クリーンアップ処理を非アクティブ化するには、CleanupCommand に空値を設定します。

次の例は、既に定義されている myApplication という浮動リソースに対する クリーンアップ処理をカスタマイズする方法を示します。StartCommand、StopCommand、 および MonitorCommand 同様、CleanupCommand によって定義されるスクリプトは、そのリソースでアクセス可能な各ノードで使用可能であり、実行可能である 必要があります。

- クリーンアップ処理を追加するには、以下のように入力します。

```
chrsrc -s “Name like ‘myApplication’ & ResourceType=1”
IBM.Application CleanupCommand=’/usr/bin/my_cleanup’
```

- 非デフォルト値の CleanupNodeList および CleanupCommandTimeout を指定してクリーンアップ処理を追加するには、以下のように入力します。

```
chrsrc -s “Name like ‘myApplication’ & ResourceType=1”
IBM.Application CleanupCommand ='/usr/bin/my_cleanup'
CleanupCommandTimeout=30 CleanupNodeList='{“machine1”, “machine2”, “machine4”}'
```

- リソースからクリーンアップ処理を削除するには、以下のように入力します。

```
chrsrc -s “Name like ‘myApplication’ & ResourceType=1”
IBM.Application CleanupCommand=“”
```

- myApplication の状態を表示させ、クリーンアップ・フラグが設定されているか どうかを確認するには、samdiag コマンドを使用します。

```
#samdiag IBM.Application:myApplication:node01
```

クリーンアップ・フラグにダーティーが設定されている場合は、 このコマンドにより、以下の出力が行われます。

```
#samdiag IBM.Application:ca_bobwang_0-rs:coralm207
Diagnosis::Resource: ca_bobwang_0-rs/Fixed/IBM.Application/coralm207
    type: Fixed Resource
    Status -
        Reported: Online        - Online
        Observed: Online        - Online
        Desired: Online        - Requested Online
        (Nominal: Offline        - Defaulted: offline)
        Automation: Idle        - Idle - Online completed
        Startable: Yes        - Resource is startable
        Binding: Bound        - Bound
        Compound: Satisfactory    - Satisfactory
        Move: None            - Resource Move State is None
Resource Based Quorum: In Quorum - Resource has Quorum
Cleanup Flag: Dirty
Groups and Aggregates:
    <---HasMember    ---- ca_bobwang_0-rs/Concurrent/IBM.Application
    <---Selects/1    ---- ca_bobwang_0-rs/Concurrent/IBM.Application
```

自動化動作

リソースのクリーンアップは、以下の状態の場合に起動されます。

- リソースの構成要素の少なくとも 1 つが、オンラインから いずれかの非アクティブ状態に予期せず切り替わる。つまり、System Automation for Multiplatforms によって送信された 停止命令の結果ではない場合です。非アクティブ状態とは、オフライン、オフラインに失敗、 および不適格です。
- リソースの始動時に障害が発生する。

クリーンアップが起動されると、CleanupNodeList の最初のノード で CleanupCommand が実行されます。このノードでコマンドが 失敗するかタイムアウトになった場合は、クリーンアップが成功するまで、次のノードの選択が反復されます。CleanupNodeList で使用可能なノードが なくなると、クリーンアップはリセットされて、再度最初のノードで開始されます。

クリーンアップに関するリソースの状態がクリーンアップ・フラグに設定されます。 リソースの開始後に、ダーティーがこのフラグに設定されます。StopCommand または CleanupCommand が正常に実行されると、このフラグが削除されます。 クリーンアップ・フラグにダーティーが設定されているリソースは、開始されません。 代わりに、リソースをクリーンアップする CleanupCommand が実行されます。

リソースのクリーンアップ処理をアクティブ化すると、このリソースの始動動作が以下のように変更されます。

- クリーンアップ・フラグの値にダーティーが設定されている場合、リソースは開始されず、まず CleanupCommand が実行されます。
- StartCommand の実行後は、クリーンアップ・フラグに ダーティーが設定されます。
- StartCommand が正常に実行されないと、クリーンアップ・フラグに ダーティーが設定されます。

リソースのクリーンアップ処理をアクティブ化すると、リソースのシャットダウン動作が以下のように変更されます。

- StopCommand が正常に実行されると、クリーンアップ・フラグにクリーンが設定されます。
- StopCommand が正常に実行されず、リソースの OpState がオフラインの 場合、クリーンアップ・フラグにはダーティーが設定されたままであり、CleanupCommand が実行されます。

進行中のクリーンアップを中断したり、保守を実行したりするためにクリーンアップ・フラグ の状態を手動で制御するには、以下のコマンドを使用します。クリーンアップ・フラグを設定するには、OpCode=1 を使用します。クリーンアップ・フラグを削除して、リソースの再開を許可するには、OpCode=3 を使用します。

```
runact -c IBM.CHARMControl HandleDirtyBit Resource="resHandle" OpCode=1
runact -c IBM.CHARMControl HandleDirtyBit Resource="resHandle" OpCode=3
```

リソースの resourceHandle 値を取得するには、次のコマンドを 入力します。

```
lsrsrc -s "Name like <resourceName> && ResourceType=0" <ResourceClass> ResourceHandle
```

オンライン・リソースの *OpState* の変更

次の構成例は、クラスター環境でのオンライン・リソースの OpState の変更を示しています。

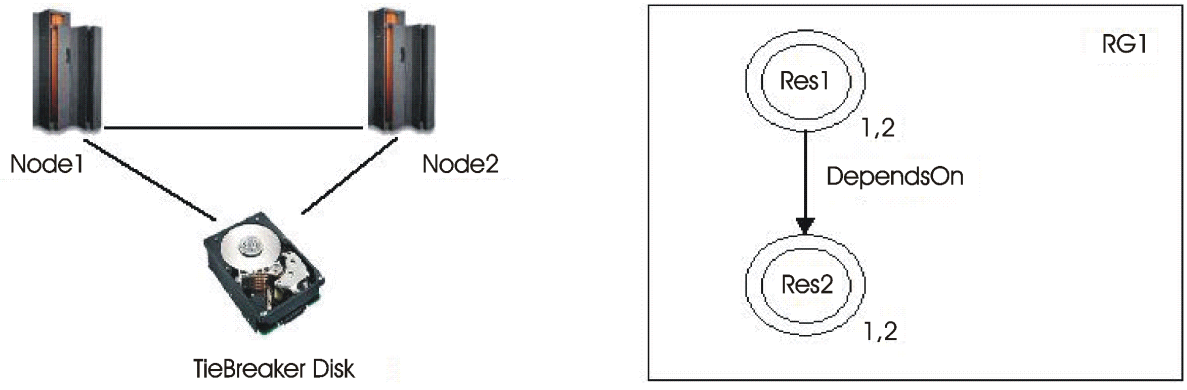


図 *78.* サンプル構成のセットアップ

このサンプル構成のセットアップは以下のとおりです。

- 2 つのノードのクラスター。
- ディスク・タイ・ブレーカー

  - Node1: 実動システム。
  - Node2: スタンバイ・システム。
- 以下を伴うリソース・グループ RG1
  - 浮動リソース: Res1
  - 浮動リソース: Res2
  - 関係: Res1 は Res2 に対して DependsOn
- Node1 上のリソースはオンラインである。

75 ページの図 79 に、System Automation for Multiplatforms によって自動化されるリソースの代表的な OpState 遷移を示します。

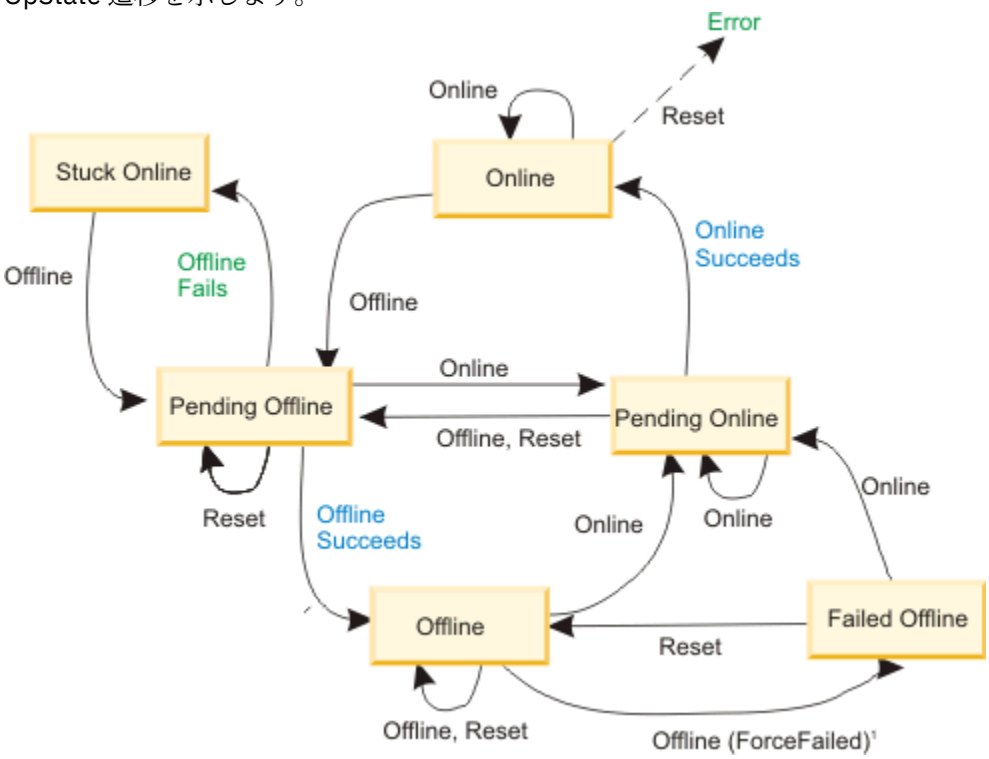


図 79. 自動化リソースの操作状態遷移

注:

1. オフライン (ForceFailed)。失敗フラグを設定してオフライン・メソッド **stopsrc** を実行すると、 リソースに、オフラインに失敗のマークが永久的に付けられます。このシチュエーションは、開始の試行が失敗した後に発生する可能性があります。 82 ページの表 18 を参照してください。

リソースの OpState には、7 個の値を指定できます。リソースの OpState は、System Automation for Multiplatforms が MonitorCommand を使用して判別します。リソースの実際の OpState は MonitorCommand の戻りコードとともに System Automation for Multiplatforms に提供されます。MonitorCommand が System Automation for Multiplatforms に OpState 値の「オンライン」および「オフライン」を戻せば十分です。 リソースの他の OpState 値は、すべてオプションで使用できます。

「不明」、「オンライン中」または「オフラインに失敗」のような OpState 値によっては、System Automation for Multiplatforms によって設定されるものがあります。例えば、OpState「不明」は、リソースの MonitorCommand がタイムアウトの場合にそのリソースに設定されます。したがって、System Automation for Multiplatforms はこのリソースの OpState についての認識を持たなくなります。

以下の 2 つの表で、リソース Res1 および Res2 の OpState の現在の例からの変更に対する System Automation for Multiplatforms の対応方法を説明します。 ここで説明する表では、この状況では該当しない特定の値も含め、リソースで使用可能なすべての OpState 値をリストします。該当しない可能性があるこれらの OpState 値を含む列は、以下のテーブルで、「無効である可能性がある」という言葉が先頭に付いて表示されます。

System Automation for Multiplatforms の障害への対応方法は、RecoveryPolicy 属性に 依存します。ここで説明する動作は、 デフォルト値 RecoveryPolicy=AutomatcRecovery の場合に適用されます。 異なるリカバリー・ポリシー・オプションが設定されている場合の動作の詳細な説明については、27 ページの『リソース・グループ・メンバーに対して使用される属性』を参照してください。

リソース *Res1* の *OpState* の変更

node1 の Res1 および Res2 の現在の状況はオンラインです。以下の表に、Res1 に対する MonitorCommand の戻り値に応じて System Automation for Multiplatforms が実行するアクションを示します。

表 *12.* リソース *Res1* の *OpState* の変更に関する *System Automation* のアクション

MonitorCommand (OpState)		**System Automation の最初のアクション**	**System Automation の 2 番目のアクション**
RC=0 (不明)	=>	なし、次に MonitorCommand が RC<>0 を戻すまで待機	なし
RC=1 (オンライン)	=>	なし	なし
RC=2 (オフライン)	=>	Res1 を開始する	なし
RC=3 (オフラインに失敗)	=>	Res2 を停止する	Res2 がオフラインになった後、node2 の両方のリソースを正しい順序で開始する
RC=4 (オンライン中)	=>	なし、オペレーター・アクションを待つ	なし
RC=5 (オンラインの保留中)	=>	無効である可能性がある、オンラインになるのを待つ	なし
RC=6 (オフラインの保留中)	=>	オフラインになるのを待つ	なし
RC=8 (不適格)	=>	Res2 を停止する	Res2 がオフラインになった後、node2 の両方のリソースを正しい順序で開始する。

リソース *Res2* の *OpState* の変更

node1 の Res1 および Res2 の現在の状況はオンラインです。以下の表に、Res2 に対する MonitorCommand の戻り値に応じて System Automation for Multiplatforms が実行するアクションを示します。

表 *13.* リソース *Res2* の *OpState* の変更に関する *System Automation* のアクション

MonitorCommand (OpState)		**System Automation の最初のアクション**	**System Automation の 2 番目のアクション**
RC=0 (不明)	=>	なし、次に MonitorCommand が RC<>0 を戻すまで待機	なし
RC=1 (オンライン)	=>	なし	なし
RC=2 (オフライン)	=>	Res1 を強制停止する	Res1 がオフラインになった後、Res2 を開始する。Res2 がオンラインになった後、Res1 を開始する。
RC=3 (オフラインに失敗)	=>	Res1 を強制停止する	node2 の両方のリソースを正しい順序で開始する

表 13. リソース *Res2* の *OpState* の変更に関する *System Automation* のアクション (続き)			
MonitorCommand (OpState)		**System Automation の最初のアクション**	**System Automation の 2 番目のアクション**
RC=4 (オンライン中)	=>	なし、オペレーター・アクションを待つ	なし
RC=5 (オンラインの保留中)	=>	無効である可能性がある、オンラインになるのを待つ	なし
RC=6 (オフラインの保留中)	=>	オフラインになるのを待つ	なし
RC=8 (不適格)	=>	Res1 を強制停止する	node2 の両方のリソースを正しい順序で開始する。

リソース・グループの *OpState* の合成

System Automation for Multiplatforms は、ポリシー・ベースの自動化製品です。自動化の制御点は、リソース・グループ・レベルです。つまり、通常、オペレーターは単一リソースを開始または停止するよりも、リソース・グループ全体を開始または停止します。これを行うには、リソース・グループの NominalState 属性をオンラインまたはオフラインに変更します。この属性が変更された直後に、System Automation for Multiplatforms は変更されたポリシーのルールに従うために開始または停止する必要があるリソースを判別します。

リカバリー・マネージャーは、NominalState を考慮し、場合によってはグループにサブミットされた要求も考慮して、グループをオンラインにする必要があるのか、オフラインにする必要 があるのかを決定し、この判断を DesiredState に保管します。

リソース・グループの OpState (操作状態) 属性は、**リソース・グループの DesiredState 値に関する**、そのリソース・グループに含まれるすべてのリソースの OpState 属性の集約です。そのため、リソース・グループの DesiredState がオンラインに変更されると、そのリソース・グループ内のすべてのリソースがオンラインになるまで、リソース・グループの OpState は「オンラインの保留中」と表示されます。また、そのリソース・グループのすべてのリソースが、リソース・グループの DesiredState 属性の値になると、リソース・グループの OpState がオンラインに変更され、リソース・グループの OpState 属性のこの値を使用して、そのグループ内のリソースの状況をモニターできるようになります。

注 : 要求が直接メンバー・リソースに出されたときのメンバー・リソースの OpState は、それを含むグループとは異なることがあります。 詳しくは、「154 ページの『リソース・グループとグループ・メンバーの開始および停止』」を参照してください。

以下の表に、前述の例の 2 つのリソース Res1 および Res2 の OpState に基づいて、System Automation for Multiplatforms がリソース・グループの OpState 属性の値を構成する方法を示します。単一のリソース・グループに含まれるリソースが増えるほど、この表が複雑になることに注意してください。

表 14. リソース・グループの *OpState* の決定				
Res2 の OpState	**Res1 の OpState**		**リソース・グループの OpState**	**System Automation のアクション**
Unknown	Unknown	=>	Unknown	なし
オフライン	オフライン	=>	オフライン	なし
オンラインの保留中	オフライン	=>	オンラインの保留中	Res2 がオンラインになるまで待つ
オンライン	オフライン	=>	オンラインの保留中	Res1 を開始する
オンライン	オンラインの保留中	=>	オンラインの保留中	Res1 がオンラインになるまで待つ
オンライン	オンライン	=>	オンライン	なし

表 14. リソース・グループの OpState の決定 (続き)				
Res2 の OpState	**Res1 の OpState**		**リソース・グループの OpState**	**System Automation のアクション**
オンライン	オフラインの保留中	=>	オフラインの保留中	Res1 がオフラインになるまで待つ
オンライン	オフラインに失敗	=>	オフラインの保留中	Res2 を停止する
オフラインの保留中	オフライン	=>	オフラインの保留中	Res2 がオフラインになるまで待つ
オフラインに失敗	オフライン	=>	オフライン	なし

集合リソースの *OpState* の合成

集合リソースの OpState は、そのリソースの構成要素の OpState に 依存します。

集合リソースでは、構成要素の状態が異なっている場合がある ため、System Automation for Multiplatforms では、集合リソースの OpState を合成する必要があります。 可能なすべての OpState は、優先順位の順にリストできます。

1. Unknown
2. オンライン中
3. オンライン
4. Starting
5. Stopping
6. 不適格
7. オフライン
8. オフラインに失敗

集合リソースの OpState を作成するために、System Automation for Multiplatforms はすべての構成要素 の実際の OpState を収集します。集合リソースの OpState は、最高の優先順位 の OpState を持つ構成要素の OpState と一致します。 78 ページの表 15 は、集合リソースの OpState の合成例を 示します。

表 15. 集合リソースの *OpState* の合成例			
Res1 の OpState	**Res2 の OpState**		**集合リソースの OpState**
Unknown	オンライン	=>	Unknown
オンライン	Starting	=>	オンライン
オンライン	オフライン	=>	オンライン
オフラインに失敗	Starting	=>	Starting
オフラインに失敗	オンライン	=>	オンライン
オフラインに失敗	オフライン	=>	オフライン

開始または停止されたリソースの *OpState* の変更

System Automation for Multiplatforms は、通常、リソース・グループの DesiredState およびリソースの OpState 値に基づいて、リソースを自動化します。 その目的は、リソースの OpState と DesiredState が同一になる状態を達成し、維持することです。 リソースの OpState が変更される場合 (例えば、稼働していたリソースがここで MonitorCommand によって「オフライン」として報告されるとき)、System Automation for Multiplatforms はアクションを起こします。

自動化アクションをトリガーするもう 1 つの要素は、StartCommand の戻りコードです。StartCommand がエラー (ゼロ以外の戻りコード) を戻し、リソースがオンラインであるとモニターされない場合は、System Automation for Multiplatforms は、リソースを別の適格なノードにフェイルオーバーします。

以降のセクションでは、リソースの StartCommand または StopCommand の実行中または実行直後にリソースの OpState が変更された場合に、System Automation for Multiplatforms が実行するアクションについて説明します。

System Automation for Multiplatforms では、リソースの MonitorCommand の実装について、より規則に準拠する必要があります。 これは、リソースの StartCommand が実行されるときは、特に重要です。 この時 System Automation for Multiplatforms は、リソースの OpState が「オンラインの保留中」であり、したがってこのリソースに OpState が設定されることを認識しています。 唯一の例外は以下のとおりです。

- MonitorCommand によって報告される OpState は「オンライン」に変わります (この場合は自動化の目的に到達している)
- MonitorCommand によって報告される OpState は「オフラインに失敗」に変わります (これにより、リソースが破壊され、事前に手操作による介入なしにはそのノードで開始できないことが System Automation for Multiplatforms に通知される)

同様に、これは、リソースの OpState がその期間「オフラインの保留中」に設定される StopCommand の実行に適用されます。 起こりえる例外は以下のとおりです。

- MonitorCommand は「オフライン」または「オフラインに失敗」に戻ります (つまり、自動化の目的は既に到達していることを示します)
- MonitorCommand は「オンライン中」に戻ります (つまり、リソースは System Automation for Multiplatforms によって停止できないため、手操作による介入が必要であることを示します)

StartCommand
System Automation for Multiplatforms の動作は、StartCommand を実行中に OpState が変更されると影響を受けます。

このセクションの表に、StartCommand の実行中の OpState の 変更に対する System Automation for Multiplatforms の対応について示します。MonitorCommand が OpState の変更を報告できるシチュエーションとして、以下の 3 つが考えられます。

1. StartCommand がまだ実行中である (長期実行中の StartCommand)。
2. StartCommand が正常に完了した (正常な状態)。
3. StartCommand がエラーとともに完了したか、タイムアウトになった。

StartCommand がまだ実行中である

表 *16. StartCommand* がまだ実行中のときの *System Automation* のアクション

StartCommand	**MonitorCommand**	**System Automation のアクション**
StartCommand は開始したが、完了していない。	RC=0 (不明)	OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=1 (オンライン)	他のリソースを開始する (ある場合)
	RC=2 (オフライン)	OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=3 (オフラインに失敗)	リソースは別のノードにフェイルオーバーされ、その他の従属リソースは強制停止されることがある
	RC=4 (オンライン中)	無効である可能性がある。 OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=5 (オンラインの保留中)	OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=6 (オフラインの保留中)	無効である可能性がある。 OpState は「オンラインの保留中」、「オンラインを待機」に設定される

注 :

1. MonitorCommand によりリソースが「オンライン」であることが報告される場合は、リソースをオンラインにするという目的が達成されているため、System Automation for Multiplatforms は StartCommand の戻りコードを無視します。
2. MonitorCommand によりリソースが「オフラインに失敗」であることが報告される場合は、フェイルオーバーが行われます。可能性のある他のすべての OpState 値は「オンラインの保留中」にマップされ、System Automation は、OpState が「オンライン」に変わるのを待ちます。

StartCommand が正常に完了した

以下の表に、StartCommand が正常に完了したときに実行される標準的な自動化アクションを示します。

表 *17. StartCommand* が正常に完了した後の *System Automation* のアクション		
StartCommand	**MonitorCommand**	**System Automation のアクション**
RC=0 (成功)、実際の再試行数 < RetryCount (samctrl)	RC=0 (不明)	OpState は、新規の状態情報を受信するまで「不明」に設定される
	RC=1 (オンライン)	OpState は「オンライン」に設定される。開始コマンドの処理は完了したと見なされる。
	RC=2 (オフライン)	OpState は「オンラインの保留中」、「オンラインを待機」に設定される オンライン・タイムアウトにより以下のアクションが起動される。 1. リセット操作が実行される。 2. 正常にリセットされると、開始が試行される。 3. 再試行回数が増加する。
	RC=3 (オフラインに失敗)	OpState は「オフラインに失敗」に設定される。開始コマンドの処理は完了したと見なされる。
	RC=4 (オンライン中)	無効である可能性がある。 OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=5 (オンラインの保留中)	OpState は「オンラインの保留中」、「オンラインを待機」に設定される
	RC=6 (オフラインの保留中)	無効である可能性がある。 OpState は「オンラインの保留中」、「オンラインを待機」に設定される
RC=0 (成功)、実際の再試行数 = RetryCount (samctrl)、オンラインのタイムアウト後		リソースは「オフラインに失敗」に設定され、停止コマンドがリソースに対して発行される。その後、このリソースは、別のノードにフェイルオーバーされ、その他の従属リソースは強制停止されることがある

StartCommand がエラーとともに完了したか、タイムアウトになった

以下の表に、リソースの StartCommand がエラーを戻すかタイムアウトになる場合に実行される標準的な自動化アクションを、リソースの OpState 別に示します。

表 *18. StartCommand* がエラーとともに完了したか、タイムアウトになった場合の *System Automation* のアクション

MonitorCommand	**StartCommand**	**System Automation のアクション**
RC=0 (不明)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	StartCommand が 1 を戻す場合は、リソースは即時に別のノードにフェイルオーバーされる。MonitorCommand の戻りコードは無視される。
RC=1 (オンライン)	RC=1 (ゼロ以外)、失敗	MonitorCommand が 1 (オンライン) を戻す場合は、StartCommand の戻りコードは無視される。これ以上のアクションは行われない。リソースはオンラインの状態を保つ。

表 *18. StartCommand* がエラーとともに完了したか、タイムアウトになった場合の *System Automation* のアクション *(*続き*)*

MonitorCommand	**StartCommand**	**System Automation のアクション**
RC=1 (オンライン)	StartCommand タイムアウト	MonitorCommand が 1 (オンライン) を戻す場合は、StartCommand のタイムアウトにより、StopCommand の実行もその他の自動化アクションも起動されない。 ただし、System Automation は、StartCommand をタイムアウト時に強制終了します。これによって、このとき StartCommand スクリプトからリソース・プロセスを正常に切り離すことができないと、このリソース・プロセス自体も強制終了されることがあります。この場合、リソースの OpState は再度「オフライン」に変更されます。MonitorCommand によって、リソースがオフラインであることが検出されるとすぐに、System Automation は同じノードでリソースを再始動します。 StartCommand が再度タイムアウトになると、この動作の結果としてループに陥ることがあることに注意してください。そのような場合は、リソースの OpState が「オンライン」に変わると内部カウンターはリセットされるため、RetryCount の影響はありません。 ヒント: 通常、ループは StartCommand 内のエラーの結果として発生します。ただし、アプリケーション・プロセスが予期せず強制終了された場合に、リソースの MonitorCommand が「オフラインに失敗」を戻すと、ループを防ぐことができます。この場合は、引き継ぎが行われ、リソースは別のノードで開始されます。 IBM.Application リソースの StartCommand を処理する方法の詳細については、108 ページの『グローバル・リソース・マネージャーの使用』を参照してください。

表 *18. StartCommand* がエラーとともに完了したか、タイムアウトになった場合の *System Automation* のアクション *(*続き*)*

MonitorCommand	**StartCommand**	**System Automation のアクション**
RC=2 (オフライン)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	StartCommand が RC=1 とともに戻された直後、リソースは System Automation for Multiplatforms によって停止され、フェイルオーバーが実行される。 開始コマンドがタイムアウトになった場合は、そのプロセスは killpg() で強制終了される (108 ページの『グローバル・リソース・マネージャーの使用』も参照)。
RC=3 (オフラインに失敗)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	MonitorCommand が以前に 3 (「オフラインに失敗」) を戻している場合は、フェイルオーバーが既に発生しており、StartCommand の実行不成功によるそれ以上の影響はない。
RC=4 (オンライン中)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	無効である可能性がある。StartCommand が RC=1 とともに戻された直後、リソースは System Automation for Multiplatforms によって停止され、オペレーター・アクションを待つ。
RC=5 (オンラインの保留中)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	StartCommand が RC=1 とともに戻された直後、リソースは System Automation for Multiplatforms によって停止され、リソースがオフラインであることが報告された後、フェイルオーバーが実行される。
RC=6 (オフラインの保留中)	RC=1 (ゼロ以外)、失敗、またはタイムアウト	無効である可能性がある。StartCommand が RC=1 とともに戻された直後、リソースは System Automation for Multiplatforms によって停止され、リソースがオフラインであることが報告された後、フェイルオーバーが実行される。

注 : MonitorCommand によりリソースが「オンライン」であることが報告される場合は、StartCommand の戻りコードは無視されます。この場合、2 つのコマンドの結果は矛盾します。

- StartCommand は、リソースの開始は失敗したことを示します。
- MonitorCommand は、リソースが「オンライン」であることを報告します。

StopCommand
System Automation for Multiplatforms の動作は、StopCommand を実行中に OpState が変更されると影響を受けます。

このセクションの表に、StopCommand の実行中の OpState の変更に対する System Automation for Multiplatforms の対応について説明します。StopCommand は、System Automation の停止方法とリセット

方法を参照します。つまり、どちらの場合も StopCommand が実行されます。同様に、StopCommandTimeout と ResetCommandTimeout の両方のタイムアウト値が、同じタイムアウト値を参照します。

MonitorCommand が OpState の変更を報告できるシチュエーションとして、以下の 3 つが挙げられます。

1. StopCommand がまだ実行中である (長期実行中の StopCommand)。
2. StopCommand が正常に完了した (正常な状態)。
3. StopCommand がエラーとともに完了したか、タイムアウトになった。

StopCommand がまだ実行中である

表 *19. StopCommand* がまだ実行中のときの *System Automation* のアクション

StopCommand	**MonitorCommand**	**System Automation のアクション**
StopCommand は開始したが、完了していない。	RC=0 (不明)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=1 (オンライン)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=2 (オフライン)	他のリソースの停止を続行する
	RC=3 (オフラインに失敗)	他のリソースの停止を続行する
	RC=4 (オンライン中)	オペレーター・アクションを待つ
	RC=5 (オンラインの保留中)	無効である可能性がある。 OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=6 (オフラインの保留中)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される

MonitorCommand によりリソースが「オフライン」または「オフラインに失敗」であることが報告されると、リソースをオフラインにするという目的は既に達成されているため、System Automation for Multiplatforms は StopCommand の戻りコードを無視します。

リソースの OpState が「オンライン中」と報告されると、リソースは停止できないため、System Automation for Multiplatforms は、オペレーターによる介入を待ちます。 可能性のある他のすべての OpState 値は「オフラインの保留中」にマップされ、System Automation for Multiplatforms は、OpState が「オフライン」に変わるのを待ちます。

StopCommand が正常に完了した

以下のテーブルに、System Automation for Multiplatforms の標準的な動作を示します。

表 *20. StopCommand* が正常に完了した後の *System Automation* のアクション		
StopCommand	**MonitorCommand**	**System Automation のアクション**
RC=0 (成功) および ((「オフライン」タイムアウトに達しない) または (リソースに対してリセットが送られ、リセット・タイムアウトに達しない))	RC=0 (不明)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=1 (オンライン)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=2 (オフライン)	OpState は「オフライン」に設定される。StopCommand の処理は完了したと見なされる。
	RC=3 (オフラインに失敗)	OpState は「オフラインに失敗」に設定される。StopCommand の処理は完了したと見なされる。
	RC=4 (オンライン中)	OpState は「オンライン中」に設定される。StopCommand の処理は完了し、オペレーターのアクション待ちと見なされる。
	RC=5 (オンラインの保留中)	無効である可能性がある。 OpState は「オフラインの保留中」、「オフラインを待機」に設定される
	RC=6 (オフラインの保留中)	OpState は「オフラインの保留中」、「オフラインを待機」に設定される
RC=0 (成功) および 「オフライン」タイムアウト後		OpState は「オフラインの保留中」に設定され、リソースに対して「リセット」が送られる
RC=0 (成功) および リセット・タイムアウト後		OpState は「オンライン中」、「オペレーターのアクション待ち」に設定される

StopCommand がエラーとともに完了したか、タイムアウトになった

以下の表に、StopCommand がエラーとともに完了したか、タイムアウトになった場合に行われる自動化アクションをリストします。アクションは、StopCommand が停止操作で呼び出されたか、またはリセット操作で呼び出されたかによって異なります。

表 *21. StopCommand* がエラーとともに完了したか、タイムアウトになった場合の *System Automation* のアクション

StopCommand	**リソースの現行 OpState**	**自動化アクション**
RC=1 (ゼロ以外。失敗、またはタイムアウト) および ((「オフライン」タイムアウトに達しない) または (リソースに対してリセットが送られ、リセット・タイムアウトに達しない))	RC=0 (不明)	リセット順序を即時に送信する
	RC=1 (オンライン)	リセット順序を即時に送信する
	RC=2 (オフライン)	他のリソースの停止を続行する
	RC=3 (オフラインに失敗)	他のリソースの停止を続行する
	RC=4 (オンライン中)	リセット順序を即時に送信する
	RC=5 (オンラインの保留中)	リセット順序を即時に送信する
	RC=6 (オフラインの保留中)	リセット順序を即時に送信する
RC=1 (不成功) および 「オフライン」タイムアウト後		OpState は「オフラインの保留中」に設定され、リソースに対して「リセット」が送られる
RC=1 (不成功) および リセット・タイムアウト後		OpState は「オンライン中」、「オペレーターのアクション待ち」に設定される

注 :

1. リソースの OpState は、その OpState が「オフライン」、「オフラインの保留中」、または「オンライン中」に変わるまで、「オフラインの保留中」に設定されます。 OpState が、「オフライン」タイムアウト期間内に「オフライン」または「オフラインに失敗」に変わらない場合、System Automation for Multiplatforms はリソースに対してリセット操作を行い、その結果、StopCommand が実行されます。 OpState が、リセット・タイムアウト期間内に依然として「オフライン」、「オフラインに失敗」、または「オンライン中」に変わらない場合、リソースは、最終的に「オンライン中」に設定され、オペレーターがリソース停止のアクションを取る必要があることを示します。 StopCommand またはリセット操作の戻りコードが 0 (ゼロ) ではない場合は、リソースでは、手操作による介入が必要な自動化状態の「問題」が発生して終了します。これを解決するには、**resetrsrc** を使用してリソースをリセットする必要があります。ただし、このリセット操作の戻りコードは、0 (ゼロ) にする必要があります。そうしないと、リソースでは自動化状態の「問題」が発生したままになります。
2. オーダーがリセットだった場合は、リソースは「リセット・オーダーの送信」の代わりに「オンライン中」に設定されます。
3. StartCommand の失敗への応答として実行されている StopCommand が失敗する場合は、リソースに対してリセットは送信されません。これは、無限ループを引き起こす恐れがある ためです。リセットによってリソースの OpState が変更されてオフラインに戻るため、リソースに対して新しい StartCommand が実行され、これもおそらく失敗し、次の StopCommand が実行されるという形で、ループが発生します。

別のノードでリソースの *OpState* が報告される場合のオンライン・リソースに 対する *System Automation* の動作

以下の表に、1 つのノードでリソースがオンラインであり、MonitorCommand が別のノード上のこのリソースの特定の OpState を戻す場合に、System Automation for Multiplatforms が実行するアクションを示します。

表 *22. MonitorCommand* が別のノード上にあるリソースの *OpState* の変更を報告した後の *System Automation* のアクション

スタンバイ・ノードの MonitorCommand	浮動リソースに対する System Automation のアクション	並行リソースに対する System Automation のアクション
RC=0 (不明)	MonitorCommand が RC<>0 で戻るまで、このリソースに対する自動化アクションは不可能。	MonitorCommand が RC<>0 で戻るまで、このリソースに対する自動化アクションは不可能。
RC=1 (オンライン)	1. 両方のノードのリソースが停止される。 これは、同様に停止される以下のリソースに影響します。 • dependsOn 関係または forcedDownBy 関係によってリンクされている従属リソース • その他のグループ・メンバー 停止されるリソースのスコープは、バインド・プログラムの実行のスコープで判別できます。つまり、関係 (グループ・メンバーシップを含む) によって接続されているリソースはすべて停止されます (56 ページの『ノード位置の割り当て』を参照)。 2. リソースはいずれかのノード上で再開される。	アクションなし。そのリソースの通常の OpState。
RC=2 (オフライン)	アクションなし。スタンバイ・ノード上のリソースの通常の OpState。	System Automation は、このリソースの StartCommand を呼び出す。
RC=3 (オフラインに失敗)	アクションなし。ただし、このノードに対する以降のフェイルオーバーは不可能。	アクションなし。ただし、このノードに対する以降のフェイルオーバーは不可能。
RC=4 (オンライン中)	無効である可能性がある。スクリプト・エラー (事前にオンライン OpState が必要)。	アクションなし。ただし、リソースの本来あるべき状態がオフラインに変わる場合は、このリソースに対する StopCommand は実行されない。
RC=5 (オンラインの保留中)	オンラインと同様。	オンラインと同様。
RC=6 (オフラインの保留中)	無効である可能性がある。スクリプト・エラー (事前にオンライン OpState が必要)。	アクションなし。ただし、停止が完了すれば、「オフライン」用のアクションが 実行される。
RC=8 (不適格)	アクションなし。ただし、このノードに対する以降のフェイルオーバーは不可能。	アクションなし。ただし、このノードに対する以降のフェイルオーバーは不可能。

System Automation for Multiplatforms の制御下にあるアプリケーションおよびリソースを手動で開始も停止もしないでください。同時に複数のノード上でオンラインであることを System Automation for Multiplatforms がモニター した場合は、スタンバイ・ノード上のリソースのみでなく、両方のノードの集

合リソース が停止されます。この集合リソースに対する依存関係があるすべての リソースも停止される可能性があります。例えば、この集合リソースに対する DependsOn 関係を持つすべてのリソースも停止されます。

コンポーネント

本製品での作業を開始する前に、System Automation for Multiplatforms の用語、概念、およびコンポーネントについて十分理解しておく必要があります。

クラスターおよびピア・ドメイン

クラスターおよびピア・ドメインは、高可用性用に構成されるホスト・システムのグループを示す、相互に交換可能な用語として使用します。ここでは、System Automation for Multiplatforms は、リソースの自動化と管理のために使用されます。クラスターは、単一ノードまたは複数ノードから構成されます (ノードとシステムも相互に交換可能な用語として使用します)。

クラスターのノード数:

Linux
単一クラスターのノードの最大数は 32 です。

AIX
単一クラスターのノードの最大数は 130 です。

Resource

リソースは、System Automation for Multiplatforms に定義できる任意のハードウェアまたはソフトウェアのコンポーネントです。 さまざまな方法で、System Automation for Multiplatforms にリソースを定義できます。

- コマンド行から **mkrsrc** (「make resource」) コマンドを使用して定義
- XML 自動化ポリシー・ファイル内に定義
- クラスター・インフラストラクチャーのリソース取得機能によって自動的に定義 (一部のリソース)

リソースは、適切なリソース・マネージャーによって制御されます (2 ページの『RSCT および System Automation for Multiplatforms によって提供されるリソース・マネージャー』を参照)。ご使用のリソースの属性と特性を定義できます。 例えば、IP アドレスを表すリソースの属性には、IP アドレスそのものとネットマスクが含まれます。

以下に示す 3 つのタイプのリソースがあります。

固定リソース
クラスター内に単一のインスタンスのみを持つリソースです。これは単一のノードに対して定義される 1 つのエンティティーを表し、そのノードでのみ実行されます。

浮動リソース
クラスター内の複数ノードで実行できますが、一度には 1 つのノードでしか実行できないリソースです (15 ページの『ResourceType 属性』を参照)。

並行リソース
クラスター内の複数ノードで実行でき、一度に複数のノードで実行できるリソースです (15 ページの『ResourceType 属性』を参照)。

リソース属性

リソース属性とは、リソースの特性を意味します。次に示す 2 つのタイプのリソース属性があります。

永続属性
. これらの特性は、一般的に継続的で不変であり、リソースを一意に識別します。 IP アドレスの属性 (IP アドレスそのものおよびネットマスク) は、永続属性の例です。永続属性は、リソースの永続する特性を意味します。IP アドレスおよびネットマスクは変更できますが、これらの特性は一般的に継続的で不変です。

動的属性
動的属性はリソースの変化する特性を示します。例えば、IP アドレスの動的属性は、その操作状態 などを示します。

リソース・クラス

リソース・クラスは、同じタイプのリソースの集合です。例えば、アプリケーションがリソースの場合、クラスター内で定義されるすべてのアプリケーションは、1 つのリソース・クラスを構成します。リソース・クラスを使用すると、同じクラスのメンバーに共通の特性を定義できます。 アプリケーションの場合、リソース・クラスには、アプリケーションの名前などの固有の特性、およびアプリケーションが稼働中であるかどうかなどの可変特性が保管されます。 したがって、クラス内の各リソースを、その特性によりいつでも特定できます。リソース・クラスは、さまざまなリソース・マネージャーによって 管理されます。2 ページの『RSCT および System Automation for Multiplatforms によって提供されるリソース・マネージャー』を参照してください。

リソース・グループ

リソース・グループは、リソースの集合の論理コンテナーです。リソース・グループを使用すると、複数のリソースを単一の論理エンティティーとして制御できます。 リソース・グループは、System Automation for Multiplatforms におけるオペレーションの主要メカニズムです。リソース・グループは ネストすることもできます。 これは、アプリケーションを、より上位の別のリソース・グループの一部である 複数のリソース・グループに分割できることを意味します。また、リソース・グループのメンバーをクラスター内の異なるシステムに配置できるように、リソース・グループを定義することもできます。

管理対象リソース

管理対象リソースは、System Automation for Multiplatforms に定義されているリソースです。リソースは、リソース・グループまたはそれと同等のものに追加されるか、管理対象関係のターゲットとして定義されることによって、管理対象リソースになります。

公称状態

リソース・グループの公称状態は、System Automation for Multiplatforms に対して、このグループのリソースが現在オンラインであるべきかオフラインであるべきかを示します。 公称状態を「オフライン」に設定すると、System Automation for Multiplatforms に対してグループ内のリソースの停止を要求していることが示され、公称状態を「オンライン」に設定すると、System Automation for Multiplatforms に対してリソース・グループ内のリソースの開始を要求していることが示されます。 `NominalState` リソース・グループ属性の値は変更できますが、 リソースの公称状態を直接設定することはできません。23 ページの『NominalState 属性』を参照してください。

同値

同値は、同じ機能を提供するリソースの 集合です。例えば、1 つの IP アドレスをホスティングする複数のネットワーク・アダプターを選択するために同値を使用します。1 つのネットワーク・アダプターがオフラインになると、System Automation for Multiplatforms は、当該 IP アドレスをホスティングする別のネットワーク・アダプターを選択します。

関係

System Automation for Multiplatforms では、クラスター内のリソース間の関係を定義できます。以下の 2 種類の関係があります。

開始または停止依存関係

リソース間の 開始および停止の依存関係を定義するために関係が使用されます。これを遂行する ために、StartAfter、StopAfter、DependsOn、DependsOnAny、および ForcedDownBy の各関係を 使用できます。例えば、あるリソースを、他のリソースが開始された後にのみ開始する必要がある場合、ポリシー要素の StartAfter 関係を使用してこのシーケンスを定義できます。

ロケーション関係

ロケーション関係は、リソースをクラスター内の同一ノードまたは異なるノードで開始する必要がある場合に適用されます。System Automation for Multiplatforms では、 ロケーション関係として Collocation、AntiCollocation、Affinity、AntiAffinity、 および IsStartable が提供されています。簡単な例を挙げると、クラスター内の任意のノードで始動可能な Web サーバーとそれに対応するサービス IP アドレスは、常に同じ場所に保管する必要があります。以前は、この動作は複雑なスクリプトを記述して定義する必要がありました。現在では、System Automation for Multiplatforms によりロケーション関係が使用できるようになり、これにより管理者はポリシーを簡単に定義できます。

関係により、以下の追加機能が 提供されます。

- リソース・グループ、リソース、および同値間の関係を 定義できる。
- クラスター内の異なるシステムで稼働するリソース間の関係を 定義できる。

クォーラム

クォーラム操作の主要な目的は、データの一貫性を保持すること、およびクリティカル・リソースを 保護することです。クォーラムは、クラスター定義の変更または特定の クラスター操作の実行に必要なクラスター内のノード数として見ることが できます。以下の 2 つのタイプのクォーラムがあります。

構成クォーラム

構成クォーラムは、クラスター内の構成変更がいつ受け入れられるのかを決定します。クラスターまたはリソースの構成に影響を与える操作は、 多数のノードがオンラインである場合にのみ実行可能です。詳しくは、5 ページの『構成クォーラム』を参照してください。

操作クォーラム

操作クォーラムは、他のリソースとの競合を発生させることなく、リソースを安全にアクティブにすることができるかどうかを決定するために使用します。クラスター分割がある場合、大多数のノードを持つサブクラスター、またはタイ・ブレーカーを取得したサブクラスターでのみリソースを開始できます。詳しくは、5 ページの『操作クォーラム』を参照してください。

タイ・ブレーカー

タイ・ブレーカーは、操作クォーラムを持たせるサブクラスターを決定するために使用します。

ノードが偶数のクラスターが 2 つのサブクラスターに区画化され、その各サブクラスターにある定義済みノードがちょうど半分の数である場合、タイ・ブレーカーは、いずれのサブクラスターに操作クォーラムを持たせるかを決定するのに使用されます。

エンドツーエンド自動化アダプター

System Automation for Multiplatforms のエンドツーエンド自動化アダプターは、自動化ドメインと System Automation Application Manager のエンドツーエンド自動化管理サーバー間の通信を確立および管理するために必要です。 アダプターは、自動的に System Automation for Multiplatforms にインストールされます。

第 2 章 チュートリアル

System Automation for Multiplatforms での作業を開始するために、以下のトピックで説明されているタスクを使用できます。

コマンド行インターフェースを使用するように非 root ユーザー ID を構成しない限り、すべての RSCT および System Automation for Multiplatforms コマンドは、root 権限を使用してシェルから発行する必要があります。詳しくは、「*System Automation for Multiplatforms* インストールと構成のガイド」を参照してください。

RSCT コマンドの詳細については、該当するマニュアル・ページおよび Reliable Scalable Cluster Technology Knowlege Center の RSCT 資料を参照してください。

System Automation for Multiplatforms コマンドの詳細な説明については、「*Tivoli System Automation for Multiplatforms* リファレンス・ガイド」を参照してください

RSCT 資料は、System Automation for Multiplatforms DVD に入っています。

RSCT のリソースの定義

ステップ 2 では、RSCT に認識される必要があるリソースをベースにして、System Automation for Multiplatforms がその機能を処理する方法を示します。さまざまなタイプのリソースを区別するために、各リソースは 1 つのリソース・クラスに属しています。

このタスクについて

特定のタイプのリソースの取得機能は、ノードおよびクラスターの始動時および実行時間中に自動的に処理されます。そのため、それらのリソースは RSCT に既に認識されています。これらの取得対象のリソースは、オペレーティング・システム環境をスキャンすることによって検出されます。通常、これらはハードウェア関連のリソース (例えば、`IBM.NetworkInterface` または `IBM.Disk` クラス内のリソース) です。

System Automation for Multiplatforms によって自動化されるアプリケーションのために、クラス `IBM.Application` 内にユーザー定義リソースを作成する必要があります。必要なアプリケーション・リソースを定義することは、自動化ポリシーを作成するために不可欠な作業です。

RSCT リソースを定義するコマンド

リソースを定義するために RSCT コマンドが使用されます。

このタスクについて

`mkrsrc` または `rmrsrc`
指定したリソース・クラスのリソースを作成または削除します。

`chrsrc`
指定されたリソース・クラス内のリソースの永続属性値を変更する。

`lsrsrc`
リソース・クラスのリソースをリストする。

`resetrsrc`
指定されたリソース・クラスのリソースをリセットする。

`startrsrc` または `stoprsrc`
リソースを開始または停止します。通常、自動化ポリシー内に含まれるリソースは、本来あるべき状態を設定することにより、自動的に開始および停止されます。

コマンドについて詳しくは、以下を参照してください。

- *System Automation for Multiplatforms* リファレンス・ガイド
- Reliable Cluster Software Technology Knowledge Center

.

高可用性 Web サーバーに対する RSCT リソースの定義

高可用性 Web サーバーおよび必要な RSCT リソースの要件について説明します。

このタスクについて

以下の例は、高可用性 Web サーバーに対して必要なリソースの定義を示しています。SA_Domain という名前のクラスターは、147 ページの『クラスターの作成』で以前に作成されています。

高可用性 Web サーバーの要件:

- Web サーバーは任意のクラスター・ノード上で始動できますが、どの時点でも 1 つのノード上でしか稼働できません。
- 障害発生時には、Web サーバーはクラスター内の同一ノードまたは別ノードで自動的に再始動されます。サービス期間中および保守期間中のノードの計画停止には、Web サーバーのこの再配置機能を使用します。
- 現在稼働しているノードに関係なく、Web サーバーは同じ IP アドレスを使用してアドレス指定可能でなければなりません。したがって、Web サーバーが別のノードへ移動した場合に、適合を処理してはならないクラスターの外部では、Web サーバーのロケーションは分かりません。

RSCT では、異なるリソース・クラスに属する 3 つの RSCT リソースが必要です。

1. apache2 という名前のアプリケーション・リソース。Web サーバー・デーモンを表します。リソース apache2 はクラス IBM.Application に属し、浮動リソースとして定義されます。浮動リソースとは、リソースをクラスター・ノード間で移動できることを意味する用語です。
2. apache2IP という名前の新規の追加 IP アドレス。Web サーバー IP アドレスを表します。リソース apache2IP は IBM.ServiceIP というクラスに属し、Web サーバーのロケーションに従って移動する必要があるため、アプリケーション・リソースと同様に浮動リソースである必要があります。

   ネットワーク・インターフェース上で追加 IP アドレスとして IBM.ServiceIP アドレスの別名が割り当てられます。
3. IBM.ServiceIP アドレス apache2IP をホストできる各ノード上で、1 つのネットワーク・インターフェースが選択される必要があります。ネットワーク・インターフェースはクラス IBM.NetworkInterface に属しています。RSCT によって自動取得されるため、ユーザーが明示的に定義しないでください。

アプリケーション・リソース apache2 の作成

アプリケーション・リソースを定義するためのコマンドまたはスクリプトについて説明します。

このタスクについて

アプリケーション・リソース apache2 を定義するときには、ユーザーが以下の 3 つのコマンドまたはスクリプトを指定する必要があります。

- アプリケーションを開始します。
- アプリケーションを停止します。
- アプリケーションの状況を照会します。

これらのコマンドおよびスクリプトは、それぞれ別個であってもかまいません。これらの関数は、開始、停止、または状況のアクションを選択するコマンド行パラメーターがある 1 つのスクリプトにまとめることができます。多くの場合、これらのスクリプトはユーザーが作成します。スクリプトの要件について詳しくは、108 ページの『IBM.Application リソース・クラス』を参照してください。

この例では、以下のスクリプトを使用します。

```
/cluster/scripts/apache2
```

このスクリプトは、Linux システムを対象とした以下のような内容になっています。

```
#!/bin/bash

OPSTATE_ONLINE=1
OPSTATE_OFFLINE=2

Action=${1}

case ${Action} in
        start)
                /usr/sbin/apache2ctl start >/dev/null 2>&1
                logger -i -t "SAM-apache" "Apache started"
                RC=0
                ;;
    stop)
                /usr/sbin/apache2ctl stop >/dev/null 2>&1
                logger -i -t "SAM-apache" "Apache stopped"
                RC=0
                ;;
    status)
                ps ax |grep -v "grep"|grep "/usr/sbin/httpd">/dev/null
                if [ $? == 0 ]
                then
                    RC=${OPSTATE_ONLINE}
                else
                    RC=${OPSTATE_OFFLINE}
                fi
                ;;
esac
exit $RC
```

注 : このスクリプトには、すべてのノード上で同じディレクトリー・パスを使用してアクセスできる必要があります。RSCT Web サーバー・リソースを作成する前に、Web サーバーが実行されるノードにスクリプトを配布します。start/stop/status スクリプトの存在は、各ノード上で検査されます。

アプリケーション・リソース apache2 の RSCT リソース定義は、mkrsrc コマンドを使用して作成されます。すべてのリソース特性をコマンド行パラメーターとして渡すことができますが、mkrsrc コマンドは、プレーン・テキスト形式の定義ファイルも受け入れます。

以下の例のように、apache2.def という名前の定義ファイルを使用した 2 番目の方法を使用します。

```
PersistentResourceAttributes:
   Name="apache2"
   StartCommand="/cluster/scripts/apache2 start"
   StopCommand="/cluster/scripts/apache2 stop"
   MonitorCommand="/cluster/scripts/apache2 status"
   MonitorCommandPeriod=5
   MonitorCommandTimeout=5
   NodeNameList={"node01","node02","node03"}
   StartCommandTimeout=10
   StopCommandTimeout=10
   UserName="root"
   ResourceType=1
```

ここで、mkrsrc コマンドで定義ファイルのロケーションと名前を指定して、リソース定義を作成できます。

```
mkrsrc -f apache2.def IBM.Application
```

このコマンドは、実行が成功した場合に出力を返しません。以下の **lsrsrc** コマンドを入力して、apache2 の浮動リソース表現を表示できます。

```
lsrsrc -s "Name='apache2' && ResourceType=1" IBM.Application
Resource Persistent Attributes for IBM.Application
resource 1:
            Name                  = "apache2"
            ResourceType          = 1
            AggregateResource     = "0x3fff 0xffff 0x00000000 0x000000000x00000000 0x00000000"
            StartCommand          = "/cluster/scripts/apache2 start"
            StopCommand           = "/cluster/scripts/apache2 stop"
            MonitorCommand        = "/cluster/scripts/apache2 status"
            MonitorCommandPeriod  = 5
            MonitorCommandTimeout = 5
```

```
        StartCommandTimeout   = 10
        StopCommandTimeout    = 10
        UserName              = "root"
        RunCommandsSync       = 1
        ProtectionMode        = 0
        HealthCommand         = ""
        HealthCommandPeriod   = 10
        HealthCommandTimeout  = 5
        InstanceName          = ""
        InstanceLocation      = ""
        SetHealthState        = 0
        MovePrepareCommand    = ""
        MoveCompleteCommand   = ""
        MoveCancelCommand     = ""
        CleanupList           = {}
        CleanupCommand        = ""
        CleanupCommandTimeout = 10
        ProcessCommandString  = ""
        ResetState            = 0
        ReRegistrationPeriod  = 0
        CleanupNodeList       = {}
        MonitorUserName       = ""
        ActivePeerDomain      = "SA_Domain"
        NodeNameList          = {"node01","node02”,”node03"}
```

オペレーティング・システム環境 (つまり、Linux のラン・レベル設定) 内の他のメカニズムによって Web サーバーが始動されないようにしてください。アプリケーションが既に開始済みで使用中であるときにリソースを定義しないでください。

コマンド出力に表示されているリソース属性について詳しくは、108 ページの『IBM.Application リソース・クラス』を参照してください。

IP アドレス・リソース apache2IP の作成

このタスクについて

Web サーバー IP アドレス apache2IP は、クラスター内でユーザーが選択する独自の IP アドレスであり、ネットワーク内で既に割り当て済みの IP アドレスと一致させることはできません。例えば、すべてのクラスター・ノードが同じ IP サブネットに属している場合、IP アドレス apache2IP は、同じサブネット内の未使用のアドレスにすることができます。

apache2IP のアドレスは、Web サーバーを実行するために選択されたノード上のネットワーク・アダプターに別名アドレスとして追加されます。Web サーバーを新しい位置に移動する必要がある場合は、移動前のノードから別名アドレスが除去され、Web サーバーが再始動される新規ノードで別名アドレスが作成されます。

IP アドレス・リソース apache2IP には以下の属性があります。

- IP 10.152.172.11
- ネットマスク 255.255.255.0
- apache2IP リソースの NodeNameList は、Web サーバー・アプリケーション・リソース apache2 の NodeNameList と同じです。

mkrsrc コマンドを使用して apache2IP リソースを作成します。

```
mkrsrc IBM.ServiceIP ¥
          NodeNameList="{’node01’,’node02’,’node03’}" ¥
          Name="apache2IP" ¥
          NetMask=255.255.255.0 ¥
          IPAddress=10.152.172.11 ¥
          ResourceType=1
```

読みやすくするためにのみ、コマンドを個別の行に分割しています。以下の **lsrsrc** コマンドを入力して、浮動 ServiceIP アドレス apache2IP を表示できます。

```
lsrsrc -s "Name='apache2IP' && ResourceType=1" IBM.ServiceIP
resource 1:
      Name             = "apache2IP"
```

```
ResourceType      = 1
AggregateResource = "0x3fff 0xffff 0x00000000 0x00000000 0x00000000 0x00000000"
IPAddress         = "10.152.172.11"
NetMask           = "255.255.255.0"
ProtectionMode    = 1
NetPrefix         = 0
ActivePeerDomain  = "SA_Domain"
NodeNameList      = {"node01","node02","node03"}
```

コマンド出力に表示されているリソース属性について詳しくは、124 ページの『IBM.ServiceIP リソース・クラス』を参照してください。

ServiceIP アドレス apache2IP をホスティングするためのネットワーク・インターフェース

クラスター・ノード上のネットワーク・インターフェースは、RSCT によって自動的に取得されるため、**mkrsrc** コマンドを使用して明示的に定義する必要はありません。

対応する RSCT リソースは、クラス IBM.NetworkInterface に属していて、**lsrsrc** コマンドを使用して表示できます。

```
lsrsrc IBM.NetworkInterface
Persistent Attributes for IBM.NetworkInterface
resource 1:
        Name             = "eth0"
        DeviceName       = ""
        IPAddress        = "10.152.172.183"
        SubnetMask       = "255.255.255.0"
        Subnet           = "10.152.172.0"
        CommGroup        = "CG1"
        HeartbeatActive  = 1
        Aliases          = {}
        DeviceSubType    = 1
        LogicalID        = 0
        NetworkID        = 0
        NetworkID64      = 0
        PortID           = 0
        HardwareAddress  = "12:7e:8d:61:41:02"
        DevicePathName   = ""
        IPVersion        = 4
        Role             = 0
        ActivePeerDomain = "SA_Domain"
        NodeNameList     = {"node01"}
resource 2:
        Name             = "eth0"
        DeviceName       = ""
        IPAddress        = "10.152.172.191"
        SubnetMask       = "255.255.255.0"
        Subnet           = "10.152.172.0"
        CommGroup        = "CG1"
        HeartbeatActive  = 1
        Aliases          = {}
        DeviceSubType    = 1
        LogicalID        = 0
        NetworkID        = 0
        NetworkID64      = 0
        PortID           = 0
        HardwareAddress  = "12:7e:8f:ed:28:02"
        DevicePathName   = ""
        IPVersion        = 4
        Role             = 0
        ActivePeerDomain = "SA_Domain"
        NodeNameList     = {"node02"}
resource 3:
        Name             = "eth0"
        DeviceName       = ""
        IPAddress        = "10.152.172.182"
        SubnetMask       = "255.255.255.0"
        Subnet           = "10.152.172.0"
        CommGroup        = "CG1"
        HeartbeatActive  = 1
        Aliases          = {}
        DeviceSubType    = 1
        LogicalID        = 0
        NetworkID        = 0
        NetworkID64      = 0
```

```
        PortID           = 0
        HardwareAddress  = "12:7e:80:0d:70:02"
        DevicePathName   = ""
        IPVersion        = 4
        Role             = 0
        ActivePeerDomain = "SA_Domain"
        NodeNameList     = {"node03"}
```

この例では、各クラスター・ノードに、共通サブネット 10.152.172 へのインターフェースがあります。ServiceIP アドレス apache2IP もこのサブネットに属しています。

Web サーバーに対して以前に定義された浮動リソースおよびそれに関連付けられたサービス IP アドレスには、それぞれに、すべてのクラスター・ノードをリストする NodeNameList 属性があります。しかし、3 つのネットワーク・インターフェースは 3 つの RSCT リソースとして表され、各リソースの NodeNameList 属性にリストされるノードは 1 つのみです。ネットワーク・インターフェース・リソースは、1 つのノード上にのみ一意的に存在するいわゆる固定リソースの一例です。

Web サーバーに対する自動化ポリシーの定義

ステップ 3 では、リソース apache2 および apache2IP が、クラスター環境内で Web サーバーを高可用性に対応させるための管理対象リソースとして定義されます。

このタスクについて

リソース apache2 および apache2IP の定義については、93 ページの『RSCT のリソースの定義』を参照してください。

System Automation for Multiplatforms は、RSCT によって提供されるインフラストラクチャーに 3 つのエレメントを追加します。

1. リソース apache2 および apache2IP は、リソース・グループに追加すると、管理対象リソースになります。リソース・グループおよびそのメンバーは、System Automation for Multiplatforms によってアクティブにモニターおよび制御されます。
2. apache2IP アドレスの別名を割り当てることができる各クラスター・ノード上のネットワーク・インターフェースの集合は、いわゆる同値によって表される必要があります。
3. ほとんどの場合、管理対象リソースのみを自動化しても十分ではありません。これは、多くの場合、リソースが相互に関連しているためです。例えば、この例のリソース apache2 と apache2IP の両方を同じノードで使用可能にして、特定のシーケンスで開始または停止する必要があります。管理対象リソース間のこのような依存関係は、管理対象関係によって記述されます。

最後に、自動化目標を設定します。例えば、管理対象リソースは、クラスター内で使用可能/開始済みまたは使用不可/停止済みになる可能性があります。

リソース・グループと同値によって構成され、関係によって結合された構成体を、自動化ポリシーと呼びます。99 ページの図 80 は、Web サーバーの例で作成される自動化ポリシーの図式の概要を示しています。

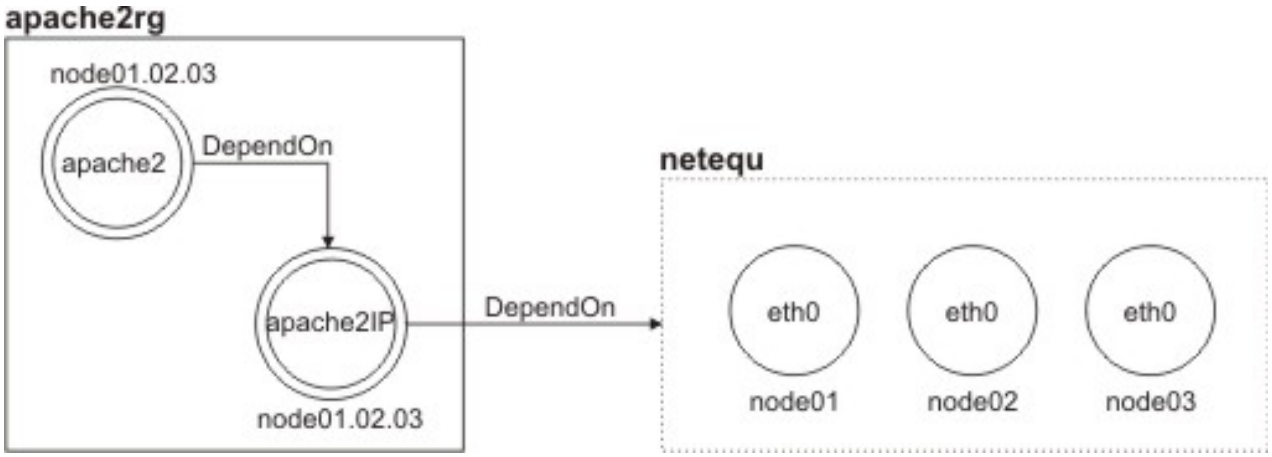


図 *80. Web* サーバー自動化ポリシー

自動化ポリシーを定義および操作するコマンド

以下のリストは、自動化ポリシーを定義および操作するために最も頻繁に使用される System Automation for Multiplatforms コマンドの概要を示しています。

mkrg
リソース・グループを作成します。

chrg
リソース・グループの永続属性値を変更します。これは、リソース・グループを開始および停止するために最も頻繁に使用されます。

lsrg
リソース・グループまたはそのメンバーの属性およびそれらの値を表示します。

rmrg
リソース・グループを削除します。

addrgmbr
1 つ以上のリソースをリソース・グループに追加します。

chrgmbr
リソース・グループ内の管理対象リソースの永続属性値を変更します。

rmrgmbr
1 つ以上のリソースをリソース・グループから削除します。

mkequ
リソースの同値を作成します。

chequ
リソースの同値の属性を変更します。

lsequ
同値およびその属性をリストします。

rmequ
1 つ以上のリソース同値を削除します。

mkrel
リソース間の管理対象関係を作成します。

chrel
リソース間の 1 つ以上の管理対象関係を変更します。

lsrel
管理対象関係をリストします。

rmrel
管理対象関係を削除します。

samctrl
System Automation for Multiplatforms 制御パラメーターを設定します。

lssamctrl
System Automation for Multiplatforms 制御パラメーターをリストします。

lssam
定義済みのリソース・グループ、そのメンバー、およびその操作状態をツリー形式でリストします。

lsrgreq
リソース・グループまたは管理対象リソースに対して適用される未解決の要求をリストします。

rgreq
リソース・グループの開始、停止、ロック、アンロック、移動を要求するか、要求をキャンセルします

rgmbrreq
管理対象リソースの開始、停止、ロック、アンロックを要求するか、要求をキャンセルします

コマンドについて詳しくは、以下を参照してください。

- *System Automation for Multiplatforms* リファレンス・ガイド
- Reliable Cluster Software Technology Knowledge Center

.

ネットワーク・アダプターの同値の作成

クラスター内のノードが複数のネットワーク接続を持つ場合、これらのすべての接続が一様に apache2IP アドレスを別名としてホスティングすることに適しているわけではありません。同値定義は、apache2IP アドレスを保持するために使用されるネットワーク・アダプターをグループ化および選択します。クラスター内の各ノードで Web サーバーが始動できるため、各ノードの 1 つ以上のアダプターが同値に示される必要があります。

ネットワーク・アダプターはクラス IBM.NetworkInterface に属します。RSCT の自動取得機能により、多数のオペレーティング・システム・リソースに対して適切なリソース定義が自動的に作成されるため、クラスター・ノード のすべてのネットワーク・アダプターにリソース定義を提供する必要はありません。

以下のコマンド (Linux の例) は、クラスター内の各ノードのネットワーク・アダプターが含まれている netequ という、いわゆる静的同値を作成します。

```
mkequ netequ IBM.NetworkInterface:eth0:node01,eth0:node02,eth0:node03
```

以下の lsequ コマンドを入力して、この同値を表示できます。

```
lsequ -e netequ
Equivalency 1:
   Name                    = netequ
   MemberClass             = IBM.NetworkInterface
   Resource:Node[Membership] = {eth0:node01,eth0:node02,eth0:node03}
   SelectString            = ""
   SelectFromPolicy        = ANY
   MinimumNecessary        = 1
   Subscription            = {}
   Color                   = 0
   ActivePeerDomain        = SA_Domain
   ConfigValidity          =
```

静的同値では、同値のメンバーは、同値の作成時に指定されます。この同値タイプの利点は、同値に含まれるリソースに対して、順序を指定できることにあります。しかし、ネットワーク・インターフェースがオペレーティング・システムから切り離されたか、その IP アドレスが削除された場合、そのインターフェースは静的同値から削除され、後でまた出現しても再度追加されることはありません。

この制限を克服するには、次のコマンド (Linux の例) を使用して、動的選択文字列で同値を作成します。

```
mkequ -D 'Name like "eth0"' netequ IBM.NetworkInterface
```

この定義には、クラスター内にあるすべての eth0 ネットワーク・インターフェースが含まれます。System Automation for Multiplatforms がリソースを開始するノードを検出する必要があるたびに、選択文字列が評

価されます。System Automation for Multiplatforms がクラスター内の新規ネットワーク・インターフェースを定期的に取得したり、未定義の (IP アドレスが削除されているかまたは切り離されている) ネットワーク・インターフェースがクラスター定義から削除されたりした場合に、使用可能な eth0 リソースのリストが変更されることがあります。動的選択文字列を使用する同値には、順序付けられたリソース・リストがありません。

動的選択文字列を使用して定義された同値の例:

```
lsequ -e netequ
Equivalency 1:
   Name                      = netequ
   MemberClass               = IBM.NetworkInterface
   Resource:Node[Membership] = {}
   SelectString              = "Name like "eth0""
   SelectFromPolicy          = ANY
   MinimumNecessary          = 1
   Subscription              = {}
   Color                     = 0
   ActivePeerDomain          = SA_Domain  ConfigValidity            =
```

コマンド出力に表示されている同値属性について詳しくは、54 ページの『同値によって使用される属性』を参照してください。

apache2 および apache2IP のリソース・グループの作成

リソース・グループを作成するには **mkrg** コマンドを使用します。

以下の例は、apache2rg という名前のリソース・グループを作成します。

```
mkrg apacherg
```

以下の **lsrg** コマンドを入力して、リソース・グループ apache2rg を表示できます。

```
lsrg -g apache2rg
Resource Group 1:
       Name                = apache2rg
       MemberLocation      = Collocated
       Priority            = 0
       AllowedNode         = ALL
       NominalState        = Offline
       ExcludedList        = {}
       Subscription        = {}
       Owner               =
       Description         =
       InfoLink            =
       Requests            = {}
       Force               = 0
       ActivePeerDomain    = SA_Domain
       OpState             = Offline
       TopGroup            = apache2rg
       ConfigValidity      =
       TopGroupNominalState = Offline
```

コマンド出力に表示されているリソース・グループ属性の説明については、20 ページの『リソース・グループによって使用される属性』を参照してください。

addrgmbr コマンドを使用して、apache2 と apache2IP の両方のリソースをリソース・グループ apache2rg に追加する必要があります。リソースをリソース・グループに追加すると、それらのリソースは管理対象リソースになります。

```
addrgmbr -g apache2rg IBM.Application:apache2
addrgmbr -g apache2rg IBM.ServiceIP:apache2IP
```

以下の **lsrg** コマンドを入力して、リソース・グループ・メンバーを表示できます。

```
lsrg -m
Class:Resource:Node[ManagedResource] Mandatory MemberOf   OpState WinSource Location
IBM.ServiceIP:apache2IP              True      apache2rg  Offline
IBM.Application:apache2              True      apache2rg  Offline
```

Web サーバー・リソース間の関係の定義

関係を定義するために必要な条件および使用されるコマンドについて説明します。

リソース apache2 と apache2IP は、以下の 2 つの条件によって相互に関連付けられます。

- 両方のリソースをクラスター内の同じノード上で使用可能にする必要があります。詳しくは、45 ページの『Collocated 関係 』を参照してください。
- Web サーバー apache2 は、同じノード上で IP アドレス apache2IP が作成された後にのみ始動できます。同様に、Web サーバーは、IP アドレス apache2IP が削除される前に停止する必要があります。

System Automation for Multiplatforms は、両方の必要条件を結合する 38 ページの『DependsOn 関係』を提供します。

管理対象関係は、**mkrel** コマンドを使用して作成されます。以下の例は、apache2_dependson_apache2IP という名前の管理対象関係を作成します。この関係により、IP アドレス apache2IP に対するリソース apache2 の依存関係が設定されます。

```
mkrel -p DependsOn -S IBM.Application:apache2 ¥
                           -G IBM.ServiceIP:apache2IP apache2_dependson_apache2IP
```

この例では、2 番目の関係が必要です。apache2IP アドレスの別名割り当てに使用できるネットワーク・アダプターの同値が既に作成されています。ネットワーク・アダプターと ServiceIP アドレスの間の関係は、apache2IP_dependson_netequ という 2 番目の関係で記述されます。この関係は、apache2IP と同値 netequ を結合します。

```
mkrel -p DependsOn -S IBM.ServiceIP:apache2IP ¥
                           -G IBM.Equivalency:netequ apache2IP_dependson_netequ
```

lsrel コマンドを使用して、定義された関係を表示できます。

```
lsrel
Name                          Class:Resource:Node[Source] ResourceGroup[Source]
apache2IP_dependson_netequ  IBM.ServiceIP:apache2IP     apache2rg
apache2_dependson_apache2IP IBM.Application:apache2     apache2rg
```

Web サーバー・リソース・グループのオンライン化

リソース・グループに追加されたリソースは、デフォルトの自動化目標 (オフライン) が設定されている管理対象リソースになります。

この値は、リソース・グループ・レベルで **chrg** コマンドを使用して変更できます。リソース・グループ apache2rg をオンラインにするには、以下のように入力します。

```
chrg -o online apache2rg
```

lssam コマンドは、自動化ポリシーの内容の概要および各ノードからの実際のリソース状況情報を表示します。

```
lssam -V
Online IBM.ResourceGroup:apache2rg Nominal=Online
     |- Online IBM.Application:apache2                   -.
          |- Online IBM.Application:apache2:node01        |
          |- Offline IBM.Application:apache2:node02       |
          '- Offline IBM.Application:apache2:node03     DO
     '- Online IBM.ServiceIP:apache2IP IP=10.152.172.11 <'  -.
          |- Online IBM.ServiceIP:apache2IP:node01            |
          |- Offline IBM.ServiceIP:apache2IP:node02           |
          '- Offline IBM.ServiceIP:apache2IP:node03         DO
Online IBM.Equivalency:netequ                                <'
     |- Online IBM.NetworkInterface:eth0:node01
     |- Online IBM.NetworkInterface:eth0:node02
     '- Online IBM.NetworkInterface:eth0:node03
```

リソースの自動化

自動化リソースおよび自動化ポリシーを処理するときには、System Automation for Multiplatforms の動作についてのいくつかの考慮事項に注意する必要があります。

- リソースを System Automation for Multiplatforms によって自動化および制御するには、そのリソースをリソース・グループに追加する必要があります。System Automation for Multiplatforms は、リソースの操作状態 (OpState) をリソース・グループの本来あるべき状態 (NominalState) と同じ状態に変更しようとします。
- System Automation for Multiplatforms によって制御されるリソース (アプリケーション) は、コマンド **chrg**、**rgreq**、または **rgmbrreq** を使用することによってのみ、安全に開始および停止できます。これらのコマンドは、本来あるべきリソース状態を変更できます。
- System Automation for Multiplatforms は、リソースのモニター中に、オペレーターによって手動で開始または停止された自動化リソースがあったかどうかを検出できます。その後、リソースが再度開始または停止されて、本来あるべき状態に戻されます。この動作は、サスペンドまたはロックされていないすべてのリソースに適用されます。自動化対象から除外済みのノード上のリソースについては、本来あるべき状態はオフラインに設定されているため、これらのリソースは停止されます。ノード上で自動化リソースの実行が許可されない場合、そのノードは自動化対象から除外されることがあります。
- 手操作による介入が必要な場合は、以下のコマンドを使用して、自動化をサスペンドできます。

  ```
  # samctrl -M T
  ```

  System Automation for Multiplatforms 自動化モードに制御を戻すには、以下のコマンドを入力します。

  ```
  # samctrl -M F
  ```

- コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。
- System Automation for Multiplatforms ドメインで稼働されるノードには、特別な保守上の考慮事項が適用されます。これは、ノードを停止するために使用されるコマンド **halt** および **reboot** の場合に特に重要です。

  ノードを停止する前に、そのノードを自動化除外リストに追加する必要があります。ノードが自動化除外リストにある場合、System Automation for Multiplatforms は、そのノードで実行されるすべてのリソースを停止します。リソースは、リソースの実行が許可される別のノードに移動されます。すべてのリソースが移動された後で、そのノードを安全に停止またはリブートできます。

  ノードを除外リストに追加するには、以下のように入力します。

  ```
  # samctrl -u a <node-name>
  ```

  これでノードを安全に停止または再始動できます。ノードが再始動され、ドメインで再びオンラインになった後で、以下のコマンドを使用して、除外リストからそのノードを削除できます。

  ```
  # samctrl -u d <node-name>
  ```

  ノードが自動化対象に再度組み込まれても、リソースが暗黙的に移動されてこのノードに戻されることはありません。この動作の目的は、リソースが現行ノード上で中断されないようにして、使用可能な状態を維持することです。別の動作が必要な場合は、62 ページの『ホーム・ノードへのフェイルバック機能』を参照してください。

  以下のコマンドを使用して、除外リストを表示できます。

  ```
  # lssamctrl
  ```

- ノードが 2 つ (あるいは偶数ノード) の System Automation for Multiplatforms ドメインの場合は、タイ・ブレーカーが必要です。タイ・ブレーカーがない場合、2 番目のノードは、障害が発生するかリブートされたノードで実行されていたリソースを引き継ぎません。タイ・ブレーカー・タイプとセットアップについては、4 ページの『クォーラム・サポート』を参照してください。

第 3 章 管理

System Automation for Multiplatforms の管理には、リソース、同値、および関係の定義および処理や、自動化ポリシーの処理などの作業が含まれます。

ノードの管理

ノードをクラスターに追加したら、ノードにリソースを追加するための管理ステップを実行する必要があります。

ノードを追加したら、以下のステップを実行します。

- タイプが IBM.Application の固定リソース (例えば アプリケーション・サーバー) を作成します。
- 集合リソースについて以下を実行します。
  - タイプが IBM.AgFileSystem のリソース: ファイル・システムがマウントされたノードでの再取得を開始します。`refrsrc -s IBM.Disk <newly_added_nodename>` を使用します。
  - 動的選択を使用した同値: 追加のステップは不要です。
  - 静的リストを使用した同値: タイプが IBM.Application のリソースを参照してください。
  - ResourceGroup、Relationship: 追加のステップは不要です。

詳しくは、「147 ページの『クラスターの作成と保守』」を参照してください。

リソースの操作

リソースは、リソース・グループ内で管理することで、操作できます。

System Automation for Multiplatforms では、リソースを直接開始または停止できないことに注意してください。

リソースの開始および停止は、リソース・グループの NominalState 属性の設定に基づきます。例えば、リソース・グループの NominalState 属性を「オンライン」に設定すると、ユーザーがリソース・グループ内の リソースの開始を要求していることが示されます。リソース・グループの NominalState 属性を 「オフライン」に設定すると、ユーザーがリソース・グループ内のリソースの停止を 要求していることが示されます。134 ページの『リソース・グループの開始および停止』および 23 ページの『NominalState 属性』の説明を参照してください。

RSCT ストレージ・リソース・マネージャーのリソースの自動化

クラス IBM.AgFileSystem の RSCT ストレージ・リソース・マネージャーのファイル・システム・リソースを自動化することができます。

System Automation for Multiplatforms は、RSCT が提供するストレージ・リソース・マネージャー機能を使用します。この機能によって、IBM.AgFileSystem クラスのファイル・システム・リソースを自動化できます。これらのリソースを自動化するために System Automation for Multiplatforms によって提供されるサポートの範囲は、ファイル・システムが保管されているストレージ・デバイスのタイプ、ストレージ・デバイスが接続されるノードのオペレーティング・システム、およびストレージ・エンティティーが取得されるか、ユーザー定義されるかどうかなどの、多くの要因によって異なります。

- 自動的に取得される **IBM.AgFileSystem** リソースを自動化する場合、これ以上の追加の手動構成は必要ありません。自動的に取得されないファイル・システムを自動化するには、ユーザー定義の **IBM.AgFileSystem** リソースを使用します。以後のセクションでは、これらのリソースを作成および自動化するために実行する必要のある手順について説明します。

ユーザー定義の IBM.AgFileSystem リソースの作成

自動的に取得されないファイル・システム (NFS デバイスなど) を自動化するには、**mkrsrc** を使用して、対応する **IBM.AgFileSystem** リソースを作成します。

例:

```
mkrsrc IBM.AgFileSystem Name=nfs1 DeviceName=nfsserver1:/exportnfs1 Vfs=nfs
                        MountPoint=/mnt NodeNameList={'node1','node2'}
```

LVM 環境におけるユーザー定義の IBM.AgFileSystem リソースの自動化

論理ボリュームに常駐し、自動的に取得されないファイル・システムを自動化するには、System Automation for Multiplatforms の方法を使用して、ファイル・システム、論理ボリューム、およびボリューム・グループ間の依存関係を定義する必要があります。次の処置を行う必要があります。

1. ボリューム・グループを表す IBM.Application クラスのリソースを定義します。ボリューム・グループの活動化、非活動化、および状況報告を行う適切なコマンドを含む、IBM.Application リソースの StartCommand、StopCommand、および MonitorCommand 属性用のスクリプトを作成します。
2. ファイル・システムを表す IBM.AgFileSystem クラスのリソースを定義します。
3. ファイル・システム・リソースがソースおよびアプリケーション・リソースであり、ボリューム・グループがターゲットであることを示す、DependsOn 関係を定義します。

取得された IBM.AgFileSystem リソースの AIX での使用

AIX の場合は、あらゆる種類のストレージの管理において、StorageRM の使用をお勧めします。StorageRM は、ドメインの始動時に、接続されているすべてのディスクから情報を取得します。また、運用中にも定期的に情報を取得します。

この結果として、StorageRM では、IBM.Disk、IBM.VolumeGroup、および IBM.AgFileSystem クラスのリソースを作成します。StorageRM は、これらのリソース間の関係および依存関係を認識します。これらのリソースを制御するには、適切な IBM.AgFileSystem リソースを選択し、それをリソース・グループに追加します。リソースが System Automation for Multiplatforms によって開始されると、StorageRM は、ディスクの予約、ボリューム・グループのアクティブ化、およびファイル・システムのマウントを行います。

ストレージ・リソース・マネージャーについて詳しくは、「Reliable Scalable Cluster Technology」を参照してください。

IBM.AgFileSystem によって使用される属性

IBM.AgFileSystem リソース・クラスでは、属性のセットが使用されますが、その一部は永続または動的です。

RMC コマンド **mkrsrc** を使用して、クラス IBM.AgFileSystem のリソースを作成できます。

表 *23. IBM.AgFileSystem* クラス属性

永続属性	**属性**	**動的属性**
名前	NodeNameList	OpState
Vfs	ResourceType	
DeviceName	ProtectionMode	
MountPoint	PreOnlineMethod	
	Force	

Name
Name 永続属性は、このマウント・ポイントのユーザー定義名です (例: mail-server-data)。集合リソースおよび構成要素リソースの両方が、この属性について同じ値を持ちます。この属性の値は、 新規 IBM.AgFileSystem リソースの作成時に指定する必要があり、固有でなければなりません。この属性は、文字ストリング型でなければなりません。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

取得したリソース名に対して、名前が固有 ID に設定されます。 AIX では、これはファイル・システム名で、「/」が「_」に変換されます。例えば、/data/internal は data_internal として取得されます。Linux では、ファイル・システムのラベルが使用されます。ラベルが設定されていない場合は、固有のストリングが作成されます。 Linux では、ファイル・システムのラベルを指定することを検討してください。

Vfs
Vfs 永続属性を使用して、リソースが表すファイル・システムのタイプを指定します。

DeviceName
DeviceName 永続属性を使用して、既存ファイル・システムの名前を指定します (例: /dev/fslv03)。

MountPoint
MountPoint 永続属性を使用して、既存のマウント・ポイントを指定します。

NodeNameList
NodeNameList 永続属性は、IBM.AgFileSystem リソースがどのノードで使用可能であるかを示す ストリングの配列です。リソースが集合リソースの場合、ストレージ・リソース・リソース・マネージャーにより、このリストのノード名ごとに 1 つの構成要素 (固定) リソースが必ず存在します。構成要素リソースは、このリストのエントリーと一致 するよう、必要に応じて暗黙的に作成または削除されます。構成要素リソースは固定リソースであり、1 つのノードでのみ使用可能であるため、この配列内に 1 つのエントリーのみを含みます。集合リソースの この属性は、管理者によって構成要素リソースが明示的に除去された場合、集合リソースと構成要素リソースの関係が常に整合性を保つよう 暗黙的に変更されます。集合リソースの場合、このリストが空の場合があります。これはこのリソースがどこにおいても使用可能でないこと、および 構成要素が単独で追加可能であることを意味します。リソースが固定の場合 (すなわち ResourceType=0)、リストには最大で 1 つの名前を含めることができます。固定リソースはノードに結合されており、したがってノード名またはノード ID なしでは作成できないため、固定リソースに対して名前が指定されていない場合は、RMC がデフォルトの名前を提供します。集合リソースのこの属性の値は、**chrsrc** コマンドで変更できます。 構成要素リソースのこの属性を変更しようとすると、エラーが生成されます。

ResourceType
ResourceType 永続属性は、リソースが固定、浮動、または並行であるか どうかを識別するために使用します。整数値 0 はリソースが固定であることを示し、値 1 は浮動リソースであることを示し、値 2 は並行リソースであることを示します。IBM.AgFileSystem リソース・クラスでは、値 0 および 1 のみをサポートしています。新規 IBM.AgFileSystem リソースの作成時にこの属性が指定されていない場合、この属性のデフォルト値は 0 (固定) です。

ProtectionMode
ProtectionMode 永続属性は、リソースがクリティカルであるかどうかを指定します。リソースがクリティカルである場合は、IBM.ConfigRM によって、要求に応じてこのリソースを開始できるかどうかが決定されます。(この動作についての詳細は、4 ページの『クォーラム・サポート』を参照してください。)属性は、整数値 0 (非クリティカル) または 1 (クリティカル) である必要があります。デフォルト値は非クリティカルです。

PreOnlineMethod
この設定では、デフォルトでストレージ・リソース・マネージャーが **fsck** コマンドを使用して ファイル・システムの検査と修復を実行するかどうかと、実行する場合の fsck コマンドの正確な形式を決定します。 詳しくは、「*RSCT* 管理ガイド」を参照してください。

Force
同じ DeviceName 属性値を指定する IBM.AgFileSystem リソースが存在する場合は Force=1 を指定します。これは、ファイル・システムがストレージ・リソース・マネージャーによって取得された場合

が該当します。(NFS の場合など) ストレージ・リソース・マネージャーにデバイスが既知でない場合、Force 属性は不要です。

OpState
この動的状態属性の値には、リソース・マネージャーによって 判別されるリソースの操作状態が含まれます。この状態の 標準的な値は「オンライン」(値は 1) および「オフライン」(値は 2) で、マウント・ポイントが マウントされているのかいないのかを意味します。

グローバル・リソース・マネージャーの使用

グローバル・リソース RM (IBM.GblResRM) は、以下の 2 つのリソース・クラスを提供します。

IBM.Application
このクラスを使用することで、RMC サブシステムを介して、追加のリソースのタイプ (例えばビジネス・アプリケーションなど) をモニターし、制御できます。また、これらのリソースを System Automation for Multiplatforms などの管理アプリケーションで自動化または リカバリーできます。

IBM.ServiceIP
このクラスは、RMC サブシステムの制御下において、ピア・ドメイン内の ネットワーク・アダプターおよびノード間で開始、停止、および移動できる IP アドレスを 管理するために使用します。これらの IP アドレスは、 通常、ドメイン内で実行されているいくつかのサービスに接続しているクライアントに対して提供されます。ドメイン内に障害が発生した場合も、System Automation for Multiplatforms を使用して、サービスおよび その関連付けられた IP アドレスをアクティブなままにすることができます。

リソース・マネージャー (およびそのクラスへのアクセス) は、ピア・ドメイン・モードでのみ 操作可能です。

以下のサブセクションでは、このリソース・マネージャーによってサポートされる リソース・クラスの外部特性について説明します。各サブセクションにおいて、持続的で動的な属性、アクションなどを含め、1 つの特定のリソース・クラスについて説明します。

IBM.Application リソース・クラス

IBM.Application リソース・クラスを使用することにより、RMC サブシステムを介して、新しいタイプの浮動リソース、並行リソース、および固定リソースを作成、モニター、および制御できます。また、これらのリソースを System Automation for Multiplatforms で自動化またはリカバリーできます。新規リソースを 作成するには、以下の 3 つのスクリプトまたはコマンドが必要です。

1. リソースをオンラインにするための開始スクリプト (またはコマンド)。
2. リソースをオフラインにするための停止スクリプト (またはコマンド)。
3. ポーリングを通してリソースをモニターするためのスクリプト (またはコマンド)。

これらのスクリプトに加えて、IBM.Application リソース・クラスには以下の基本パラメーターがあります。

1. リソースの名前。
2. リソースを稼働できるクラスター・ノード。
3. アプリケーションを開始/停止/モニターするために使用するユーザー名。
4. アプリケーションを同期的または非同期的に開始/停止するために使用するメソッド。
5. さまざまなタイムアウト値。
6. リソースの特徴 (クリティカルまたは非クリティカル)。
7. モニター頻度の決定。
8. リソースの識別 (固定、浮動、または並行)。

IBM.Application リソース・クラスを通して装備される各汎用リソースは、単一のノードに結合されないことを意味するグローバル・リソースであると見なされます。ただし、リソースを、クラスターのノードのサブセットにのみ 存在するものとして定義できます。それぞれの汎用リソースごとに、IBM.Application リソース・クラスのインスタンスを 1 つ作成する必要があります。このインスタンス

は ノード間で移動したり、複数ノードで開始したりできる、浮動リソースまたは並行リソースを表すため、集合リソースと 呼ばれます。さらに、汎用リソースが存在するそれぞれのノードごとに、IBM.Application リソース・クラスの 1 つのインスタンスがあります。これらは、集合リソースの構成要素リソースと呼ばれます。構成要素リソースは、クラスターのただ 1 つのノードに存在するという意味で 固定リソースです。109 ページの図 81 は、集合リソースと構成要素リソースの 違いを説明したものです。

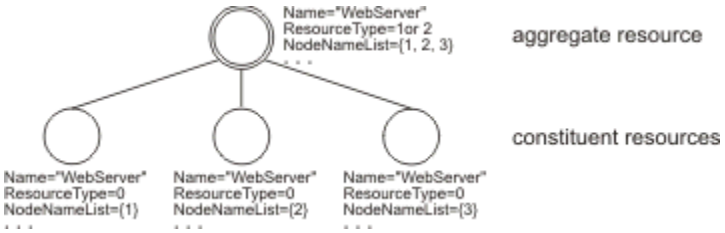


図 *81.* 集合リソースと 構成要素リソース

構成要素リソースは、集合リソースの定義が変更されるときに、自動的に 作成または削除されます。ほとんどの管理操作は集合リソースを 介して実行されますが、アプリケーションによっては、構成要素を直接モニターまたは操作する ことを選択できるものもあります。集合リソースへの変更は自動的にすべての構成要素に適用されますが、構成要素の属性の変更はその構成要素のみに作用し、その他のリソースに引き渡されることはありません。例えば、1 つのノード上の構成要素は、異なる開始コマンドまたはモニター間隔を使用することがあります。

集合リソースの NodeNameList からノードが除去されると、構成要素は自動的に削除されます。

IBM.Application によって使用される属性

以下の属性は、IBM.Application リソース・クラスのリソースによって使用されます。

RMC コマンド **mkrsrc** を使用してこのクラスのリソースを 作成する場合、リソースは以下の属性を保持している必要があります。

- Name
- StartCommand
- StopCommand
- MonitorCommand
- UserName

このクラスのリソースは、以下の動的属性を保持します。

- OpState

109 ページの表 24 では、IBM.Application リソース・クラスのリソースによって使用される任意指定の属性をリストします。

表 *24. IBM.Application* リソース・クラスによって使用される任意指定の属性

任意指定の属性	将来の利用のために予約されている任意指定の属性
NodeNameList	HealthCommand
ResourceType	HealthCommandPeriod
StartCommandTimeout	HealthCommandTimeout
StopCommandTimeout	MovePrepareCommand
MonitorCommandTimeout	MoveCompleteCommand
MonitorCommandPeriod	MoveCancelCommand
RunCommandsSync	ResetState

表 24. *IBM.Application* リソース・クラスによって使用される任意指定の属性 (続き)	
任意指定の属性	**将来の利用のために予約されている任意指定の属性**
ProtectionMode	ReRegistrationPeriod
MonitorUserName	
ProcessCommandString	
CleanupCommand	
CleanupCommandTimeout	
CleanupNodeList	

Name 属性

Name 永続属性は、ユーザーが定義した汎用アプリケーション・リソースの名前です。集合リソースおよび構成要素リソースの両方が、この属性について同じ値を持ちます。

この属性の値は、新規 IBM.Application リソースの作成時に指定する必要があり、固有でなければなりません。この属性は、文字ストリング型でなければなりません。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

NodeNameList 属性

NodeNameList 永続属性は、IBM.Application リソースがどのノードで使用可能であるかを示すストリングの配列です。リソースが集合リソースの場合、System Automation for Multiplatforms により、このリストのノード名ごとに 1 つの構成要素 (固定) リソースが必ず存在します。

構成要素リソースは、このリストのエントリーと一致 するよう、必要に応じて暗黙的に作成または削除されます。構成要素リソースは固定リソースであり、この配列内に 1 つのエントリーしか含まないため、1 つのノードでのみ使用可能です。集合リソースの この属性は、管理者によって構成要素リソースが明示的に除去された場合、集合リソースと構成要素リソースの関係が常に整合性を保つよう 暗黙的に変更されます。

集合リソースの場合、このリストが空の場合があります。これはこのリソースがどこにおいても使用可能でないこと、および構成要素が単独で追加可能であることを意味します。リソースが固定の場合 (すなわち ResourceType=0)、リストには最大で 1 つの名前を含めることができます。固定リソースに対して名前が 1 つも指定されていない場合は、固定リソースはノードに結合されており、ノード名またはノード ID なしでは作成できないため、RMC がデフォルトの名前を提供します。

集合リソースのこの属性の値は、**chrsrc** コマンドで変更できます。 構成要素リソースのこの属性を変更しようとすると、エラーが生成されます。

ResourceType 属性

ResourceType 永続属性は、リソースが固定、浮動、または並行であるか どうかを示すために使用されます。整数値 0 はリソースが固定であることを示し、値 1 は浮動リソースであることを示し、値 2 は並行リソースであることを示します。新規 IBM.Application リソースの作成時にこの属性が指定されていない場合、この属性のデフォルト値は 0 (固定) です。

StartCommand 属性

StartCommand 永続属性の値には、リソース・マネージャーが対応するリソース・インスタンスの開始要求を受け取ったときに実行される正確なコマンドおよび引数が含まれます。コマンドは、オンライン要求が集合リソースに対して発行された場合でも、構成要素リソースによってのみ実行されます。この場合は、 そのオンライン要求を実行するためのすべての構成要素リソースが、リソース・マネージャーによって選択されます。

コマンドは、UserName 属性で指定されたユーザー ID の下で実行されます。コマンドは、指定されたユーザーの権限および環境で 実行されます。

コマンドが完了するまでリソース・マネージャーが 待機するかどうかは、RunCommandsSync 属性によって制御されます (詳細はコマンドの説明を 参照)。コマンド名は文字ストリングでなければならず、絶対パスである 必要があります (すなわち、「/」で開始しなければなりません)。これは、リソースがアクセス可能な (すなわち構成要素がある) 各ノードに存在し、実行可能でなければなりません。この属性は、新規 IBM.Application リソースの定義時に指定する必要があります。

IBM.Application リソースの **StartCommand** は、ディレクトリー・パスに関して POSIX.2 標準に準拠しなければなりません。マルチバイト・エンコードを必要とする文字を含めることはできません。

コマンドからは以下の値が戻されます。

0
コマンドは正常に実行されました。

!= 0
コマンド処理中にエラーが発生しました。

117 ページの『自動化コマンドの戻りコードの処理』を参照してください。

StopCommand 属性
StopCommand 永続属性の値には、リソース・マネージャーがリソース・インスタンスの停止要求を受け取ったときに実行される正確なコマンドおよび引数が含まれます。集合の停止要求により、すべての構成要素が StopCommand を実行します。コマンドの実行に関するその他のすべての特徴は、StartCommand の場合と同様です。この属性は、新規 IBM.Application リソースの定義時に指定する必要があります。

IBM.Application リソースの StopCommand は、ディレクトリー・パスに関して POSIX.2 標準に準拠する必要があります。マルチバイト・エンコードを必要とする文字を 含めることはできません。

コマンドからは以下の値が戻されます。

0
コマンドは正常に実行されました。

!= 0
コマンド処理中にエラーが発生しました。

117 ページの『自動化コマンドの戻りコードの処理』を参照してください。

MonitorCommand 属性
MonitorCommand 永続属性の値には、リソースの操作状態 (OpState 属性) を判別または更新するために定期的に実行される正確なコマンドおよび引数が含まれます。コマンドからの 終了値が、リソースの新規 OpState として使用されます。

不明 = 0
オンライン = 1
オフライン = 2
オフラインに失敗 = 3
オンライン中 = 4
オンラインの保留中 = 5
オフラインの保留中 = 6
不適格 = 8

MonitorCommand の終了値が上にリストされた値と異なる場合、リソースの OpState は「不明」に設定されます。

少なくとも「オンライン」状況および「オフライン」状況は、**MonitorCommand** スクリプトによって設定する必要があります。

リソース・マネージャーは、OpState 動的属性のサブスクライバーがある場合に、このコマンドを **MonitorCommandPeriod** の値 (秒) の間隔ごとに実行します。コマンドの実行に関するその他のすべての特徴は、**StartCommand** について説明した内容と同じです。このコマンドは通常、システム・リソースの消費を回避するために有効です。

コマンドは、適当な時間内、通常は 2 から 3 秒、遅くとも 360 秒以内に戻る必要があります。

MonitorCommand の名前は絶対パスである必要があります (すなわち、「/」で開始しなければなりません)。これは、リソースがアクセス可能な (すなわち構成要素がある) 各ノードに存在し、実行可能でなければなりません。

この属性は、新規 IBM.Application リソースの定義時に指定する必要があります。
MonitorCommand 属性の戻り値について詳しくは、116 ページの『IBM.Application リソースの定義』および 117 ページの『自動化コマンドの戻りコードの処理』を参照してください。

IBM.Application リソースの **MonitorCommand** は、ディレクトリー・パスに関して POSIX.2 標準に準拠する必要があります。マルチバイト・エンコードを必要とする文字を 含めることはできません。

MonitorCommandPeriod 属性
MonitorCommandPeriod 永続属性の値には、**MonitorCommand** の呼び出し間隔中に待機する時間 (秒数) を指定します。この期間は、前の呼び出しが完了した後に開始されます。この属性は整数型で、0 より大きくなければなりません。デフォルト値は 5 秒です。

前の呼び出しの完了後の期間の開始以降、MonitorCommandTimeout 属性の値は、MonitorCommandPeriod 属性の値を上回っている可能性があります。

MonitorCommandTimeout 属性
MonitorCommandTimeout 永続属性は、モニター・コマンドが停止されずに、実行が許可される時間を指定します。コマンドがタイムアウトになると、リソースの操作状態 (OpState 属性) は 不明 = 0 に設定されます。

この属性は整数型でなければならず、0 以上でなければなりません。この値 は、360 秒未満でなければなりません。 この属性のデフォルト値は 5 秒です。

StartCommandTimeout 属性
StartCommandTimeout 永続属性は、開始コマンドが停止されずに、実行が許可される時間を指定します。この属性は、System Automation for Multiplatforms が、リソースがオンラインになることを待機する 経過時間も指定します。すなわち、System Automation for Multiplatforms は、制御パラメーターによって与えられるデフォルトのタイムアウト値の代わりにこの値を使用します。

この属性は整数型で、0 以上でなければなりません。この属性の値 0 は、タイムアウトがないことを意味します。 この属性は、RunCommandsSync 属性が 0 に設定されている場合は使用されません。この属性のデフォルト値は 5 秒です。

StopCommandTimeout 属性
StopCommandTimeout 永続属性は、停止コマンドが停止されずに、実行が許可される時間を指定します。この属性は整数型でなければならず、0 以上でなければなりません。値 0 は、タイムアウトがないことを意味します。 この属性のデフォルト値は 5 秒です。

RunCommandsSync 属性
RunCommandsSync 永続属性により、リソースの自動化動作における複数の側面を制御できます。この属性は、8 ビットのビット・フィールドとして実装されています。すべてのビットは、設定されているか (=1) 設定されていないか (=0) のいずれかです。

表 *25. RunCommandsSync* 属性設定の概要

名前	ビット	値	デフォルト
同期コマンド	1	1	設定あり
ログイン環境	2	2	設定なし
オンラインに反転	3	4	設定なし
強制終了の前に終了	4	8	設定なし
バイナリー・モード	5	16	設定なし
<<将来の利用のために予約>>	6	32	
ProbePID	7	64	設定なし

表 25. *RunCommandsSync* 属性設定の概要 (続き)			
名前	**ビット**	**値**	**デフォルト**
<<将来の利用のために予約>>	8	128	

同期コマンド:

RunCommandsSync 属性の先頭のビットを使用すると、開始コマンドおよび停止コマンドを同期実行できるかどうかを制御できます。このビットの値が設定されている場合、このコマンドが完了するかタイムアウトになるまで、開始コマンドおよび停止コマンドに対する応答は実行されません。すべての stderr 出力および stdout 出力の出力結果が応答で戻されます。

このビットが設定されていない場合、IBM.GblResRM は開始コマンドおよび停止コマンドを実行してすぐに復帰します。fork または exec が正常に完了するとすぐに、リソース・マネージャーはそれらのモニターを停止し、それらのコマンドはリソース・マネージャーから切り離されて実行されます。この属性のデフォルト値は 1 です。この属性が 0 に設定されていると、コマンドにタイムアウトが適用されません。

StartCommand がアプリケーション実行可能であり、コマンドは一定の時間内には戻らないが、アプリケーションが実行されている限り実行される場合、同期モードは使用できません。同期モードの場合、リソース・マネージャーは、**StartCommand** がタイムアウトになった後、コマンドを強制終了します。**StartCommand** の完了に必要な通常の時間が分かる場合は、同期コマンドを設定してください。**StartCommandTimeout** 属性をこの時間に合わせます。システム負荷が高い場合、いずれのエラー状態もなく正常に開始した場合でも、この時間が長くなることがあります。

アプリケーションが実行されるまではコマンドが完了しない場合 (例えばアプリケーションの実行可能プログラム) は、同期コマンドを設定解除します。アプリケーション実行可能プログラムを直接実行する (非同期モード) 一方で、同期コマンド・モードを使用する場合は、I/O ファイル記述子をリダイレクトし、コマンドをシェル・バックグラウンドに送信してください。リソース・マネージャーがプロセスまたは I/O 記述子、またはその両方にまだ接続している場合は、**StartCommand** のタイムアウトにより、プロセスが終了されます。

ログイン環境

このビットは、開始/停止コマンドおよびモニター・コマンドが実行される環境を制御します。コマンドに対する基本的な環境、あるいはユーザー・プロファイルおよびシェル構成ファイルを含む十分なログイン環境を稼働できるリソース・マネージャーのいずれかを選択できます。これは、システム・コマンド su (ユーザーの切り替え) に類似しています。 現行の環境を保持することができます。または、su - を実行して、選択したユーザー用の完全なプロファイル (ログイン・シェル) を実行することもできます。

オペレーティング・システム (AIX または Linux) に基づいて、 以下の環境変数 (変数の値はサンプル・ユーザーを示す) がユーザーの限定された環境に含まれます。

```
SHELL=/bin/bash
USER=myuser
PATH=:/usr/ucb:/bin:/usr/bin
LOGIN=myuser
PWD=/home/myuser
LANG=de_DE.UTF-8
HOME=/home/myuser
SHLVL=2
LOGNAME=myuser
```

これが、指定された開始/停止コマンドおよびモニター・コマンドに対して十分ではない場合は、開始/停止コマンドまたはモニター・コマンド自体に必要な変数を設定するか、ユーザー・プロファイル (またはシェル構成) にそれらを入れ、ログイン環境を設定します。

オンラインに反転

このビットは、プロセスの開始に影響します。**StartCommand** が正常に終了した後で、リソースが開始されます。オンラインに反転ビットが 設定されていない場合は、リソースのオンライン状態を確認するモニターが 実行されます。StartCommand 自体でオンライン状態を確認できる場合は、

オンラインに反転ビットを設定できます。オンラインに反転ビットが設定されている場合は、モニターの実行がスキップされ、**StartCommand** が終了されるとすぐにリソースはオンラインであるとマーク付けされます。

強制終了の前に終了
任意のコマンドがタイムアウトになった場合 (属性 **MonitorCommandTimeout**、**StartCommandTimeout**、**StopCommandTimeout**、および **CleanupCommandTimeout** で指定)、デフォルトの動作ではそのコマンドを実行しているプロセスに kill シグナルが送信されます。この kill シグナルによりコマンドの 終了が強制されますが、コマンドがクリーンアップ・ルーチンを実行することは 許可しません。強制終了の前に終了ビットが設定されている場合は、まず終了 (termination) シグナルがコマンドに送信されます。コマンドにシグナル・ハンドラーが実装 されていれば、このハンドラーが制御を取得してクリーンアップおよび終了の処理を 整然と実行します。この終了処理は 2 秒以内に実行される必要があります。実行されない場合は、これに続いて、終了を強制する kill シグナルが GblResRM によって送信されます。

バイナリー・モード
コマンドを開始するデフォルトの方式では、Korn シェル・プロセスが spawn され、 そこでコマンドが実行されます。コマンドがスクリプトでなく バイナリー実行可能プログラムである場合は、中間のシェルは不要です。 代わりに、GblResRM の子プロセスとしてコマンドを直接実行できます。

ProbePID
この設定は、リソースに ProcessCommandString も指定されている場合のみアクティブです。通常は、ProcessCommandString がプロセス・テーブルで検出されると、リソースはオンラインであると評価されます。ProbePID を 設定すると、この動作が変更されます。ProcessCommandString を検出した後で、リソースの **MonitorCommand** が実行されます。次に、このコマンドにより、アプリケーションの状況が判別されます。ProcessCommandString を使用してオフライン・リソースに対する時間を節約する一方で、リソースがオンラインである必要があるかどうかを完全に検査する場合に、ProbePID を設定します。

RunCommandsSync ビット・マスクの処理

RunCommandsSync の値を使用することにより、特定の属性がアクティブであるかどうかを評価できます。設定されているオプションを 判別するには、以下の手順を実行します。

1. lsrsrc を使用して、RunCommandsSync の現行値を表示します。
2. RunCommandsSync の 10 進値を 2 進に変換します。
3. 設定しようとしているビットがアクティブ化されているのか (=1) いないのか (=0) を確認します。

RunCommandsSync オプションの状況を変更するには、以下の手順を実行します。

- 非アクティブ化された RunCommandsSync オプションをアクティブ化するには、RunCommandsSync の 10 進値にそのオプションの値を加算します。
- アクティブな RunCommandsSync オプションを非アクティブ化するには、RunCommandsSync の 10 進値からそのオプションの値を減算します。

UserName 属性
UserName 永続属性は、**MonitorCommand**、**StartCommand**、**StopCommand**、および **CleanupCommand** が実行される際のユーザー名を定義します。コマンドは、指定されたユーザーの権限および環境で実行されます。リソースの 構成が変更された場合は常に、ユーザー名が存在するかどうかを確認するために 各ノードが検査されます。この属性は、新規 IBM.Application リソースの定義時に指定する必要があります。この属性は、文字ストリング型でなければなりません。指定されたユーザー用のアクセス可能なホーム・ディレクトリーを定義する必要があります。

ProtectionMode 属性
ProtectionMode 永続属性は、リソースがクリティカルであるかどうかを指定します。リソースがクリティカルである場合、IBM.ConfigRM が、そのリソースを要求に応じて開始できるかどうかを判断します。この 動作についての詳細は、4 ページの『クォーラム・サポート』を参照してください。

この属性は、整数値 0 (非クリティカル) または 1 (クリティカル) をとり、デフォルトは非クリティカルです。リソースが クリティカルに設定されている場合、このリソースに対するサブスクライバーが存在しなくても、モニターが即時に開始されます。

OpState 属性
この動的状態属性の値には、**MonitorCommand** の定期的な実行による終了コードによって決定されるリソースの操作状態が含まれます。RMC アーキテクチャーの状態遷移図によって定義される この属性に指定可能な値を以下に示します。

不明 = 0
オンライン = 1
オフライン = 2
オフラインに失敗 = 3
オンライン中 = 4
オンラインの保留中 = 5
オフラインの保留中 = 6
不適格 = 8

この属性は、集合リソースおよびすべての構成要素リソースの 両方から使用可能です。集合リソースの値は、各構成要素からの 状態のロールアップです。

MonitorUserName
オプションです。モニター・コマンドは、MonitorUserName で指定されたユーザー ID のセキュリティー・コンテキストで実行されます。この属性が空の場合、UserName で指定されているユーザー ID が代わりに使用されます。最大 1024 文字のストリング値です。指定されたユーザー用のアクセス可能なホーム・ディレクトリーを定義する必要があります。

ProcessCommandString
オプションです。ProcessCommandString 属性が設定されている場合は、リソースのモニター動作が変更されます。MonitorCommandPeriod の有効期限が切れると、MonitorCommand を実行する代わりに、プロセス・テーブル・スキャンが開始されます。ProcessCommandString で定義されているストリングが検出される場合、リソースはオンラインであると評価されます。その他の場合は、オフラインであると 評価されます。この動作は、RunCommandsSync 属性の ProbePID ビットを設定することにより、さらにカスタマイズできます。

最大 1024 文字のストリング値です。

CleanupCommand 属性
オプションです。CleanupCommand 属性は、開始コマンドや停止コマンドに似たクリーンアップ・コマンドを指定します。このコマンドが定義されている場合は、このリソースに対するクリーンアップ処理がアクティブです。リソースに対するクリーンアップ処理は、 リソースの計画外の停止が検出された場合に、必ずクリーンアップ・コマンドを 実行します。クリーンアップ・コマンドは、次に、CleanupNodeList (CleanupNodeList が空の場合は、NodeNameList) で定義されているノードで実行されます。クリーンアップ処理は、障害が発生したときに永続情報を残すリソースに対して使用できます。通常は、このリソースを再開または フェイルオーバーする前に、この永続情報を削除する必要があります。 適切なクリーンアップ・コマンドによって、この作業を実行できます。 コマンド名は文字ストリングでなければならず、絶対パスである 必要があります (すなわち、/ で開始しなければなりません)。このコマンドは、クリーンアップが実行される可能性のある各ノードに存在し、実行可能でなければなりません。

CleanupNodeList 属性
オプションです。CleanupNodeList 属性は、CleanupCommand の実行に関して、ノードとノードの順序を定義します。クリーンアップは、CleanupNodeList の最初の適格ノードに対して実行されます。CleanupCommand がタイムアウトになるか、ゼロ以外の戻りコードで復帰する場合は、そのノードには、クリーンアップが失敗したとマークが付けられ、次のノードで CleanupCommand が実行されます。すべてのノードが失敗する場合は、ノードのクリーンアップ状態がリセットされて、最初のノードが CleanupCommand の実行を再度開始します。値を指定せずに CleanupCommand を設定した場合は、CleanupNodeList は NodeNameList のミラーになります。

CleanupCommandTimeout 属性
オプションです。CleanupCommandTimeout 属性は、クリーンアップ・コマンドが停止されずに、実行が許可される時間を指定します。この属性は整数型でなければならず、0 以上でなければなりません。値 0 は、タイムアウトがないことを意味します。 この属性のデフォルト値は 5 秒です。

HealthCommand
将来の利用のために予約されています。

HealthCommandPeriod
将来の利用のために予約されています。

HealthCommandTimeout
将来の利用のために予約されています。

InstanceName
将来の利用のために予約されています。

InstanceLocation
将来の利用のために予約されています。

IBM.Application によって使用されるアクション

このセクションでは、IBM.Application リソース・クラスのリソースに対して実行可能なアクションについて説明します。

refreshOpState アクション
通常の操作において、MonitorCommandPeriod 属性は、IBM.Application リソースの OpState がそのモニター・スクリプトによって評価される間隔を決定します。リソースが自ら障害を検出できる場合は、リソースを モニターするモニター・スクリプトの即時実行を起動できます。この結果、アプリケーション・リソースの OpState が即時にリフレッシュされます。

例えば、node02 上で稼働中のリソース「WebServer」の OpState をリフレッシュするには、任意のノードで以下のコマンドを発行します。

```
runact -s "Name='WebServer' && NodeNameList={'node02'}" IBM.Application
refreshOpState
```

SendEIFevent
将来の利用のために予約されています。

IBM.Application リソースの定義

IBM.Application リソースを定義する場合は、以下の点に留意してください。

1. 目標駆動型自動化に対応するため、System Automation for Multiplatforms リソースは常にモニターされます。リソースを開始する場合、または停止する場合も、モニター・ コマンドによりリソースの実際の状況を判別する必要があります。常に存在しないファイルシステムに、モニター・コマンドを入れないでください (例えば、ポリシーの一部であり、NFS リソースが開始する場合のみマウントされる NFS マウント)。 IBM.Application OpState に Unknown が表示される場合は、システム・ログを調べ、リソース・マネージャーが随時モニター・コマンドにアクセスできることを確認してください。
2. リソース・マネージャーは、許可されるタイムアウト値を超えて実行されている すべてのコマンドを強制終了します。システム負荷が高いときに OpState は Unknown であると報告する IBM.Application リソースが存在する場合は、コマンドの完了にかかる時間のほうが、そのリソースで許可されているタイムアウト値よりおそらく長くなっています。CPU 時間が長い場合は、これが原因で問題が発生するこ とはありません。モニター・コマンドは正常に完了し、有効な OpState を再度報告できます。システム・ログを調べ、タイムアウトが原因でコマンドが強制終了されたか どうかを確認してください。コマンドが通常操作中に強制終了された場合は、リソース定義のタイ ムアウト値を確認し、調整します。
3. モニターは、プロセスとそのプロセスが担当するアプリケーションを明確に示す必要があります。プロセスは、プロセス・テーブルで一意的に定義する必要があります。
4. プリンター・スプーラーやメール移送エージェントなどのように、基本システム・サービスを自動化する場合は、これらのサービスが、自動化でのみ開始または停止されるようにし、システム実行レベルまたは init プロセスでは開始または停止されないようにしてください。
5. プロセスの有無のみでなく、アプリケーションによるサービスの配信が可能であるかも判別できる、より高度なモニターを使用する場合、その事実を自動化エンジンに報告することはできず、このアプリケーションの停止またはシャットダウンを 起動することはできません。プロセスまたはアプリケーショ

ンが使用可能であっても、停止またはスタック していることがモニターにより検出された場合、モニター自体がこのプロセス/アプリケーショ ンを強制終了します。自動化は、アプリケーションが既に (次回のモニターで) 使用可能 でないことを認識し、代わりのアプリケーションを再始動するか、または別のノードに 移動しようとします。

6. クラス IBM.Application のオンライン・リソースを (例えば **rmrsrc** を使用して) 除去しようとすると、コマンドは拒否されます。除去できるのは、オフライン・リソース のみです。Force=1 を設定することにより強制的に除去できます。例えば、WebServer というリソースを除去するには、以下のように入力します。

   ```
   rmrsrc -s "Name=='WebServer' && ResourceType==1" IBM.Application Force=1
   ```

7. オープン・ファイルの数に関するソフト制限が小さすぎる場合は、StartCommand、StopCommand、CleanupCommand、または MonitorCommand として実行されるスクリプト内で、必要に応じて調整できます。

   制限を大きくするには、リソースのコマンド・スクリプトのヘッダーの下に次のステートメントを追加します。

   ```
   # ulimit -S -n <new-limit-for-number-of-open-files>
   ```

   これが影響を与えるのは、例えば、WebSphere Application Server のような大量のオープン・ファイルを必要とするアプリケーションに対してのみです。

自動化コマンドの戻りコードの処理

以下の例は、リソース・マネージャーが、StartCommand、StopCommand 、および MonitorCommand の戻りコードを処理する方法を示します。

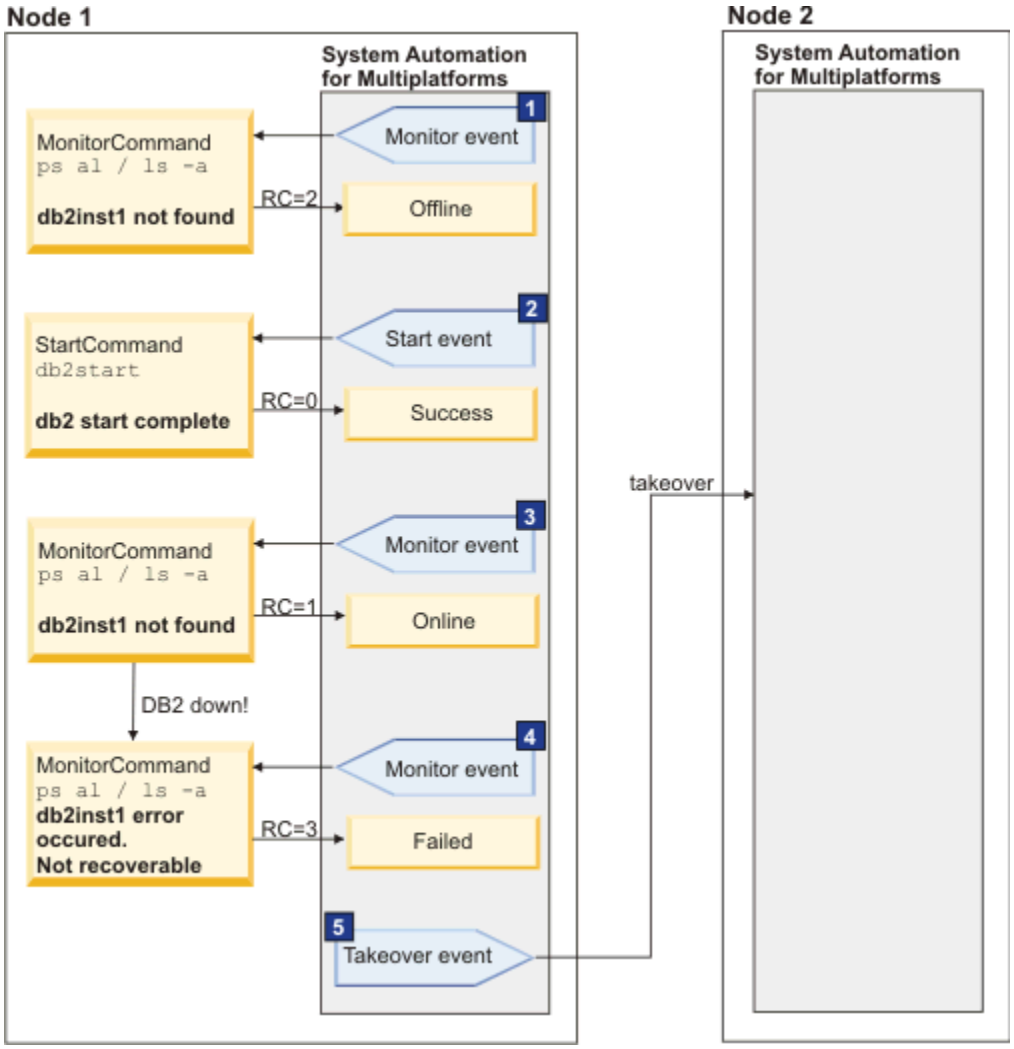


図 *82.* リソースのモニターと自動化の制御フローの例

1. System Automation for Multiplatforms がモニター・イベントを送信します。RC=2 が返されます。リソース DB2® はオフラインです。
2. System Automation for Multiplatforms が開始イベントを送信します。RC=0 が返されます。DB2 が正常に開始されます。
3. System Automation for Multiplatforms がモニター・イベントを送信します。RC=1 が返されます。これで、リソース DB2 はオンラインです。
4. System Automation for Multiplatforms がモニター・イベントを送信します。RC=3 が返されます。リソース DB2 が停止しました。
5. System Automation for Multiplatforms がノード 1 からノード 2 に引き継ぎイベントを送信します。ノード 2 の DB2 が 引き継ぎます。

リソース・マネージャーは、StartCommand、StopCommand、および MonitorCommand の戻りコードを以下のように処理します。

StartCommand

1. StartCommand がリソースを開始できた場合、リソースが正しく開始され、数秒以内にオンラインになることを示す戻り値 0 が戻されます。
2. StartCommand がリソースを開始できなかった場合、0 以外の値が戻されます。これは、自動化がリソースを再始動しなかったことと、リソースの操作状態が「オフラインに失敗」に設定されたことを知らせます。これは、ユーザーが手動で介在して、リソースを修正しなければならないことを示します。リ

ソースが開始されたら、**resetrsrc** コマンドを使用して、「オフラインに失敗」の操作状態をリセットします。StartCommand が失敗すると常に、自動化により StopCommand が実行され、開始に失敗したアプリケーションの残存物が除去されることに注意してください。

3. StartCommand が正常に完了し、値 0 が戻されたが、一定の時間 (自動化制御構成の設定に依存する) 後にリソースの MonitorCommand がオンライン状態を報告しない場合、自動化によりリソースの再始動が試行されます。開始試行の合計数は RetryCount 属性に定義され (*IBM Tivoli System Automation for Multiplatforms* リファレンス の **lssamctrl** コマンドの説明を参照)、デフォルト値は 3 です。最後の試行後もリソースの操作状態が「オンライン」に変わらない場合は、StopCommand が出され、リソースは「オフラインに失敗」に設定されます。
4. リソースの StartCommand が StartCommandTimeout 属性に指定した時間内に完了しない場合、リソース・マネージャーは StartCommand を強制終了し、項目 118 ページの『2』 で説明した失敗した開始コマンドのように開始を処理します。
5. リソースの定義時に StartCommand は有効であったが、後に削除されたか存在していない場合 (例えば、NFS マウントの欠落のため)、開始手順は、118 ページの『2』 項目での説明のように欠落 StartCommand と同様に処理されます。

StopCommand

1. StopCommand がリソースを停止できた場合、リソースが正しく停止され、数秒以内にオフラインになることを示す戻り値 0 が戻されます。
2. IBM Tivoli System Automation for Multiplatforms には、失敗した StopCommand を処理するメカニズムはありません。StopCommand は 0 以外の値を戻すことにより、アプリケーションの停止に失敗したことを示しますが、これによって自動化アクションは引き起こされません。

   リソースが、 一定時間後に OpState「オフライン」に達しない場合、IBM Tivoli System Automation for Multiplatforms はリソースに対して リセット操作を行い、その結果、StopCommand の 2 番目の呼び出しが行われます。この StopCommand の 2 番目の呼び出しは、2 番目の呼び出しの場合は 1 に設定される SA_RESET 環境変数を調べることで、StopCommand スクリプト内で判別できます。 StopCommand は、以下の例のように、これを有効と見なすことができます。

```
#/bin/sh
# A sample stop/reset automation script for the lpd application
if [ $SA_RESET == 1 ]; then
  killall -9 lpd
  exit $?
else
  /etc/init.d/lpd stop
  exit $?
fi
```

   StopCommand の 2 番目を実行する必要がない場合、例えば次のようにして、スクリプトをただちに終了できます。

```
#/bin/sh
# A sample stop/reset automation script for the lpd application
if [ $SA_RESET == 1 ]; then
  exit 0
else
  /etc/init.d/lpd stop
  exit $?
fi
```

   SA_RESET 環境変数を評価することによって、拡張停止動作を簡単に実行できます。 2 番目の StopCommand でもリソースの操作状態が「オフライン」にならない場合、リソースの OpState は「オンライン中」に設定されます。 この時点で、リソースを停止するためのオペレーター介入が必要になります。 リソースが最終的に停止し、OpState が「オフライン」に変わった後、IBM Tivoli System Automation for Multiplatforms は再度このリソースを制御します。
3. 以下のいずれかが当てはまる場合、変数 SA_RESET が 1 に設定されます。
   - StopCommand が 2 度目に実行される。
   - `resetrsrc` を使用したリソースのリセットのために、StopCommand が呼び出される。

- リソースを開始しようとして失敗した後に、リソースがリセットされる。これは、StartCommand が 0 (正常) を戻したが、MonitorCommand がリソースをオンラインとしてモニターしない場合に起こります。この場合、System Automation for Multiplatforms は、次の開始を試行する前に、リセット (SA_RESET=1 を設定した StopCommand) を開始します。

4. リソースの StopCommand が StopCommandTimeout 内に完了しない場合、リソース・マネージャーはコマンドを強制終了し、項目 119 ページの『2』 で説明した失敗した StopCommand のように停止を処理します。
5. ゼロ以外の戻りコード StopCommand またはリセット操作では、リソースは、自動化状態の「問題」に設定されます。この状態は、このリソースに対して **resetrsrc** が正常に実行されることによってのみリカバリーできます。リソースに対して **resetrsrc** を実行すると、リソースの StopCommand がトリガーされるため、StopCommand が正常に実行されることも必要になります (つまり、このコマンドがゼロの戻りコードを送信する必要があります)。一般に、リソースを停止させる本当の問題が存在する場合にのみ、リソースの StopCommand がゼロ以外の戻りコードで戻るように構成することをお勧めします。その他のすべての状態では、リソースが「問題」状態で終了しないように、StopCommand がゼロの戻りコードで戻るように構成する必要があります。
6. リソースの定義時に停止コマンドは有効であったが、後に削除されたか存在していない場合 (例えば、NFS マウントの欠落のため)、停止手順は、119 ページの『2』 項目での説明のように欠落 StopCommand と同様に処理されます。

MonitorCommand

1. MonitorCommand がリソースの操作状態を判別できた場合、 有効な操作状態のいずれかを戻します (109 ページの『IBM.Application によって使用される属性』 を参照)。MonitorCommand の終了値が有効な値と異なる場合は、リソースの OpState が 0 に設定されます。この場合、0 は「オンライン」の操作状態を示す戻り値ではなく、自動化の最も深刻な状態である「不明」の操作状態を示す戻り値です。不明の操作状態にあるリソースは、以後自動化されず、このリソースへの依存関係を持つ他のリソースにも影響を与える場合があります。
2. リソースのモニター・コマンドが MonitorCommandTimeout 内に完了しない場合、リソース・マネージャーは MonitorCommand を強制終了し、RMC 操作状態を「不明」に設定します。これは、リソースに重大な問題があることを表します。MonitorCommand が不明以外の操作状態を戻すまで、このリソースに対する自動化は行われません。
3. リソースの定義時にモニター・コマンドは有効であったが、後に削除されたか存在していない場合 (例えば、NFS マウントの欠落のため)、操作状態は「不明」に設定されます。これは、リソースに重大な問題があることを示します。
4. いずれのケースでも、システム負荷の軽減後または NFS が再び存在するようになった後、MonitorCommand は有効な操作状態を報告し続けることができます。それにより、自動化では、リソースの管理を続行できます。
5. リソースの StartCommand の実行の間、リソースの OpState は、MonitorCommand によって報告された実際の OpState に関係なく常に「オンラインの保留中」に設定され、リソースが実際には開始のプロセスにあることを反映します。 これには 2 つの例外があります (MonitorCommand が「オンライン」または「オフラインに失敗」に戻る場合)。 なぜなら、これらの OpState 値は、リソースが既にオンラインであるか破壊されていて、開始できないことを示しているためです。
6. リソースの StopCommand の実行の間、リソースの OpState は、MonitorCommand によって報告された実際の OpState に関係なく常に「オフラインの保留中」に設定され、リソースが実際には停止のプロセスにあることを反映します。 これには 3 つの例外があります (MonitorCommand が「オフライン」、「オフラインに失敗」、または「オンライン中」に戻る場合)。 なぜなら、これらの OpState 値は、リソースが既にオフラインであるかオンライン中であって、 停止できないことを示しているためです。

リソース・マネージャー操作

UNIX/Linux カーネルの最初のプロセスは、`init` プロセスであり、これはシステム・リソース・コントローラー (src) を作成および始動 (spawn) します。以下の図で示すように、システム・リソース・コントローラーは、System Automation for Multiplatforms リソース・マネージャーを担当します。

```
Init-+-atd
     |
     |-srcmstr
     |          |
     |          |-IBM.GblResRMd---IBM.GblResRMd---13*[Ibm.GblResRMd]
```

図 *83.* リソース・マネージャー操作

グローバル・リソース・マネージャーは追加のプロセスを作成して、IBM.Application リソースの開始、停止、およびモニター・コマンドを実行します。グローバル・リソース・マネージャーが作成したすべてのコマンドは、指定されたユーザーのシェルで実行されます。以下に、この機能の疑似コードをいくつか示します。

```
{
  fork;
  if child
    switch to specified user ID;
    run the users default shell and execute the command e.g.
      bash -c /usr/bin/mycommand;
  endif
}
```

前述の疑似コードの mycommand のようなコマンド自体を、追加プロセスを作成したり、ジョブ制御を使用したりするシェル・スクリプトを含む実行可能プログラムにすることができます。

さまざまなシナリオを以下に説明します。

1. IBM.Application は、Asynchronous コマンド・モードで定義されます。

   ```
   StartCommand="/usr/bin/mycommand"
   RunCommandsSync=0
   ```

   この場合、グローバル・リソース・マネージャーはコマンドのプロセスを作成 (fork) してから、新規プロセスから完全に切り離して、新規プロセスに対するファイル記述子をすべてクローズします。

   グローバル・リソース・マネージャーは、新規プロセスにこれ以上関与しません。理論上、コマンドは永久に実行できます。

   新規プロセスが親を所有しなくなったため、これは孤立することになり、init プロセスによって選択されます。最終的にプロセスが終了したときに、init はプロセスの戻りコードを収集します。

2. IBM.Application は、Synchronous コマンド・モードで定義されます。

   ```
   StartCommand="/usr/bin/mycommand"
   RunCommandsSync=1
   ```

   ここでは、Global Resource Manager は、新規に作成された (fork された) プロセスからグローバル・リソース・マネージャー自身を切り離しません。ファイル記述子はオープンされ、リソース・マネージャーはコマンドの完了を待機します。

   - コマンドが誤った戻りコードを戻す場合は、stderr からのメッセージが収集され、グローバル・リソース・マネージャーのトレースおよび **start** または stop コマンドのエラー・ブロックに書き込まれます。
   - StartCommandTimeout 属性に指定されている時間内にコマンドが戻らなかった場合、グローバル・リソース・マネージャーは、fork されたプロセス (ユーザー・デフォルト設定のシェル) に対して SIGKILL を送信します。SIGKILL は、ユーザー・シェルのすべての子プロセスに伝搬されるため、すべての子プロセスは終了します。

3. IBM.Application は、synchronous コマンド・モードで定義されますが、ユーザー・シェルのジョブ制御を使用します。

   ```
   StartCommand="/usr/bin/mycommand &"
   RunCommandsSync=1
   ```

このセットアップでは、ユーザー・デフォルトのシェルは、シェルに対するオープン・ファイル記述子を持つすべての子プロセスが終了するまで完了しません。ここでは、StartCommandTimeout 属性に指定されている時間は、mycommand にも適用されます。これが、ユーザー・シェルのバックグラウンドで実行されている場合も同様です。このため、StartCommandTimeout 属性に指定されている時間よりも長時間 mycommand が実行された場合、グローバル・リソース・マネージャーはユーザー・シェルに SIGKILL を送信し、そのユーザー・シェルはすべてのバックグラウンド・プロセスに SIGKILL を伝搬します。

このセットアップを作動する場合、シェルの子プロセスのファイル記述子がすべて シェルから切り離されていることを確認する必要があります。以下のように行います。

```
StartCommand="/usr/bin/mycommand > /dev/null 2>&1 &"
RunCommandsSync=1
```

これでユーザー・シェルは、コマンドに対プロセスを作成 (fork した)し、新規プロセスのファイル記述子から切り離した直後に終了できます。コマンド mycommand 自体は最初のシナリオで説明したように動作して、オーファンとなり、init プロセスが親になります。

IBM.Application リソースの実装

以下の例は、SUSE ベースの Linux システム上の lpd プリンター・スプーラーを System Automation for Multiplatforms で管理するための準備方法を示します。

1. システムのデフォルトの実行レベルから lpd を除去します。このリソースを複数ノードで浮動リソ ースとして実行するには、各ノードの runlevel を確認する必要があります。
2. IBM.Application の開始コマンドと停止コマンドでは、lp デーモンに付属のデフォルト init スクリプトが使用されます。

```
StartCommand: /etc/init.d/lp start
StopCommand: /etc/init.d/lp stop
```

3. モニター・コマンドの場合は、プロセス・テーブルの lpd プロセスを検査する単純なシェル・スクリプトを使用します。

```
File: /root/lpmon
#!/bin/bash

OPSTATE_ONLINE=1
OPSTATE_OFFLINE=2

ps -ax | grep -v "grep" | grep "/usr/sbin/lpd" > /dev/null
if [ $? == 0 ]
then
  exit $OPSTATE_ONLINE
else
  exit $OPSTATE_OFFLINE
fi
```

あるいは、System Automation for Multiplatforms に付属の **pidmon** コマンドを使用 することもできます。このコマンドは、基本的に、指定されたコマンド・ストリングのプロセス・テーブルを検索します。コマンド・ストリングが見つかると、RMC OpState が戻されます。**pidmon** コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」で、このコマンドの詳細な説明を参照してください。

```
MonitorCommand: /root/lpmon
or
MonitorCommand: /usr/sbin/rsct/bin/pidmon '/usr/sbin/lpd'
```

4. 浮動リソースの場合は、すべてのノードが同一パスで開始/停止コマンドおよび モニター・コマンドにアクセスできるようにします。lpd は小さく単純なアプリケーションであるため、デフォルトの Start- または Stop- および MonitorCommandTimout の値 (デフォルトは 5 秒) を使用できます。init スクリプトを使用して lpd を開始するには、IBM.Application のユーザー名として root を指定します。

これで、**mkrsrc** コマンドを使用して IBM.Application リソースを定義できます。

```
# mkrsrc IBM.Application  ¥
        Name                 = "line_printer_daemon"  ¥
        ResourceType         = 1  ¥
        StartCommand         = "/etc/init.d/lpd start"  ¥
        StopCommand          = "/etc/init.d/lpd stop"  ¥
        MonitorCommand       = "/usr/sbin/rsct/bin/pidmon '/usr/sbin/lpd' "  ¥
        MonitorCommandPeriod  = 15  ¥
        MonitorCommandTimeout = 5  ¥
        StartCommandTimeout   = 5  ¥
        StopCommandTimeout    = 5  ¥
        UserName             = "root"  ¥
        RunCommandsSync      = 1  ¥
        ProtectionMode       = 0  ¥
        NodeNameList         = "{'node01','node02'}"
```

このコマンドを実行すると、3 つのリソース (潜在的にノード node01 および node02 でオンラインにすることができる line_printer_daemon という名前の 1 つの集合リソースと、1 つは node01、もう 1 つは node02 にある、これも line_printer_daemon という名前の 2 つの構成要素リソース) が作成されます。集合リソースに対して開始要求が発行されると、グローバル・リソース RM は構成要素の 1 つを選択し、それを StartCommand 属性で指定されているスクリプト (またはコマンド) を使用して開始します。

IBM.Application リソースのサポート・リソースを構成する

IBM.Application を IBM.Equivalency および DependsOn 関係と組み合わせて使用する場合、自動化は同値からリソースを選択し、このリソースをサポート・リソースとして IBM.Application に提供します。

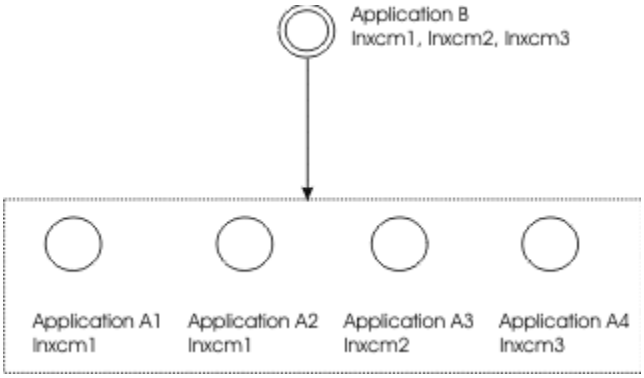


図 *84. IBM.Application* リソースのサポート・リソースを構成する

このサンプル構成では、自動化は、同値からアプリケーション Ax を選出し、アプリケーション B からアプリケーション Ax への関係が施行されることを確認します。詳細については、*IBM Tivoli System Automation for Multiplatforms* 管理者とユーザーのガイドを参照してください。同値からのより多くのリソースが DependsOn 関係のロケーション制約を満たすことができるため (この例では A1 および A2 が lnxcm1 上で実行できる)、自動化は、どのリソースがリソース B の開始コマンドに選択されているのかを示す情報を提供します。

必要に応じて、リソース B の開始スクリプトはこの情報を使用して、アプリケーションに特別なパラメーターを受け渡したり、または同値から選択されたリソースとともに実行される必要がある専用のコードを活動化したりすることができます。

この情報を開始コマンドに受け渡すには、リソース・マネージャーがリソースの 開始コマンド環境で以下の環境変数を設定します。

表 *26.* リソースの開始コマンドの環境変数

変数	**値**
SA_SUPPORTING_RESOURCE_RH	サポートするリソースのリソース・ハンドル (例えば 0x601d 0xffff 0xcac15160 0xbad91087 0x0f933128 0x58888f98)

表 26. リソースの開始コマンドの環境変数 (続き)	
変数	**値**
SA_SUPPORTING_RESOURCE_NAME	サポートするリソースの名前 (例えば A1)

以下のいくつかのサンプル・コードは、サポートするリソースの環境変数をリソース B の開始スクリプトが受け入れる方法を示します。

```
# start command for resource B with logic for
# supporting resource
…
if [ $SA_SUPPORTING_RESOURCE_NAME = "A1" ]
then
  # start resource B and connect it to supporting
  # resource A1
  …
fi

if [ $SA_SUPPORTING_RESOURCE_NAME = "A2" ]
  # start resource B and connect it to supporting
  # resource A2
  …
then
fi
```

IBM.ServiceIP リソース・クラス

IBM.ServiceIP リソース・クラスは、ピア・ドメイン内のアダプターとノード間で開始、停止、および移動できる IP アドレスを管理するために使用します。このクラスの各リソースは 1 つの IP アドレスを示します。 これらの IP アドレスは、 通常、ドメイン内で実行されているいくつかのサービスに接続しているクライアントに対して提供されます。ドメイン内に障害が発生した場合も、リカバリー管理アプリケーションを 使用して、サービスおよびその関連付けられた IP アドレスをアクティブなままに保持 できます。

サービス IP アドレスは、管理者が、そのリソースを潜在的にオンラインにできる ようにする各ノードごとに 1 つの構成要素リソースを持つ 集合リソースです。これは、一度にアクティブな構成要素を 1 つのみ保持できるため、浮動集合リソースです。

IBM.ServiceIP リソース・クラスは、以下の基本パラメーターを使用します。

1. リソースの名前。
2. リソースを稼働できるノード。
3. 移動可能な IP アドレス。
4. IP アドレスのネットマスク。

ネットワーク・インターフェースのブロードキャスト・アドレスおよびフラグは、開始時に ServiceIP リソースがマップされる親インターフェースから取得されます。

IBM.ServiceIP は、IBM.ServiceIP が存在するネットワークの 静的ルーティング・エントリーを生成します。このネットワークおよび経路は、ServiceIP がマップされるデバイスのネットワーク構成を破壊しないことを 確認してください。

また、自動化は動的ルーティングを管理しません。親 ネットワーク・インターフェースのサブネットにない ServiceIP を指定する場合、この物理デバイスが 2 つの異なるネットワークをホストします。このネットワークを サポートするために、必ず自動化クラスターの外部のルーティングを正しくセットアップしてください。

IBM.ServiceIP の特性

IBM.ServiceIP クラスには、以下の 2 種類の特徴があります。

1. **適切なネットワーク・インターフェースを自動的に選択する IBM.ServiceIP**

   IBM.ServiceIP が開始要求を受け取ると、リソース・マネージャーは、適切なネットワーク・インターフェースを選択しようとします。目的のインターフェースが見つかると、サービス IP はそのインター

フェースにマップされます。このインターフェースを、サポート・リソースまたはサポート・ネットワーク・インターフェースと呼びます。

適切なネットワーク・インターフェースを決定するために、リソース・マネージャーは、IBM.ServiceIP リソースの IPAddress 属性をすべての既存のネットワーク・インターフェースの IP アドレスと比較します。一致する適切なネットワーク (サブネット) が見つかった 場合、リソース・マネージャーは、そのネットワーク・インターフェースに別名アドレスを割り当てます。一致するサブネットが見つからなかった場合は、オンライン要求は失敗します。

自動インターフェース選択アルゴリズムの場合、自動化によってネットワーク・インターフェースの 実際の状態を評価する機会はありません。ネットワーク・インターフェースが構成され、実行されている限り、ServiceIP をネットワーク・インターフェースに割り当てることができます。例えばネットワーク・ケーブルのプラグが抜けたなどの場合、UNIX/Linux ではネットワーク・インターフェースの状況が変更されないことに注意してください。デバイス・ドライバーが 欠落したケーブルを検出できても、インターフェースは構成および実行されたままです。

RSCT ハートビート・メカニズムを活用して欠落したケーブルなどのネットワーク・インターフェース障害を検出する場合は、次のセクションで説明する IBM.ServiceIP のサポート・リソースのセットアップを使用します。

例

IBM.ServiceIP が以下のように定義されています。

```
IPAddress=192.168.1.5
NetMask="255.255.255.0"
NodeNameList="{'node01','node02'}"
```

クラスター内で以下のネットワーク・インターフェースが使用可能です。

- ```
  1. IPAddress=192.168.1.1
     Netmask=255.255.255.0
     NodeNameList="{'node01'}"
  ```
- ```
  2. IPAddress=9.152.172.91
     Netmask=255.255.255.0
     NodeNameList="{'node02'}"
  ```
- ```
  3. IPAddress=192.168.2.1
     Netmask=255.255.255.0
     NodeNameList="{'node03'}"
  ```

番号 1 のインターフェースのみがサービス IP を保持できます。その他のすべてのインターフェースは、サービス IP のアドレス (サブネット) と一致しません。IBM.ServiceIP を node01 以外の他のノードで開始する呼び出しは、適切なサポート・ネットワーク・インターフェースがないため失敗します。

2. **サービス IP アドレスが存在する別名に対するサポート・リソースを受け取る IBM.ServiceIP**

この場合、IBM.ServiceIP リソースの IPAddress 属性および NetMask 属性に制限はありません。サービス IP は、ネットワーク・インターフェースの同値に対して DependsOn 関係 (38 ページの『DependsOn 関係』を参照) を保持していなければなりません (サポートするリソースについては、52 ページの『同値』を参照)。System Automation for Multiplatforms は、特定のノードのサービス IP をオンラインにすることを決定すると、ネットワーク・インターフェースの同値から適切なネットワーク・インターフェースを選出し、これをサポート・リソースとして IBM.ServiceIP に提供します。これは、別名をホスティングする インターフェースです。

この構成では、サービス IP アドレスが別名を 割り当てられるネットワーク・インターフェースの操作状態を、自動化によりモニターできます。RSCT ハートビートによって検出されるインターフェース障害の場合は、自動化により依存関係のチェーンが強制的に終了され、ServiceIP が停止されます。同値に別のオンライン・ネットワーク・インターフェースがある場合、自動化により、ServiceIP を割り当てるためにこのデバイスが選択されます。

例:
  ```

IBM.ServiceIP が以下のように定義されています。

```
IPAddress=9.152.192.1
NetMask="255.255.255.0"
NodeNameList="{'node01','node02','node03'}"
```

クラスター内で以下のネットワーク・インターフェースが使用可能です。

- 1. IPAddress=192.168.1.1
  Netmask=255.255.255.0
  NodeNameList="{'node01'}"
- 2. IPAddress=192.168.1.2
  Netmask=255.255.255.0
  NodeNameList="{'node02'}"
- 3. IPAddress=192.168.1.3
  Netmask=255.255.255.0
  NodeNameList="{'node03'}"

3 つのネットワーク・インターフェースはすべて 1 つの同値に含まれます。

以下の図ではセットアップを説明します。

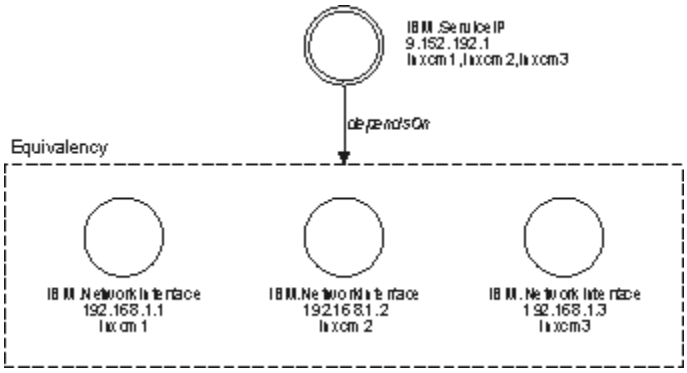


図 *85. 1* つの同値からの *3* つのネットワーク・インターフェース

System Automation for Multiplatforms は、サービス IP の開始を決定すると、同値からネットワーク・インターフェースを 選出します。この例では、サービス IP は、node01、node02、および node03 の 間を浮動できます。これは、3 つのノードのすべてが同値内にサポート・リソースとして動作できる ネットワーク・インターフェースを保持しているためです。

IBM.ServiceIP によって使用される属性

このトピックでは、IBM.ServiceIP リソース・クラスのリソースによって使用される属性について説明します。

RMC コマンド **mkrsrc** を使用してこのクラスのリソースを 作成する場合、リソースは以下の属性を保持している必要があります。

- 名前
- IPAddress

このクラスのリソースは、以下の属性を保持できます。

- NodeNameList
- ResourceType
- NetMask (または NetPrefix)
- ProtectionMode

このクラスのリソースは、以下の動的属性を保持します。

- OpState

Name 属性
Name 永続属性は、このサービス IP アドレスのユーザー定義名 です (例えば「mail-server-ip」)。集合リソースおよび構成要素リソースの両方が、この属性について 同じ値を持ちます。この属性の値は、新規 IBM.ServiceIP リソースの作成時に指定する必要があり、固有でなければなりません。この属性は、文字ストリング型でなければなりません。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

NodeNameList 属性
NodeNameList 永続属性は、IBM.ServiceIP リソースがどのノードで使用可能であるかを示すストリングの配列です。

リソースが集合リソースの場合、グローバル・リソース・マネージャーにより、このリストのノード名ごとに 1 つの構成要素 (固定) リソースが必ず存在します。構成要素リソースは、このリストのエントリーと一致 するよう、必要に応じて暗黙的に作成または削除されます。構成要素リソースは固定リソースであり、この配列内に 1 つのエントリーしか含まれないため、1 つのノードでのみ使用可能です。

集合リソースの この属性は、管理者によって構成要素リソースが明示的に除去された場合、集合リソースと構成要素リソースの関係が常に整合性を保つよう 暗黙的に変更されます。集合リソースの場合、このリストが空の場合があります。これはこのリソースがどこにおいても使用可能でないこと、および 構成要素が単独で追加可能であることを意味します。

リソースが固定の場合 (すなわち ResourceType=0)、リストには最大で 1 つの名前を含めることができます。固定リソースに 対して名前が 1 つも指定されていない場合は、固定リソースはノードに結合されており、ノード名またはノード ID なしでは作成できないため、RMC がデフォルトの 名前を提供します。集合リソースのこの属性の値は、**chrsrc** コマンドで変更できます。構成要素リソースのこの属性を 変更しようとすると、エラーが生成されます。

ResourceType 属性
ResourceType 永続属性は、リソースが固定、浮動、または並行であるか どうかを識別するために使用します。整数値 0 はリソースが固定であることを示し、値 1 は浮動リソースであることを示し、値 2 は並行リソースであることを示します。IBM.ServiceIP リソース・クラスでは、値 0 および 1 のみをサポートしています。新規 IBM.ServiceIP リソースの作成時にこの属性が指定されていない場合、この属性のデフォルト値は 0 (固定) です。

IPAddress 属性
IPAddress 永続属性を使用して、リソースがオンラインになるネットワーク・ インターフェース上で別名が割り当てられる IP アドレスを指定します。新規 IBM.ServiceIP リソースの作成時にこの属性が必要です。IPv4 の場合、IP アドレスは、「ドット 10 進」表記で文字ストリングとして指定する必要があります (9.152.80.251 など)。 IPv6 の場合、IPv6 アドレスの標準形式を使用できます (2001:db8::1428:57ab など)。IPv6 では、リンク・ローカル・アドレスはこの属性に使用できないことに注意してください。

NetMask 属性
NetMask 永続属性を使用して、IP アドレス属性に定義した IP アドレスに割り当てられるネットマスクを指定します。この属性は、文字ストリングとして指定しなければなりません。例えば、255.255.255.0 のようにします。IPAddress 属性が IPv6 アドレスを含んでいるか 、NetPrefix が指定されている場合は、NetMask 属性の設定は許可されません。

NetPrefix 属性
この属性は、IPAddress 属性で指定された IPv6 アドレスのネット・プレフィックス値を指定します。これは IPv6 と IPv4 の両方 の IP アドレス範囲で有効です。これは、IPv6 においては NetMask 属性の必須の置換属性として機能し、IPv4 においてはオプションの置換属性として 機能します。この属性には、整数値のみを使用します (80 など)。 数値の前に、スラッシュまたはその他の文字を指定しないでください。

ProtectionMode 属性
ProtectionMode 永続属性は、リソースがクリティカルであるかどうかを指定します。リソースがクリティカルである場合、IBM.ConfigRM が、そのリソースを要求に応じて開始できるかどうかを判断し

ます。(この動作についての詳細は、4 ページの『クォーラム・サポート』を参照してください。)この 属性は、整数値 0 (非クリティカル) または 1 (クリティカル) をとり、デフォルトは クリティカルです。

OpState 属性
この動的状態属性の値には、リソース・マネージャーによって 判別されるリソースの操作状態が含まれます。この状態の 標準的な値は「オンライン」(値は 1) および「オフライン」(値は 2) で、IP アドレスが 操作可能である、または操作不能であることを意味します。

IBM.ServiceIP が開始した場合に発生する現象

リソース・マネージャーが、選択されたネットワーク・インターフェースの IP アドレスを割り当てることができる場合、以下の機能が実行されます。

1. OSA ネットワーク・ハードウェアを使用する Linux on System z 上では、指定された IP アドレスに対して IP アドレスの引き継ぎが開始されます。
2. ServiceIP が IP の別名として指定されたネットワーク・デバイス上に作成されます。
3. 誤ったハードウェア・アドレス (MAC アドレス) の ServiceIP がある可能性のある、他の IP ホストの ARP (Address Resolution Protocol) キャッシュ項目を無効にするために、非送信請求/無償 ARP パケットがネットワークにブロードキャストされます。(IPv6 アドレスには適用されません。)
4. ServiceIP が、指定されたネットワーク・インターフェースで開始されたことを記録するシステム・ログ・メッセージが作成されます。

以下にも注意してください。

- IP アドレスの別名が、ホストのシステム・ルーティング・テーブル内に新規項目を生成する場合があります (ServiceIP ネットワークがデバイスのネットワークと異なる場合)。
- IBM.ServiceIP は、ご使用のデフォルトのゲートウェイ設定、または他のネットワーク/ルーティング構成を変更しません。

例 *1: IP* アドレスを *IBM.ServiceIP* リソースとして定義する

IP が 9.152.172.11、ネットマスクが 255.255.255.0 で、ノード node05 および node06 で稼働可能な IP アドレスを定義するには、以下の RMC コマンドを発行します。

```
mkrsrc IBM.ServiceIP
  Name="WebServerIP"
  NodeNameList="{'node05','node06'}"
  IPAddress=9.152.172.11
  NetMask=255.255.255.0
```

例 *2: IP* アドレスを *IBM.ServiceIP* リソースとして定義し、ネットワーク・インターフェースの *IBM.Equivalency* を使用する

先の例で示したように、IP が 9.152.172.11、ネットマスクが 255.255.255.0 で、ノード node05 および node06 で稼働可能な IP アドレスを 定義します。

```
mkrsrc IBM.ServiceIP
  Name="WebServerIP"
  NodeNameList="{'node05','node06'}"
  IPAddress=9.152.172.11
  NetMask=255.255.255.0
```

ノード node05 および node06 は、それぞれ複数の ネットワーク・インターフェースを保持します。ノード node05 および node06 のデバイス eth1 を 含む同値を形成するには、以下のように入力します。

```
mkequ MyInterfaces IBM.NetworkInterface:eth1:node05,eth1:node06
```

これで、同値を使用して ServiceIP に接続できます。

```
mkrel -p dependson -S IBM.ServiceIP:WebServerIP -G IBM.Equivalency:MyInterfaces
        WebIp_depon_MyInterfaces
```

例 *3: IPv6* アドレスを *IBM.ServiceIP* として定義する

次の例を使用するには、まず IPv6 サポートを使用可能に設定する 必要があります。これについては System Automation for Multiplatforms インストールと構成のガイドのセクションで説明します。アドレス IP が 2001:db8::1428:57ab、ネット・プレフィックスが 64 で、ノード node05 および node06 で稼働可能な IPv6 アドレスを定義するには、以下の RMC コマンドを発行します。

```
mkrsrc IBM.ServiceIP
  Name="WebServerIP"
  NodeNameList="{'node05','node06'}"
  IPAddress=2001:db8::1428:57ab
  NetPrefix=64
```

前の例で説明されているように、IPv6 リソース用の IBM.Equivalency も 使用できます。これで、物理インターフェースごとに、IPv4 用に 1 つ と IPv6 用に 1 つの、2 つの IBM.NetworkInterface リソースを定義したため、 コマンドを微調整して同値を作成する必要があります。

```
mkequ -S "Name='eth1' & IPVersion=6" MyInterfaces IBM.NetworkInterface
```

追加された -S 引数により、名前が eth1 であり IPVersion 属性に 6 が設定されている IBM.NetworkInterface リソースのみが選択されます。 これで、リソースを同値に接続できるようになりました。

```
mkrel -p dependson -S IBM.ServiceIP:WebServerIP
  -G IBM.Equivalency:MyInterfaces WebIP_depon_MyInterfaces
```

テスト・リソース・マネージャーの使用

このセクションでは、テスト・リソース・マネージャーの特性について説明します。

このタスクについて

IBM テスト・リソース・マネージャー (IBM.TestRM) は、テスト・リソースを管理し、テスト・リソースの操作状態を操作する機能を提供します。このリソース・マネージャーは、ピア・ドメイン・モードでのみ操作可能で、リソース・クラス IBM.Test を提供します。IBM.TestRM は、実リソースを制御しません。

IBM.Test リソース・クラス

IBM.Test リソース・クラスを使用することにより、RMC サブシステムを介して新しいタイプの固定リソースおよび浮動リソースを作成、モニター、および制御できます。これらのリソースは実リソースではなく、それらを定義、モニター、および制御するための単なるコンテナーです。また、これらのリソースを System Automation for Multiplatforms で自動化またはリカバリーできます。IBM.Test クラスの目的は、実リソースのオーバーヘッドなしで自動化のシナリオをシミュレートするために、単純で扱いやすいリソースを提供することにあります。IBM.Test クラスによって制御される各リソースは、リソースが定義される各ノードにおいて、1 つの集合と 1 つの構成要素に分割されるグローバル・リソースであると見なされます。詳しくは、108 ページの『IBM.Application リソース・クラス』の 109 ページの図 81 を参照してください。

IBM.Test リソース・クラスは、実リソースの動作をシミュレートするための永続リソース属性のセットを提供します。

IBM.Test に使用される属性

このセクションでは、IBM.Test リソース・クラスによって使用される永続属性について説明します。

RMC コマンド **mkrsrc** を使用して IBM.Test のリソースを作成する場合、リソースは以下の永続属性を保持している必要があります。

Name
Name 永続属性は、テスト・リソースのユーザー定義名です。集合リソースおよび構成要素リソースの両方が、この属性について 同じ値を持ちます。この属性の値は、新規 IBM.Test リソースの作成時に指定する必要があり、固有でなければなりません。以下のいずれかの System Automation 機能でリソースを使用する場合は、必ず値の長さを 64 文字以下にしてください。

- **sampolicy**
- System Automation for Multiplatforms への接続。

IBM.Test のリソースは、以下の属性を保持します。

NodeNameList
NodeNameList 永続属性は、IBM.Test リソースがどのノードで使用可能であるかを示すストリングの配列です。

リソースが集合リソースの場合、TestRM により、このリスト内のノード名ごとに 1 つの構成要素 (固定) リソースがあることが確認されます。構成要素リソースは、このリストのエントリーと一致 するよう、必要に応じて暗黙的に作成または削除されます。構成要素リソースは固定リソースであり、この配列内に 1 つのエントリーしか含まれないため、1 つのノードでのみ使用可能です。集合リソースの この属性は、管理者によって構成要素リソースが明示的に除去された場合、集合リソースと構成要素リソースの関係が常に整合性を保つよう 暗黙的に変更されます。

集合リソースの場合、このリストが空の場合があります。これはこのリソースがどこにおいても使用可能でないこと、および 構成要素が単独で追加可能であることを意味します。

ResourceType
ResourceType 永続属性は、リソースが固定、浮動、または並行であるかどうかを識別するために使用します。整数値 0 はリソースが固定であることを示し、値 1 は浮動リソースであることを示し、値 2 は並行リソースであることを示します。新規 IBM.Test リソースの作成時にこの属性が指定されていない場合、この属性のデフォルト値は 0 (固定) です。

ForceOpState
この属性を使用して、RMC **chrsrc** コマンドを介してテスト・リソースの OpState の変更を開始します。これは、resetrsrc メソッドによってはリセットできないリソース内の、障害をシミュレートするために使用できます。最新の状態変更がこの永続リソース属性に 保管されます。リソースの作成時にこの属性を指定しても 影響はありません。通常、集合リソースはリソース全体の OpState を収集するため、ForceOpState 変更は構成要素リソースで実行します。この属性に使用できる値は以下のとおりです。

不明 = 0
オンライン = 1
オフライン = 2
オフラインに失敗 = 3
オンライン中 = 4
オンラインの保留中 = 5
オフラインの保留中 = 6
不適格 = 8

TimeToStart
TimeToStart 属性は、テスト・リソースが開始コマンドを受け取った後、リソースがその OpState を「オンラインの保留中」から「オンライン」に変更するまでの時間 (秒単位) を指定します。デフォルト値は 0 秒で、リソースは 即時にオンラインになります。

TimeToStop
TimeToStop 属性は、テスト・リソースが停止コマンドを受け取った後、リソースがその OpState を「オフラインの保留中」から「オフライン」に変更するまでの時間 (秒単位) を指定します。デフォルト値は 0 秒で、リソースは 即時にオフラインになります。

WriteToSyslog
クラス IBM.Test のリソースは、syslog 機能に、オンライン、オフライン、および ForceOpState イベントを記録できます。WriteToSyslog 属性を使用して、syslog デーモンへの書き込みをオンまたはオフします。この属性に使用できる値は以下のとおりです。

0
syslog に書き込みません (これがデフォルトです)

1
syslog に書き込みます

MoveFail
内部使用のために予約されています。

MoveTime
内部使用のために予約されています。

IBM.Test のリソースは、以下の動的属性を保持します。

OpState
この動的状態属性の値には、リソースの操作状態が 含まれます。IBM.Test リソースの OpState は、RMC の start または stop コマンド、あるいはオペレーターまたはテスト・スクリプト (自動化テスト・ケース) からの ForceOpState イベントに従います。RMC アーキテクチャーの状態遷移図によって定義される この属性に指定可能な値を以下に示します。

不明 = 0
オンライン = 1
オフライン = 2
オフラインに失敗 = 3
オンライン中 = 4
オンラインの保留中 = 5
オフラインの保留中 = 6
不適格 = 8

MoveState
内部使用のために予約されています。

OpQuorumState
内部使用のために予約されています。

例: テスト・リソースの作成およびその OpState の操作

2 つのノードで IBM.Test リソースを作成するには、以下の RMC コマンドを発行します。

```
mkrsrsc IBM.Test ¥
  Name="mytest" ¥
  NodeNameList="{'node01','node02'}" ¥
  ResourceType=1 ¥
  TimeToStart=5 ¥
  TimeToStop=2 ¥
  WriteToSyslog=1
```

以下のコマンドにより、node02 の構成要素の OpState が「オフラインに失敗」に変更されます。リソースが System Automation for Multiplatforms によって自動化されている場合、自動化マネージャーが 別のノードでリソースを開始します。

```
chrsrc -s "Name='myTest' && NodeNameList={'node02'}" IBM.Test ForceOpState=3
```

リソース・グループの管理

このセクションでは、リソース・グループを作成および管理する方法について説明します。 リソース・グループの属性を作成、削除、および変更することができます。 リソース・グループのメンバーを管理し、個々のリソース・グループを開始したり停止したりすることもできます。

リソース・グループの作成

リソース・グループを作成するには、**mkrg** コマンドを使用して、System Automation for Multiplatforms に対して以下のことを定義します。

- リソース・グループを稼働できる場所。
- 他のリソース・グループに対する関係におけるリソース・グループの 相対的な優先順位 (24 ページの『Priority 属性』で説明しているように、Priority 属性を 使用します)。
- リソース・グループのメンバー・リソース間のロケーション 関係 (44 ページの『ロケーション関係』で説明しています)。

新規作成されたリソース・グループは、デフォルトでオフラインの NominalState に設定されます。これにより、ユーザーまたは管理者がリソース・グループ およびそのリソースを完全に構成できます。

例えば、ロケーション関係が「None」で、許可されたノード名が「node03」の、apacherg2 という名前の新規リソース・グループを定義するには、以下のように入力します。

```
mkrg -l None -n node03 apacherg2
```

mkrg コマンドの詳細については、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

リソース・グループのノード・リストを設定するには、以下のいずれかを使用できます。

- **mkrg** コマンド。新規リソース・グループを作成します。
- **chrg** コマンド。既存のリソース・グループに対して使用します。

上記 2 つのコマンドで使用するノード・リストを指定するには、以下のいずれかを実行できます。

- **mkrg**/**chrg** コマンドの使用時にノード名を入力します。
- 同値 として、リソース・グループをアクティブにすることができる ノードのセットを設定します。これは、ノード・リストの設定前に 実行する必要があります。次に **mkrg**/**chrg** コマンドを使用します。必要な同値がリソース・グループに付加されます。同値についての詳細は、52 ページの『同値』を参照してください。

リソース・グループへのメンバー・リソースの追加

1 つ以上の新規メンバー・リソースをリソース・グループに追加するには、**addrgmbr** コマンドを 使用します。

注 :

1. メンバー・リソースを、同時に複数のリソース・グループに含めることは できません。
2. メンバー・リソースを、リソース・グループおよび 同値に 同時に含めることはできません。

例えば、リソース・グループ apacherg2 に、リソース・クラス IBM.Application に属するメンバー・リソース apache1 を追加するには、以下のように入力します。

```
addrgmbr -g apacherg2 IBM.Application:apache1
```

addrgmbr コマンドについて詳しくは、 「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

リソース・グループとそのメンバーのリスト

リソース・グループとそのメンバーをリストするには、以下のコマンドを使用します。

lssam
リソース・グループとそのメンバーをツリー形式でリストします **lssam** コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

lsrg
リソース・グループまたはリソース・グループのメンバーをリストします。 このセクションでは、コマンドの使用方法の概要、および若干の使用例を示します。

lsrg コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

リソース・グループまたはそのメンバーをリストするには、**lsrg** コマンドを使用できます。 リソース・グループ名を省略すると、すべてのリソース・グループがリストされます。-m オプションを使用してリソース・グループ名を指定した場合、メンバー・リソース名およびリソース・クラスがリストされます。

Attr パラメーターを指定すると、リソース・グループについて指定された 属性がリストされます。

以下の例では、3 つのリソース・グループが定義されます。

例 1: コマンド **lsrg** を入力すると、以下の種類の情報が表示されます。

```
Resource Group Names:
apacherg2
apacherg3
apacherg4
```

例 2: すべてのリソース・グループのすべてのメンバーをリストするには、以下のように 入力します。

```
lsrg -m
```

以下の情報が表示されます。

```
Displaying Member Resource information:
Class:Resource:Node[ManagedResource]  Mandatory MemberOf  OpState  WinSource  Location
IBM.Application:apache1               True      apacherg2 Offline  Nominal    node03
```

例 3: apacherg2 には、1 つのリソースが含まれています。apacherg2 の メンバーをリストするには、以下のコマンドを入力します。

```
lsrg -m -g eez-was-rg
```

リソース・グループ eez-was-rg の次の Member Resource 情報が 表示されます。

```
Member Resource 1:
Class:Resource:Node[ManagedResource]  = IBM.ServiceIP:eez-ip
Mandatory                             = True
MemberOf                              = eez-was-rg
SelectFromPolicy                      = ORDERED
Subscription              = {}
Instances                             = 0
Requests                              = {}
Ordering                              = 0
ActivePeerDomain                      = clmanager
ConfigValidity       =
OpState                               = Online
```

例 4: リソース・グループ apacherg2 の 属性をリストするには、以下のコマンドを入力します。

```
lsrg -g eez-was-rg
```

リソース・グループ eez-was-rg の次の Member Resource 情報が 表示されます。

```
Resource Group 1:
Name                                = eez-was-rg
MemberLocation                      = Collocated
```

```
Priority                              = 0
AllowedNode                           = ALL
NominalState                          = Offline
ExcludedList                          = {}
Subscription                          = {}
Owner                                 =
Description                           =
InfoLink                              =
Requests                              = {}
ActivePeerDomain                      = clmanager
OpState                               = Online
TopGroup                              = eez-srvdb2-rg
ConfigValidity                        =
TopGroupNominalState                  = Online
```

リソース・グループの開始および停止

リソース・グループを開始または停止するには、リソース・グループの NominalState 属性をそれぞれオンラインまたはオフラインに設定します。これを実行するには、 **chrg** コマンドを使用します。

例えば、apacherg2 というリソース・グループを開始するには、以下のように入力します。

```
chrg -o online apacherg2
```

chrg コマンドの詳細については、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

リソース・グループの属性の変更

1 つ以上のリソース・グループの永続属性値を変更するには、**chrg** コマンドを使用します。-c オプションを指定して このコマンドを使用することにより、リソース・グループの名前を変更することもできます。

例 1: グループ apacherg2 のロケーション関係を「Collocated」に変更するには、以下のように入力します。

```
chrg -l collocated apacherg2
```

例 2: グループ apacherg3 の名前を apacherg4 に変更するには、以下のように入力します。

```
chrg -c apacherg4 apacherg3
```

chrg コマンドの詳細については、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

リソース・グループ・メンバーの属性の変更

リソース・グループ・メンバーの属性を変更するには、**chrgmbr** コマンドを使用します。

このコマンドは、 -m オプションを使用することによって管理対象リソースの Mandatory 属性に対する変更を指定したり、-c オプションを使用することによってリソースが属するリソース・グループを変更したりできるようにします。

例えば、リソース・クラス IBM.Application のメンバー・リソース apache2 が属するリソース・グループを、現行のリソース・グループ apacherg2 からリソース・グループ apacherg3 に変更するには、以下のように入力します。

```
chrgmbr -c apacherg3 -g apacherg2 IBM.Application:apache2
```

chrgmbr コマンドについて詳しくは、 *System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

リソース・グループからのメンバー・リソースの除去

rmrgmbr コマンドを使用して、以下を除去します。

- 指定されたリソース・グループのすべてのメンバー・リソース。
- 指定されたリソース・グループの指定されたメンバー・リソースのみ。
- 選択文字列と一致するメンバー・リソース。

また、System Automation for Multiplatforms により関連する管理対象関係または同値も必ず 更新されます。

例えば、リソース・クラス IBM.Application に属するメンバー・リソース apache2 をリソース・グループ apacherg3 から除去するには、以下のように入力します。

```
rmrgmbr -g apacherg3 IBM.Application:apache2
```

詳しくは、**rmrgmbr** マニュアル・ページまたは *System Automation for Multiplatforms* リファレンス・ガイド のコマンドの説明のいずれかを参照してください。

リソース・グループの除去

1 つ以上のリソース・グループを除去するには、**rmrg** コマンドを使用します。除去するリソース・グループは、*Resource_group* パラメーターを使用するか、選択文字列と一致させることにより指定します。除去するリソース・グループに関連するメンバー・リソースも、System Automation for Multiplatforms によって除去されます。除去されるリソース・グループがオンラインの場合は、リソース・グループは除去されません。削除されたリソース・グループがソースである関係も削除されることに注意してください。

このことは、削除されるリソース・グループ内にネストされたリソース・グループも、すべて削除されることを意味します。含まれるリソース・グループが再帰的に削除されないようにするには、以下のようにします。

1. **rmrgmbr** コマンドを使用して、削除されるリソース・グループから、これらのリソース・グループをメンバーとして削除します。
2. これらが入っていたリソース・グループを除去します。

例えば、apacherg2 および apacherg3 というリソース・グループを除去するには、以下のように入力します。

```
rmrg apacherg2 apacherg3
```

rmrg コマンドについて詳しくは、 「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

同値の管理

このセクションのトピックでは、同値の属性を作成、削除、およびリストする方法について説明します。

同値の作成

リソース間で同値を作成するには、**mkequ** コマンドを使用します。

mkequ コマンドについて詳しくは、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

例えば、リソース・クラス IBM.NetworkInterface の、Linux システム node01 および node02 上にある 2 つのイーサネット・インターフェース eth0 の NetworkInterfaces という静的同値を作成するには、以下のように入力します。

```
mkequ NetworkInterfaces IBM.NetworkInterface:eth0:node01,eth0:node02
```

Linux システムのクラスター内の使用可能なすべてのイーサネット・インターフェースを含む、NetworkInterfacesDynamic という動的同値を作成するには、以下のように入力します。

```
mkequ -D "Name like 'eth%'" NetworkInterfacesDynamic IBM.NetworkInterface
```

AIX システムのクラスターでは、以下のように入力します。

```
mkequ -D "Name like 'en%'" NetworkInterfacesDynamic IBM.NetworkInterface
```

1 つ以上の同値のリスト

1 つ以上の同値をリストするには、**lsequ** コマンドを使用します。

lsequ コマンドの詳細については、*System Automation for Multiplatforms* を参照してください。

同値名を省略すると、定義されたすべての 同値がリストされます。同値を指定すると、 この同値の永続属性がリストされます。オペランドとして属性名を 指定すると、同値に指定された属性がリストされます。

例えば、同値 NetworkInterfaces の永続属性をリストするには、以下のように入力します。

```
lsequ -A p -e NetworkInterfaces
```

同値の変更

同値に含まれるリソースを、追加、除去、または完全に置き換えるには、**chequ** コマンドを使用します。

chequ コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。

また、このコマンドを使用して同値の名前を変更することもできます。

例えば、リソース・クラス IBM.NetworkInterface に属す、Linux システム node01 上にあるリソース eth1 を NetworkInterfaces という同値に追加するには、以下のように入力します。

```
chequ -u a NetworkInterfaces IBM.NetworkInterface:eth1:node01
```

同値の除去

1 つ以上の同値を除去するには、**rmequ** コマンドを使用します。

rmequ コマンドの詳細については、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

オペランドとして同値 名を使用するか、選択文字列を使用して、1 つ以上の同値を指定します。

例えば、NetworkInterfaces という同値を除去するには、以下のように入力します。

```
rmequ NetworkInterfaces
```

関係の管理

このセクションのトピックでは、関係を作成、リスト、変更、および削除する方法について説明します。

関係の作成

1 つのソース・リソースと 1 つ以上のターゲット・リソースの間の関係を作成するには、**mkrel** コマンドを使用します。

ソース・リソースは、リソース・グループのメンバーであるか、それ自体がリソース・グループである、管理対象リソースでなければなりません。オプションで、ターゲット・リソースは、リソース・グループのメンバーにすることができます。

例えば、クラス IBM.Application のソース・リソース FloatWebServerA について、クラス IBM.Application のターゲット・リソース FloatWebServerB に対する AntiCollocated 関係を条件「IfOnline」および名前「Rel1」で定義するには、以下のように入力します。

```
mkrel -p anticollocated -o ifonline -S IBM.Application:FloatWebServerA
      -G IBM.Application:FloatWebServerB Rel1
```

詳しくは、**mkrel** マニュアル・ページか、「*System Automation for Multiplatforms* リファレンス・ガイド」の **mkrel** コマンドの説明のいずれかを参照してください。

関係のリスト

関係をリストするには、**lsrel** コマンドを使用します。

関係名を入力しないと、現在定義されているすべての関係が リストされます。

```
lsrel

Displaying Managed Relations :

Name Class:Resource:Node[Source]     ResourceGroup[Source]
Rel1 IBM.Application:FloatWebServerA RG_WebApp
```

-M オプションを使用して関係名を指定すると、指定された関係の永続属性がリストされます。例えば、関係 Rel1 の属性をリストするには、以下のように入力します。

```
lsrel -M Rel1

Displaying Managed Relationship Information:
for Managed Relationship "Rel1".


Managed Relationship 1:
        Name                          = Rel1
        Class:Resource:Node[Source] = IBM.Application:FloatWebServerA
        Class:Resource:Node[Target] = {IBM.Application:FloatWebServerB}
        Relationship                  = AntiCollocated
        Conditional                   = IfOnline
        ResourceGroup[Source]         = RG_WebApp
```

IBM.Application: FloatWebServerA が -S オプションのソースであるすべての関係をリストすると、以下のような類似した出力結果が得られます。

```
lsrel -S IBM.Application:FloatWebServerA

Displaying Managed Relationship Information:

Managed Relationship 1:
        Name                          = Rel1
        Class:Resource:Node[Source] = IBM.Application:FloatWebServerA
        Class:Resource:Node[Target] = {IBM.Application:FloatWebServerB}
        Relationship                  = AntiCollocated
        Conditional                   = IfOnline
        ResourceGroup[Source]         = RG_WebApp
```

詳しくは、**lsrel** マニュアル・ページか、「*System Automation for Multiplatforms* リファレンス・ガイド」の **lsrel** コマンドの説明のいずれかを参照してください。

関係の変更

関係を変更するには、**chrel** コマンドを使用します。

例えば、Rel1 という名前の関係 (上記で作成) を AntiAffinity に変更するには、以下のように入力します。

```
chrel -p antiaffinity Rel1
```

詳しくは、**chrel** マニュアル・ページか、「*IBM Tivoli System Automation for Multiplatforms* リファレンス」の **chrel** コマンドの説明のいずれかを参照してください。

関係の除去

ソース・リソースとターゲット・リソースの間の関係を除去するには、 **rmrel** コマンドを使用します。

例えば、クラス IBM.Application のソース・リソース FloatWebServerA の関係を除去するには、以下のように入力します。

```
rmrel -S IBM.Application:FloatWebServerA
```

rmrel コマンドの詳細については、*System Automation for Multiplatforms* リファレンス・ガイドを参照してください。

XML 自動化ポリシーの管理

自動化ポリシーを管理することで、System Automation for Multiplatforms の自動化動作を定義できます。

自動化ポリシーは、System Automation for Multiplatforms の主要な要素の 1 つです。 このポリシーを使用して、System Automation for Multiplatforms の自動化動作 (アクションと決定) を判別するルールを定義します。ポリシーには、リソースの定義、リソース・グループの定義、リソース間およびグループ間 (またはこのいずれか)、および同値間の関係が含まれています。通常、管理者のメインタスクは、このポリシーを保守し、正確かつ再作成可能なものにすることです。

注 : ポリシーの変更によって、変更されたリソースに関するいくつかのイベントが生成されます。さらに、アクティブなポリシー内にある他のリソースに関するいくつかのイベントが生成される場合があります。

1 つ以上のポリシーを保守するには、**sampolicy** コマンドを使用します。 このコマンドは、以下の 3 つの異なるフレーバーで使用できます。

1. ポリシーを保管し、後の時点で復元する。
2. ポリシーを完全に置き換える。
3. アクティブなポリシーを更新する。

sampolicy コマンドは、 リソース・グループ、 関係、 同値、 IBM.Application、 IBM.ServiceIP、 IBM.AgFileSystem、 IBM.Test リソース、制御パラメーター (samctrl)、ならびに IBM.TieBreaker リソースを処理します。

sampolicy コマンドを使用したポリシーの管理

以降のセクションでは、 **sampolicy** コマンドを使用したポリシーの管理方法について説明します。詳細については、「*IBM Tivoli System Automation for Multiplatforms* リファレンス」のコマンドの説明を参照してください。

sampolicy 確認要求を使用不可にする:

ポリシーを活動化、非活動化、または更新するために **sampolicy** コマンドを実行すると、構文検査の完了後に、アクションを確認するようにプロンプトが出されます。例えばバッチ・スクリプトで、この確認要求を使用不可にするには、 **sampolicy** コマンドの -q パラメーターを使用できます。

例: XML ファイル myPolicy.xml に指定されたポリシーを活動化するときに確認要求を抑制するには、以下のコマンドを実行します。

```
sampolicy -q -a myPolicy.xml
```

ポリシーの保管

アクティブ・ポリシーをファイル /usr/xml/myPolicy.xml に保存するには、次のコマンドを実行します。

```
sampolicy -s /usr/xml/myPolicy.xml
```

保管された構成の活動化、復元、および置換

sampolicy コマンドを使用すると、ポリシーを活動化、復元、または置換できます。

例: XML ファイル `/usr/xml/myPolicy.xml` に指定されたポリシーを活動化するには、以下のコマンドを実行します。

```
sampolicy -a /usr/xml/myPolicy.xml
```

アクティブなポリシーの更新

sampolicy コマンドを使用すると、現在アクティブなポリシーを、混乱を 生じさせない方法で更新できます。更新処理については、次のセクションで説明します。

例: XML ファイル `update.xml` に指定されたエレメントでアクティブなポリシーを更新するには、以下のコマンドを実行します。

```
sampolicy -u update.xml
```

アクティブなポリシーの非活動化

現在アクティブなポリシーを非活動化するには、以下のコマンドを実行します。

```
sampolicy -d
```

ポリシー情報の照会

共通するポリシー情報 (例えば、PolicyName、PolicyDescription、PolicyAuthor) を表示するには、**sampolicy** コマンドで `-i` フラグを使用します。

例:

ファイル `/usr/xml/myPolicy.xml` から共通するポリシー情報を表示するには、以下のコマンドを使用します。

```
sampolicy -i /usr/xml/myPolicy.xml
```

自動化ポリシーでの XML の使用

自動化ポリシーを XML ファイルに定義できます。

この方法では、多数のオプションが提供されます。

- XML ファイルに System Automation for Multiplatforms ドメインの初期自動化ポリシーを定義できます。
- 自動化ポリシーを更新できます。

XML ポリシー・ファイルは、手動で、または System Automation Application Manager バージョン 3.2.2 で提供されているグラフィカル・ポリシー・エディターを使用して、作成および編集できます。

XML を使用した System Automation for Multiplatforms ドメインの初期自動化ポリシーの定義

このシナリオでは、自動化ポリシーを定義および活動化することにより、新規の System Automation for Multiplatforms ドメインをセットアップするために実行する必要があるステップの概要を説明します。 このシナリオでは、以下の前提条件ステップを既に実行済みであることを想定しています。

1. System Automation for Multiplatforms が、クラスターを形成するすべてのノードにインストールされている。
2. *System Automation for Multiplatforms* インストールと構成のガイドの説明に従って、クラスターを定義し、開始してある。

以下のステップを実行します。

1. XML 内の必要な XML エレメントを定義します。これらのエレメントについて詳しくは、「*Tivoli System Automation for Multiplatforms* リファレンス・ガイド」のを参照してください (ポリシー定義の例があります)。
2. ポリシー・プール・ディレクトリーに XML ファイルを保存します。
3. 次のコマンドを発行して自動化ポリシーを検査し、警告やエラーがないか確認します。

   ```
   sampolicy -c <file_name>
   ```

   <file_name> は、検査対象の自動化ポリシーが定義されている XML ファイルの完全修飾名です。 自動化ポリシーの検査が終わると、以下のように結果が表示されます。

   - 問題が検出されなかった場合は、自動化ポリシーを活動化する準備ができていることを確認するメッセージが表示されます。
   - 検査中に問題が見つかった場合は、問題のリストが表示されます。 検出されたエラーをすべて解決しないと、ポリシーは活動化できません。警告が発行された場合は、問題を解決していなくてもポリシーを活動化することは可能ですが、基になっている問題を回避できるかどうかを検査してください。
4. 次のコマンドを発行して自動化ポリシーを活動化します。

   ```
   sampolicy -a <file_name>
   ```

   <file_name> は、活動化する自動化ポリシーが定義されている XML ファイルの完全修飾名です。

XML を使用しての自動化ポリシーの更新

以下のステップを実行します。

1. アクティブな自動化ポリシーを XML ファイルに保存します。これを行うには、コマンド行から以下のコマンドを実行します。

   ```
   sampolicy -s <file_name>
   ```

   ここで、<file_name> は、自動化ポリシーの保存先 XML ファイルの完全修飾名を表します。現在アクティブな自動化ポリシーが、XML ポリシー・ファイルを活動化した結果であるか、コマンド行のコマンドを使用して定義されたものかに関係なく、このステップを実行する必要があります。特に、これには、現在アクティブなポリシーを将来活動化して参照するためにバックアップできるという利点があります。
2. XML ファイルのコピーを作成して、現在アクティブな自動化ポリシーの定義が保存されるように、別の、意味のある名前で保存します。
3. XML ポリシー・ファイルで自動化ポリシーに必要な変更を行います。 これは、例えば、新規リソースとグループの削除または作成、グループ・メンバーシップの変更、属性値の変更、関係の追加、除去、または変更を意味します。
4. 次のコマンドを発行して自動化ポリシーを検査し、警告やエラーがないか確認します。

   ```
   sampolicy -c <file_name>
   ```

   <file_name> は、検査対象の自動化ポリシーが定義されている XML の完全修飾名です。 検査結果が表示されます。

   - 問題が検出されなかった場合は、自動化ポリシーを活動化する準備ができていることを確認するメッセージが表示されます。
   - 検査中に問題が見つかった場合は、問題のリストが表示されます。 検出されたエラーをすべて解決しないと、ポリシーは活動化できません。警告が発行された場合は、問題を解決していなくてもポリシーを活動化することは可能ですが、基になっている問題を回避できないかどうかを検査する必要があります。

自動化ポリシーを活動化する準備ができていることが確かな場合は、ポリシーを活動化するためにいくつかのオプションを使用できます。

中断を伴う活動化

このタイプのポリシー活動化では、新規の自動化ポリシーによって古いポリシーは完全に置き換えられます。つまり、現在アクティブな定義はすべて除去され、すべてのリソースが停止されてから、新規のポリシー・ファイルが活動化されます。 その影響は、アクティブな自動化ポリシーを非活動化して、新規のポリシーを活動化した場合と同じです。

この方法で自動化ポリシーを活動化するには、以下のコマンドを使用します。

```
sampolicy -a <file_name>
```

ここで、<file_name> は、 活動化する自動化ポリシーを含む XML ファイルの完全修飾名を表します。

差分の活動化 (新規ポリシーにない現在アクティブなエレメントは保存)

このタイプのポリシーの活動化は、混乱を生じさせない活動化で、アクティブなポリシーに現在定義されていて、新規ポリシーには記載されていないエレメントはそのまま変更せずに残します。

この方法で自動化ポリシーを活動化するには、以下のコマンドを使用します。

```
sampolicy -u <file_name>
```

ここで、<file_name> は、 活動化する自動化ポリシーを含む XML ファイルの完全修飾名を表します。

この方法で自動化ポリシーを活動化すると、以下の影響があります。

- XML ポリシー・ファイルに定義されている新規エレメントは、現在アクティブな自動化ポリシーに追加されます。
- 現在アクティブな自動化に存在するエレメントに対する XML ポリシー・ファイルでの変更がすべて、適用されます。
- 現在アクティブな自動化ポリシーと新規自動化ポリシーの両方に同じように定義されているエレメントには影響はありません。
- 現在アクティブな自動化ポリシーには存在して、新規自動化ポリシーには存在しないエレメントには影響はありません。

このポリシー活動化方法では、新規エレメントおよび既存エレメントへの変更のみを含む XML ポリシー・ファイルを使用して、現在アクティブな自動化ポリシーを拡張および変更できます。ただし、そのような XML ポリシー・ファイルは、活動化するために正しい形式で、有効かつエラー・フリーであることが必要であるという意味で、完全でなければならないことに注意してください。

差分の活動化 (新規ポリシーにない現在アクティブなエレメントは削除)

このタイプのポリシーの活動化は、混乱を生じさせない活動化ですが、現在アクティブな自動化ポリシーには存在し、活動化する自動化ポリシーにはないエレメントは除去されます。

この方法で自動化ポリシーを活動化するには、以下のコマンドを使用します。

```
sampolicy -r <file_name>
```

ここで、<file_name> は、 活動化する自動化ポリシーを含む XML ファイルの名前を表します。

この方法で自動化ポリシーを活動化すると、以下の影響があります。

- 現在アクティブな自動化ポリシーと新規自動化ポリシーの両方に同じように定義されているエレメントには影響はありません。
- XML ポリシー・ファイルに定義されている新規エレメントは、現在アクティブな自動化ポリシーに追加されます。
- 現在アクティブな自動化に存在するエレメントに対する XML ポリシー・ファイルでの変更がすべて、適用されます。
- 現在アクティブな自動化ポリシーには存在して、新規の自動化ポリシーには存在しないエレメントは除去されます。

このタイプのポリシー活動化を使用すると、新規エレメント、既存エレメントへの変更、および影響を受けないエレメントのみを含む XML ポリシー・ファイルを使用して、現在アクティブな自動化ポリシーを拡張および変更できます。ただし、そのような XML ポリシー・ファイルは、活動化するために正

しい形式で、有効かつエラー・フリーであることが必要であるという意味で、完全でなければならないことに注意してください。

ポリシー・テンプレートの使用

複数の反復可能な値を含む大規模なポリシーを使用する場合、またはポリシーをある環境から別の環境により迅速に移動する必要がある場合は、 **sampolicy** コマンドによって提供されるテンプレート処理機能を使用できます。

実際の値の代わりに、%%parname%% のようなパラメーターを XML ファイルで使用します。ここで、parname はパラメーターの名前です。 このパラメーターは、XML ポリシー・テンプレート・ファイルの var エレメントに定義する必要があります。

このエレメントには以下の属性があります。

name
パラメーターの名前。

value
パラメーターの値。

1 つのパラメーターを複数の場所と複数のファイルで使用できますが、変更は 1 つの場所でのみ行う必要があります。これによって、ポリシーをある環境から別の環境にマイグレーションできます。

また、他の XML ファイルを XML ポリシー・テンプレート・ファイルに含めて、XML ポリシー・テンプレートをより操作しやすくすることもできます。

新規の XML ファイルは include エレメントを使用して添付できます。そのノード値には、添付されたファイルへのパスが含まれている必要があります。このパスは相対パスにできます。

テンプレートの XML ファイルには別のルート・エレメントである AutomationPolicyTemplate があります。テンプレートを使用してポリシーを活動化または更新する場合は、**sampolicy** コマンドで -t パラメーターを使用する必要があります。

例:

```
sampolicy -a -t top.xml
```

top.xml ファイルの内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<AutomationPolicyTemplate productID="SAM" version="4.1.0"
          xmlns="http://www.ibm.com/TSA/Policy.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.ibm.com/TSA/Policy.xsd
          SAMPolicyTemplate.xsd ">
    <PolicyInformation>
        <PolicyName>template</PolicyName>
        <AutomationDomainName>%%domain%%</AutomationDomainName>
        <PolicyToken>1.0</PolicyToken>
        <PolicyDescription>MyDescription</PolicyDescription>
         <PolicyAuthor>admin</PolicyAuthor>
    </PolicyInformation>
    <var name="domain" value="lnx"/>
    <var name="node1" value="lnxcm11x"/>
    <include>internal.xml</include>
</AutomationPolicyTemplate>
```

internal.xml の内容:

```
<?xml version="1.0" encoding="UTF-8"?>
<AutomationPolicy productID="SAM" version="4.1.0"
          xmlns="http://www.ibm.com/TSA/Policy.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.ibm.com/TSA/Policy.xsd
          SAMPolicy.xsd">

    <ControlInformation>
        <Timeout>60</Timeout>
        <RetryCount>3</RetryCount>
```

```
        <ResourceRestartTimeout>5</ResourceRestartTimeout>
    </ControlInformation>
    <Resource name="T1" class="IBM.Test" node="%%node1%%">
        <ClassAttributesReference>
            <IBM.TestAttributes name="IBM.Test.T1"/>
        </ClassAttributesReference>
    </Resource>
    <IBM.TestAttributes name="IBM.Test.T1" >
         <TimeToStart>0</TimeToStart>
         <TimeToStop>0</TimeToStop>
         <WriteToSyslog>0</WriteToSyslog>
    </IBM.TestAttributes>
</AutomationPolicy>
```

上記のポリシーは、テンプレートを使用して単純な IBM.Test リソースを作成します。結果として作成される XML ポリシーには、すべての組み込み XML ファイルおよび置き換えられるパラメーターが入ります。このポリシーは、最上位ファイルの名前にサフィックス .complete.tmp を付けて現行フォルダーに保存されます。

コマンド **sampolicy -a -t top.xml** を使用して上記の 2 つのサンプル・ファイル (top.xml および internal.xml)ファイルから生成されるファイル top.xml.complete.tmp は以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<AutomationPolicy productID="SAM" version="4.1.0"
          xmlns="http://www.ibm.com/TSA/Policy.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.ibm.com/TSA/Policy.xsd SAMPolicy.xsd">

    <PolicyInformation>
        <PolicyName>template</PolicyName>
        <AutomationDomainName>lnx</AutomationDomainName>
        <PolicyToken>1.0</PolicyToken>
        <PolicyDescription>MyDescription</PolicyDescription>
        <PolicyAuthor>admin</PolicyAuthor>
    </PolicyInformation>

    <ControlInformation>
        <Timeout>60</Timeout>
        <RetryCount>3</RetryCount>
        <ResourceRestartTimeout>5</ResourceRestartTimeout>
    </ControlInformation>

    <Resource name="T1" class="IBM.Test" node="lnxcm11x">
        <ClassAttributesReference>
        <IBM.TestAttributes name="IBM.Test.T1"/>
        </ClassAttributesReference>
    </Resource>

    <IBM.TestAttributes name="IBM.Test.T1" >
        <TimeToStart>0</TimeToStart>
        <TimeToStop>0</TimeToStop>
        <WriteToSyslog>0</WriteToSyslog>
    </IBM.TestAttributes>

</AutomationPolicy>
```

シャドー・リソースの使用

シャドー・リソースを使用すると、異なるノード上で稼働している固定リソース間の関係をセットアップできます。

シャドー・リソースには、以下の特性があります。

- これは、別の固定リソースの OpState をモニターするクラス IBM.Application の固定リソースです。
- これは同値のメンバーで、SelectFromPolicy 属性値が NoControl に設定されています。すなわち System Automation for Multiplatforms がメンバー・リソースの OpState のみを評価して、開始または停止を行わないように指定しています。

タイプ DependsOn の関係 (浮動リソースから、異なるノードで実行する 2 つ以上の固定リソースのいずれかへの) を 定義する場合は、DependsOn 関係が StartAfter および StopAfter の両方の動作を引き起こすため、 シャドー・リソースを使用する必要があります。

この種の DependsOn 関係をシャドー・リソースの定義なしにセットアップすると、好ましくない自動化動作を起こします。

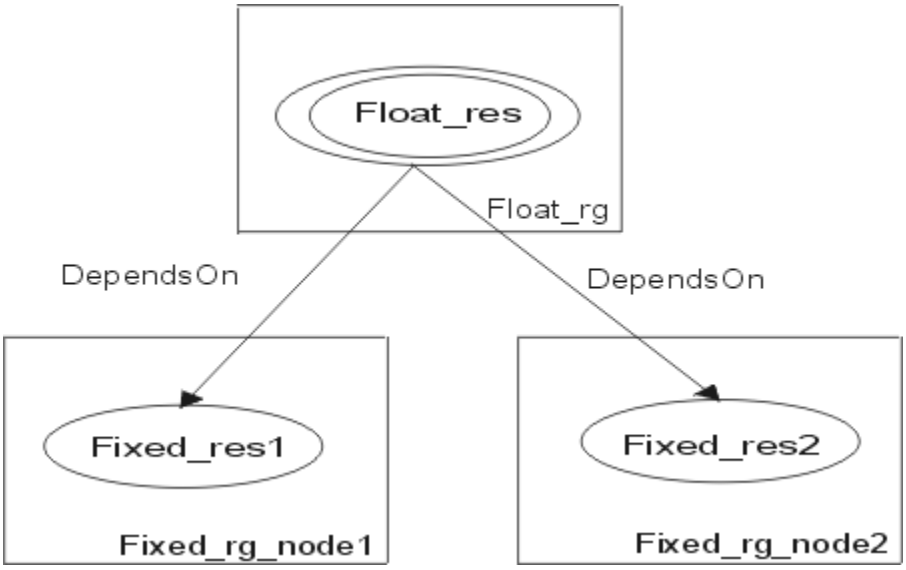


図 *86.* シナリオ *1:* シャドー・リソースなしの *DependsOn* 関係

シナリオ 1 では、使用できない固定リソースが一方のみの場合でも System Automation for Multiplatforms は浮動リソースを停止し、両方の固定リソースが使用可能な場合にのみ System Automation for Multiplatforms は浮動リソースを再始動できます。

本来あるべき自動化動作を行う場合は、NoControl 同値のメンバーであるシャドー・リソースを作成します。

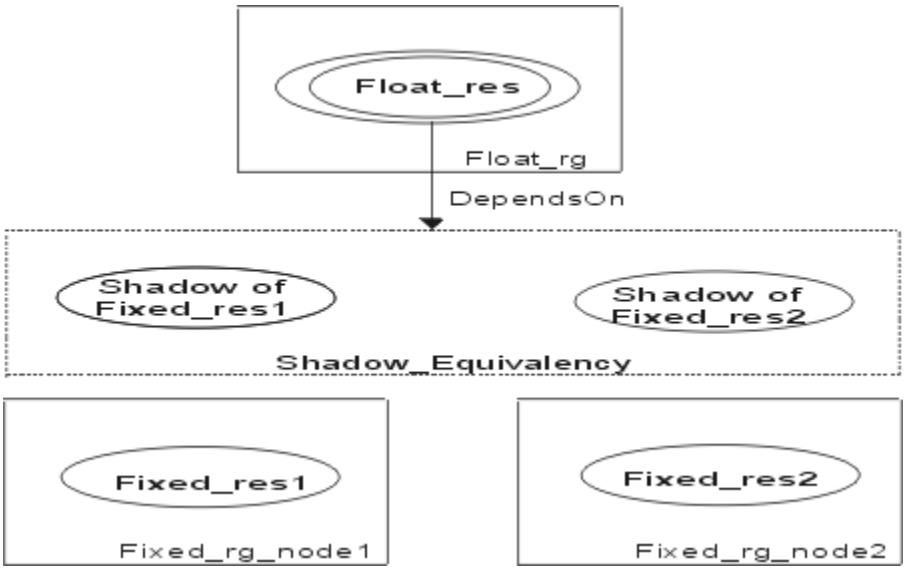


図 *87.* シャドー・リソース付きの *DependsOn* 関係

シナリオ 2 では、現在選択されている同値のメンバーの操作状態が「オフライン」に変わる場合は、DependsOn 関係の ForceDown 動作により、System Automation for Multiplatforms は、浮動リソースを停止しますが、ここで、同値の他のメンバーで少なくとも 1 つがオンラインならば System Automation for Multiplatforms は浮動リソースを再始動できます。

シャドー・リソースの定義

このタスクについて

シャドー・リソースを作成するには、以下のステップを実行します。

1. シャドー・リソースのコマンド・スクリプトを作成する。

   System Automation for Multiplatforms が使用するのは MonitorCommand のみであっても、有効な StartCommand と有効な StopCommand の両方の指定も必要です。 MonitorCommand は、シャドー対象リソースの OpState を照会します。 これは、以下のいずれかの方法で行えます。

   - 以下を使用して OpState を照会することによって

     ```
     ‘lsrsrc -s ‘Name like “<res> “’IBM.Application OpState
     ```

   - 元のリソースの MonitorCommand を複写することによって
   - 元のリソースの OpState をファイルに保管し、シャドー・リソースの MonitorCommand 内でそれを読み取ることによって

   例:

   以下のサンプル・スクリプトには、リソース "fixed_rs1" のシャドー・リソースの必須コマンドが含まれています。

   ```
   #!/bin/ksh
   #
   # shadow_sample.sh
   #
   # init section
   #

   Action=${1:-status}
   ResName=${2:-myresource}

   UNKNOWN=0
   ONLINE=1
   OFFLINE=2
   FAILED_OFFLINE=3

   export CT_MANAGEMENT_SCOPE=2    # necessary to execute SA MP commands

   #
   # main section
   #
   case ${Action} in
     start)
            # is not executed .. so irrelevant
            RC=0
            ;;
     stop)
            # is not executed .. so irrelevant
            RC=0
            ;;
   status|*)
   RCval=$(lsrsrc -xt -s 'Name="'${ResName}'"'
      IBM.Application OpState)
            RCx=${RCval:-2}
            case ${RCx} in
              [1]*)   RC=${ONLINE}
                      ;;
              *)      RC=${OFFLINE}
                      ;;
            esac
            #logger -i -t "$(basename $0)" "${ResName} monitored: ${RC}"
   esac

   exit ${RC}
   ```

2. リソースのそれぞれに以下のようなコマンドを使用して、シャドー・リソースを作成する。

   ```
   # mkrsrc IBM.Application ¥
       Name="fixed_rs1_shadow" ¥
       ResourceType=0 ¥
   ```

```
    NodeNameList="{'node1'}" ¥
    UserName="root" ¥
    StartCommand="/samplepath/shadow_sample.sh start fixed_rs1" ¥
    StopCommand="/samplepath/shadow_sample.sh stop fixed_rs1" ¥
    MonitorCommand="/samplepath/shadow_sample.sh status fixed_rs1" ¥
    MonitorCommandPeriod=10 ¥
    RunCommandsSync=1
```

3. NoControl の SelectFromPolicy 属性値を用いて同値を作成する。

   ステップ 2 で作成した固定リソースを含む同値を作成するには、以下のコマンドを使用します。

```
# mkequ -p A,NoControl <equ-name> ¥
IBM.Application:<fixed-resource1>:<node-name1>, ¥
<fixed-resource2>:<node-name2>[,...]
```

   例:

```
mkequ -p A,NoControl shadow_equ ¥
IBM.Application:fixed_rs1_shadow:node1,fixed_rs2_shadow:node2
```

   Ordered ポリシーを定義するには、-p O,NoControl を使用します。

4. 浮動リソースから同値 shadow_equ への DependsOn 関係を作成する

```
mkrel -p DependsOn -S IBM.Application:float1 -G ¥
IBM.Equivalency:shadow_equ float1-depon-shadow_equ
```

第 4 章 稼働中

稼働中のシステムを操作する方法および実行できるタスクについて説明します。

クラスターの作成と保守

既存のインフラストラクチャーに高可用性を適用するには、1 つ以上のクラスターを作成して保守する必要があります。

クラスターの作成

クラスターの作成方法、タイ・ブレーカーのセットアップ、クラスターへのノードの追加、および System Automation for Multiplatforms リカバリー RM デーモン (IBM.RecoveryRM) の状況の確認方法について説明します。

クラスターを定義および管理するコマンド

クラスター定義を処理するには、Reliable Scalable Cluster Technology (RSCT) コマンドを使用して、クラスターを定義および管理します。

preprpnode

preprpnode コマンドは、コマンドを実行するノードにセキュリティーを準備し、クラスターに定義できるようにします。これにより、クラスター操作をこのノードで処理できます。**mkrpdomain** または **addrpnode** コマンドを使用してノードをクラスターに結合できるようにするには、このコマンドを実行しておく必要があります。

mkrpdomain

このコマンドは、指定されたクラスターの名前および含まれるノードのリストに従って、クラスターを作成します。

lsrpdomain

lsrpdomain コマンドは、コマンドを実行するノードのクラスター・メンバーシップについての情報を表示します。

startrpdomain、stoprpdomain

これらのコマンドは、クラスターを始動または停止するために使用されます。

rmrpdomain

このコマンドは、クラスター定義を削除します。

addrpnode、rmrpnode

これらのコマンドは、現在のオンライン・ピア・ドメイン内の 1 つ以上のクラスター・ノードを追加または削除します。

lsrpnode

このコマンドは、クラスター・ノードおよびそれらの作動状態 (OpState) のリストを表示します。このコマンドは、クラスター内のオンライン状態のノードで最も役立ちます。

startrpnode、stoprpnode

これらのコマンドは、それぞれクラスター内で個々のノードをオンラインおよびオフラインにするために使用します。多くの場合、これらは、特定のシステムでの保守に使用されます。通常、ノードが停止された後で、保守が適用されます。ノードが再始動されると、ノードがクラスターに再結合されます。これらのコマンドは、クラスター内のオンライン状態の任意のノードから使用できます。

lssrc

このコマンドは、RSCT サブシステムの状況をリストします。

startsrc、stopsrc

これらのコマンドは、個々の RSCT サブシステムを始動または停止します。

コマンドについて詳しくは、以下を参照してください。

- *System Automation for Multiplatforms* リファレンス・ガイド
- Reliable Cluster Software Technology Knowledge Center

.

2 ノード・クラスターの作成

2 ノード・クラスターを作成する方法および発生する可能性があるエラーについて説明します。

クラスターの作成を開始する前に、クラスター・ノードのネットワーキングが正しくセットアップされているかどうかを確認することが重要です。

- 各クラスター・ノードの IP、ネットマスク、およびブロードキャスト・アドレ スに一貫性がある必要があります。
- ネーム解決が正しく、DNS 項目に一貫性がある必要があります。または、各ローカル `/etc/hosts` ファイルにすべてのクラスター・ノードの項目が含まれている必要があります。長いホスト名と短いホスト名は、**preprpnode** および **mkrpdomain** などの RSCT コマンドで大/小文字を区別して処理されます。そのため、それらのスペルがホスト名および `/etc/hosts` 項目として正しいことを確認してください。すべての文字を小文字または大文字にすると、分かりやすくなります。
- 同じホスト名を参照する外部の項目 (例えば、`127.0.0.2 my_hostname` などの項目)を `/etc/hosts` テーブルから削除します。このような項目は、オペレーティング・システムをインストールする際に作成されることがありますが、クラスター・インフラストラクチャーとは矛盾しています。
- 各ノード上で同一サブネットへの複数のネットワーク・インターフェースを定義しないでください。
- ローカルのファイアウォールを含むすべてのファイアウォールが、クラスター・ノード間の ICMP ping および IP トラフィックに対して、次のポート番号の通過を許可する必要があります。
  - RSCT `cthats` サブシステム用の `12347/udp`。
  - RSCT `cthags` サブシステム用の `12348/udp`。
  - RSCT `rmc` サブシステム用の `657/udp`。
  - RSCT `rmc` サブシステム用の `657/tcp`。

クラスター・ノード間のネットワーキングのセットアップについての詳しい手順と推奨事項については、「*System Automation for Multiplatforms* インストールと構成のガイド」を参照してください。

以下の手順で 2 ノード・クラスターを作成します。

1. クラスターの各ノードにログオンします。
   - 各ノード上でシェルを開き、root としてログオンします。**sudo** コマンドを使用する適切な権限を持つ非 root ユーザー ID を使用することもできます。
   - 各ノードで、環境変数 `CT_MANAGEMENT_SCOPE` を 2 に設定およびエクスポートします。この環境変数は RSCT および System Automation for Multiplatforms コマンドを実行するときには常に設定する必要があるため、この変数は永続的に設定します。例えば、`/etc/profile` 内で変数を設定します。`CT_MANAGEMENT_SCOPE` 変数は、すべての RSCT および System Automation for Multiplatforms コマンドがローカル (1) に、またはクラスター (2) スコープ内で、実行されるように構成します。
   - すべてのノードで **preprpnode** コマンドを実行し、クラスター・ノード間の通信を許可します。

     ```
     preprpnode node01 node02
     ```

2. これで、ノード node01 および node02 を含む SA_Domain という名前のクラスターを作成する準備ができました。コマンドは、2 つのノードのいずれかから 1 回入力する必要があります。

```
mkrpdomain SA_Domain node01 node02
```

mkrpdomain コマンドで RSCT ピア・ドメイン (クラスター) を作成する場合、ピア・ドメイン名に使用できる文字は、A から Z、a から z、0 から 9、ピリオド (.)、および下線 (_) に制限されます。ドメイン名は大/小文字を区別して処理されます。

クラスター SA_Domain の状況を確認するには、**lsrpdomain** コマンドを入力します。

```
lsrpdomain
```

出力:

```
Name       OpState RSCTActiveVersion MixedVersions TSPort GSPort
SA_Domain  Offline 3.1.5.3           No            12347  12348
```

クラスターは定義済みですが、まだ始動されていないため、オフラインです。

3. -k フラグを使用して、クラスター共有秘密鍵 (CSSK) を設定できます。CSSK は、ピア・ドメイン内のノード間で送信される制御メッセージを認証するために、RSCT コンポーネントによって使用されます。デフォルトでは、CSSK は使用不可です (つまり、CSSKTYPE_None に設定されています)。メッセージ認証を使用可能にするには、-k フラグを指定して CSSKTYPE_DES_MD5 などの CSSK 値を使用します。メッセージ認証を使用可能にすると、パフォーマンスに影響します。暗号化アルゴリズムの複雑度によってこの影響が変わります。

メッセージ認証は、ピア・ドメイン内のノードの時刻クロック (TOD) が、システム時刻に従って相互に 2 分以内で同期していることも必要とします。ノードの TOD がピア・ドメイン全体にわたって同期しているとき、この機能はメッセージのリプレイ・アタックから防御するのに役立ちます。ノードの TOD が、相互に 2 分以内で同期していない場合、送信ノードから受信ノードに渡されるメッセージのうち時差が 2 分より長いものは破棄されます。

-k フラグを使用してメッセージ認証を使用可能にするときに、-r フラグを使用して鍵のリフレッシュ間隔を指定できます。デフォルトでは、鍵は毎日リフレッシュされます。

CSSK 設定のセットアップと管理について詳しくは、「Administering RSCT」ガイドを参照してください。

4. **startrpdomain** コマンドを発行して、クラスターを オンラインにします。

```
startrpdomain SA_Domain
```

始動がまだ進行している間には、**lsrpdomain** コマンドを再び実行すると、クラスターの OpState は **Pending Online** として表示されることがあります。

```
Name       OpState        RSCTActiveVersion MixedVersions TSPort GSPort
SA_Domain  Pending online 3.1.5.3           No            12347  12348
```

しばらくすると、クラスターの始動が完了します。**lsprdomain** コマンドを再び実行すると、クラスターは Online として表示されます。

```
Name       OpState RSCTActiveVersion MixedVersions TSPort GSPort
SA_Domain  Online  3.1.5.3           No            12347 12348
```

クラスター作成中に発生する可能性がある一般的なエラー

- **mkrpdomain** コマンドを発行すると、以下のようなエラー・メッセージが返されることがあります。

```
2632-044 The domain cannot be created due to the following
           errors that were detected while harvesting information from the target
           nodes:node1: 2632-068 This node has the same internal identifier as node02
           and cannot be included in the domain definition.
```

このエラーは、複製されたオペレーティング・システム・イメージを使用している場合に最も頻繁に発生します。

この場合は、クラスター構成をリセットする必要があります。このような問題を解決するには、エラー・メッセージ内で示されているノード上で次のコマンドを実行します。これにより、RSCT ノード ID がリセットされます。

```
/usr/sbin/rsct/install/bin/recfgct
```

次に、**preprpnode** コマンドから始まるクラスター作成手順を繰り返す必要があります。

- 以下のようなエラー・メッセージを受信することがあります。

```
2632-044 The domain cannot be created due to the following
            errors that were detected while harvesting information from the target nodes:
            node1: 2610-418 このコマンドで指定されたリソースまたはリソース・クラスに対する
            アクセス権がありません。
```

ホストのネーム解決を確認します。すべてのローカル /etc/hosts ファイルで、各クラスター・ノードまたはネーム・サーバー (あるいはその両方) のすべての項目が同一であることを確認します。

- 以下のようなエラー・メッセージを受信することがあります。

```
2612-022 A session could not be established with the RMC
            daemon on <node_name>.
```

このエラーは、ファイアウォールがクラスター・ノード間の通信をブロックしている場合に最も頻繁に発生します。問題の解決には、以下のポートをファイアウォール・ソフトウェアで使用可能にします。

```
rmc port 657 TCP IN/OUT Source Port Range: ephemeral port
rmc port 657 UDP IN/OUT Source Port Range: ephemeral port
cthats port 12347 UDP IN/OUT Source Port Range: 1024 - 65535
cthags port 12348 UDP IN/OUT Source Port Range: 1024 - 65535
```

すべてのファイアウォール・ルールで、BROADCAST パケットが cthats ポートを通過できるように許可されている必要があります。ファイアウォールおよびネットワークの問題について詳しくは、「*IBM RSCT* 管理ガイド」の『付録 A. RSCT ネットワークの考慮事項 (Appendix A. RSCT network considerations)』を参照してください。

クラスターへのノードの追加

ノードを追加することにより、クラスターを拡張できます。

2 ノード・クラスターを作成した後、SA_Domain に 3 番目のノードを追加できます。新規ノード node03 をクラスターに追加するには、以下のようにします。

1. 新規ノードが追加されるときには、クラスターおよびそのすべてのメンバー・ノードがオンライン状態である必要があります。**lsrpdomain** および **lsrpnode** コマンドを入力して、現在のクラスター状態を検査します。

```
lsprdomain
Name       OpState RSCTActiveVersion MixedVersions TSPort GSPort
SA_Domain  Online  3.1.5.3           No            12347  12348

lsrpnode

Name    OpState   RSCT Version
node02  Online    3.1.5.3
node01  Online    3.1.5.3
```

2. **preprpnode** コマンドを入力して、既存のノードと新規ノードとの間の通信を許可します。クラスター内の各ノードで **preprpnode** コマンドを入力する必要があります。

   node3 では、セキュリティー情報を準備し、クラスター内の既存のノードとセキュリティー情報を交換します。

node03 に ログオンし、以下のように入力します。

```
preprpnode node01 node02
```

node2 では、セキュリティー情報を準備し、新規クラスター・ノードとセキュリティー情報を交換します。node02 にログオンし、以下のように入力します。

```
preprpnode node03
```

node1 では、セキュリティー情報を準備し、新規クラスター・ノードとセキュリティー情報を交換します。node01 にログオンし、以下のように入力します。

```
preprpnode node03
```

3. クラスターがオンライン状態である場合は、オンライン・ノード node01 または node02 のいずれかから addrpnode コマンドを入力して、クラスターに新規ノード node03 を追加します。

```
addrpnode node03
```

lsrpnode コマンドを入力して、すべてのノードの状況を表示します。

```
Name    OpState   RSCT Version
node02  Online    3.1.0.1
node03  Offline   3.1.0.1
node01  Online    3.1.0.1
```

4. オンライン・ノード node01 または node02 のいずれかから node03 を始動します。

```
startrpnode node03
```

しばらくしてから lsrpnode コマンドを再度入力すると、node03 がオンラインになっています。

```
Name    OpState   RSCT Version
node02  Online    3.1.5.3
node03  Online    3.1.5.3
node01  Online    3.1.5.3
```

詳しくは、「105 ページの『ノードの管理』」を参照してください。

クラスターまたはノードのオフライン化

ノードを保守するか、アプリケーションをアップグレードするために、クラスター全体またはクラスターの個々のノードを停止できます。

147 ページの『クラスターを定義および管理するコマンド』のコマンドのリストから分かるように、コマンド **stoprpnode** および **stoprpdomain** を使用して、個々のクラスター・ノードまたはクラスター全体をオフラインに設定できます。

現在アクティブな自動化ポリシーを処理しない限り、ノードまたはクラスターをオフラインにすることはできません。ノードまたはクラスター上で実行されているアクティブなリソースがある場合、**stoprpnode** および **stoprpdomain** コマンドの実行は拒否されます。ノードまたはクラスターを停止する前に、他のノードにリソースを移動するか、リソースを停止する必要があります。この処理に必要な手順は、このチュートリアルでは扱いません。

ノードまたはクラスターの削除

ハードウェアをアップグレードする場合、またはクラスター構成全体を再編成する場合は、クラスターから個々のノードを除去するか、クラスター定義全体を除去する必要が生じることがあります。

以下のコマンドを使用して、クラスターからノードを除去するか、クラスター全体を除去します。

rmrpnode
クラスターからノードを除去するには、**rmrpnode** を使用する必要があります。**stoprpnode** コマンドを使用して、除去するノードを停止する必要があります。このシステム上のリソースはすべて非アクティブである必要があります。

rmrpdomain
クラスター定義全体を除去するには、**rmrpdomain** コマンドを使用します。クラスターの除去には、クラスター上の各ノードからの クラスター定義の除去も含まれます。クラスターおよびクラスター内のすべてのノードがオンラインになっている必要があります。クラスター内およびそのすべてのノード上で、すべての自動化リソースが事前に停止されている必要があります。停止されていない場合、**rmrpdomain** コマンドの実行は拒否されます。

ノードの置換

永続的なノード障害の場合は、ノードを置換する必要があります。

ノードを置換するには、ファイル /var/ct/cfg/ct_node_id を古いノードから新しいノードにコピーします。ポリシーを変更する必要はありません。置換後のノードは古いノードと同じ機能を果たします。

一度に置換できるノードは 1 つのみです。

ノードの停止

ノードを停止するには、そのノードを自動化から除外します。

このタスクについて

以下のコマンドを使用して、ノードを除外します。

```
samctrl -u a <node_name>
```

以下のコマンドを使用して、ノードを停止します。

```
stoprpnode <node_name>
```

ノード上で現在オンラインであるノードが存在しない場合でも、除外を実行する必要があります。

ノードを再始動して、ノードがオンラインになった後、以下のように入力してそのノードで 自動化を再設定する必要があります。

```
samctrl -u d <node_name>
```

ノードのリブート

ノード上でリソースが稼働している場合、このノードはリブートしないでください。

このタスクについて

まず 以下のコマンドを使用して、稼働中のすべてのリソースを停止します。

```
samctrl -u a <node_name>
```

これで安全にノードをリブートできます。

また、このコマンドにより、このノードでリソースが開始されなくなります。以下のように入力して、再度このノードでリソースを稼働できるようにします。

```
samctrl -u d <node_name>
```

障害時におけるイベントの生成

RSCT では、リソースの動的属性が変更された場合にイベントを生成できます。 これは、イベント応答リソース・マネージャー (ERRM) によって実行されます。イベント応答 リソース・マネージャーは、ユーザーが興味のあるイベントをモニターし (条件と呼びます)、イベントが発生した場合に RMC システムに特定の方法で応答させる (応答と呼びます) ことが できるコマンドのセットを提供します。

例えば、リソース・グループの OpState をサブスクライブし、 状況が変更された場合に E メールを受け取ることができます。また、システム内のさまざまなクリティカル・リソースを モニターすることもできます。このようなイベントの生成方法について詳しくは、「*IBM Reliable Scalable Cluster Technology for Linux, Technical Reference*」(SA22-7893) を参照してください。

自動化アダプターおよびパブリッシャーの制御

エンドツーエンド自動化アダプター、Tivoli Netcool/OMNIbus イベント・パブリッシャー、またはレポート・データ・パブリッシャーを開始または停止します。

構成ダイアログのランチパッド・ウィンドウで、「**制御**」を選択して、「自動化アダプターおよびパブリッシャーの制御」ウィンドウを表示します。タブの解説は、以下の項に述べられています。構成ダイアログの起動方法について詳しくは、System Automation for Multiplatforms インストールと構成のガイドを参照してください。

「自動化アダプター」タブ

「自動化アダプター」タブでは、エンドツーエンド自動化アダプターを開始または停止できます。

「自動化アダプター」タブのフィールド:

目的
エンドツーエンド自動化アダプターの目的の説明。

状況
自動化アダプターの現在の状況。使用可能な状況値は次のとおりです。

開始済み
アダプターは現在稼働中です。提示されるアクションはアダプターの「**停止**」のみです。

停止中
アダプターが現在稼働していません。提示されるアクションはアダプターの「**開始**」のみです。

Unknown
現在、アダプター状況を特定できません。提示されるアクションはアダプターの「**開始**」のみです。

停止中、未構成
アダプターが現在稼働しておらず、アダプター構成はまだ完了していません。アクションを実行できません。構成を完了するには、System Automation for Multiplatforms インストールと構成のガイド上でホスト名または IP アドレスを定義します。

アクション
アダプターを**開始**または**停止**するには、現在のアダプターの状況に応じてボタンを選択します。アクションが完了すると、それに応じて状況が更新されます。

「イベント・パブリッシャーおよびデータ・パブリッシャー」タブ

「**イベント・パブリッシャーおよびデータ・パブリッシャー**」**タブ**では、Tivoli Netcool/OMNIbus イベント・パブリッシャーまたはレポート・データ・パブリッシャーを開始または停止できます。

このタブの「イベント・パブリッシャー」ボックス内のフィールド:

目的
Tivoli Netcool/OMNIbus イベント・パブリッシャーの目的の説明。

状況
パブリッシャーの現在の状況。使用可能な状況値は次のとおりです。

開始済み
Tivoli Netcool/OMNIbus イベント・パブリッシャーは現在稼働中です。 提示されるアクションはパブリッシャーの「**停止**」のみです。

停止中
Tivoli Netcool/OMNIbus イベント・パブリッシャーが現在稼働していません。提示されるアクションはパブリッシャーの「**開始**」のみです。

開始済み、未構成
Tivoli Netcool/OMNIbus イベント・パブリッシャーは現在稼働中ですが、EIF イベント・パブリッシュの構成が System Automation for Multiplatforms インストールと構成のガイドで使用不可になっています。 提示されるアクションはパブリッシャーの「**停止**」のみです。

イベント・パブリッシュが構成されていないため、今すぐ停止アクションを実行してください。

停止中、未構成
Tivoli Netcool/OMNIbus イベント・パブリッシャーは現在稼働しておらず、EIF イベント・パブリッシュの構成が System Automation for Multiplatforms インストールと構成のガイドで使用不可になっています。 アクションを実行できません。

アクション
Tivoli Netcool/OMNIbus イベント・パブリッシャーを**開始**または**停止**するには、現在のアダプターの状況に応じてボタンを選択します。アクションが完了すると、それに応じて状況が更新されます。

このタブの「レポート・データ・パブリッシャー」ボックス内のフィールド:

目的
レポート・データ・パブリッシャーの目的の説明。

状況
パブリッシャーの現在の状況。使用可能な状況値は次のとおりです。

開始済み
レポート・データ・パブリッシャーは現在稼働中です。提示されるアクションはパブリッシャーの「**停止**」のみです。

停止中
レポート・データ・パブリッシャーが現在稼働していません。提示されるアクションはパブリッシャーの「**開始**」のみです。

開始済み、未構成
レポート・データ・パブリッシャーは現在稼働中ですが、レポート・データ収集の構成が System Automation for Multiplatforms インストールと構成のガイドで使用不可になっています。 提示されるアクションはパブリッシャーの「**停止**」のみです。

イベント・パブリッシュが構成されていないため、今すぐ停止アクションを実行してください。

停止中、未構成
レポート・データ・パブリッシャーは現在稼働しておらず、レポート・データ収集の構成が System Automation for Multiplatforms インストールと構成のガイドで使用不可になっています。 アクションを実行できません。

自動化アダプターまたはパブリッシャーの開始または停止には、System Automation for Multiplatforms のネイティブ・コマンド `samadapter` および `samctrl` を使用することもできます。 これらのネイティブ・コマンドを使用する場合は、現在の値を使用して「**リフレッシュ**」ボタンを選択し、自動化アダプターおよびパブリッシャーの状況をリフレッシュします。

リソース・グループとグループ・メンバーの開始および停止

開始要求と停止要求を使用して、リソース・グループまたは個々のリソース・グループ・メンバーをオンラインまたはオフラインにします。

開始要求と停止要求は、常にリソース・グループの NominalState 値を変更します。公称状態自体は変更されません。

コマンド行から開始要求および停止要求を実行依頼するには、以下のコマンドを使用します。

- リソース・グループを開始または停止するには、以下のコマンドを入力します。

```
rgreq -o <start|stop>
```

- リソース・グループ・メンバーを開始または停止するには、以下のコマンドを入力します。

```
rgmbrreq -o <start|stop>
```

開始要求および停止要求のスコープ

開始要求または停止要求のスコープは、それが実行依頼される対象がリソース・グループか、あるいはリソース・グループの特定のメンバーかによって異なります。

- リソース・グループに対して実行依頼された要求は、グループのすべてのメンバーに影響します。 リソース・グループまたはそのメンバーのすべて、あるいはその両方のポリシーで関係が定義された場合は、追加のリソースにも影響します。
- リソース・グループ・メンバーに対して実行依頼された要求が影響するのは、その特定のリソースに限られます。 リソースが関係を持つ場合は、追加のリソースに影響することがあります。

  注 : リソース・グループの必須メンバーに対して出された要求は、リソース・グループの監視状態および複合状態にも影響があります。 これが起こるのは、要求により、メンバーがリソース・グループの本来あるべき状態と矛盾する監視状態になるときです。 この場合、リソース・グループの監視状態は「保留中」状態に変わり、その複合状態は「エラー」に変わります。

  例:

  オペレーターが、本来あるべき状態が「オンライン」のリソース・グループの必須メンバーに対して停止要求を実行依頼すると、メンバー・リソースが停止したときに、リソース・グループの監視状態は「オンラインの保留中」に変わり、その複合状態は「エラー」に変わります。

注 : リソース・グループまたは管理対象リソースの要求リストに含めることができる要求は各ソースにつき 1 つのみであるため、同じソースからの開始要求と停止要求はお互いを置き換えます。

要求は伝搬されるため (例えば、グループからそのメンバーへ、または依存関係のソースからターゲットへ)、同じソースからの目標が競合するいくつかの要求、および優先度が同じいくつかの要求が 1 つのリソースに対して動作できます。この場合、開始要求は停止要求よりも優先度が高くなっています。例えば、グループに停止要求があり、メンバーに同一ソースからの優先度が同じ開始要求がある場合、グループ・メンバーはオンラインのままになります。

グループ間の関係に沿った要求の伝搬によって、従属するグループの本来あるべき状態が次のように変わる場合があります。

- これにより、ノード位置の割り当てに関する動作が変わります。
- 詳しくは、56 ページの『ノード位置の割り当て』内の『ヒント』セクションを参照してください。

開始要求と停止要求の優先順位の変更

優先順位とその「ソース」属性の値によって、要求の優先順位が決まります。「ソース」属性と要求の優先順位は、要求の実行依頼方法に従ってデフォルト値に設定されます。 **rgreq** および **rgmbrreq** コマンドを発行するときに、-S オプションを使用してソースを指定し、-p オプションを使用して要求の優先順位を設定することによって、デフォルト値を指定変更できます。要求の優先順位付け方法の詳細については、171 ページの『要求の優先順位付け』を参照してください。

開始要求と停止要求の取り消し

開始および停止要求は永続的です。これらの要求は、リソースの要求リストに無期限に保存されます。 リストから除去するには、取り消す必要があります。現在アクティブな要求が取り消され、要求リストに他の要求があるときは、リストの次のランキングの要求が活動化されます。

- リソース・グループの要求リストで要求を取り消すには、以下のコマンドを入力します。

```
rgreq -o cancel
```

- リソース・グループ・メンバーの要求リストで要求を取り消すには、以下のコマンドを入力します。

```
rgmbrreq -o cancel
```

要求リストの表示

実行依頼された要求を表示する場合は、**lsrgreq** コマンドを使用します。

例のシナリオ

この例のシナリオで、オペレーターが要求を発行およびキャンセルして、リソース・グループの操作状態を変更する方法、ならびに、**lsrgreq** コマンドを使用して、リソースの要求リストを表示し、要求の成否を追跡する方法を説明します。 以下の例で使用されているコマンドの詳細な説明については、「*IBM Tivoli System Automation for Multiplatforms* リファレンス」を参照してください。

グループ「top-rg」の NominalState は「オフライン」ですが、その OpState は「オンライン」です。

```
lnxcm3x:# lsrg -g top-rg | grep State
Displaying Resource Group information:
For Resource Group "top-rg".

Resource Group 1:
        NominalState         = Offline
        OpState              = Online
        TopGroupNominalState = Offline
```

lsrgreq コマンドの出力では、グループの OpState の原因は、エンドツーエンド自動化からの「アクティブ」開始要求であることを示しています。

```
lnxcm3x:# lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
For Resource Group "top-rg".

Resource Group 1:
        ResourceGroup = top-rg
        Priority      = High
        Action        = start
        Source        = Automation
        NodeList      = {}
        ActiveStatus  = Active
        Token         = 8f5697eb5f84c0f044995b3d00040a5b
        UserID        =
        MoveStatus    = None
```

グループ「top-rg」をオフラインにするために、オペレーターは停止要求を出します。

```
rgreq -o stop top-rg
```

lsrgreq コマンドの出力では、オペレーターの停止要求は、要求リストに追加されたが、優先されなかったことを示しています。 停止要求は、デフォルトの優先順位 (低) で出されたため非アクティブのままであり、エンドツーエンド自動化からの高ランキングの開始要求に対しては優先されません。

```
lnxcm3x:# lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
For Resource Group "top-rg".

Resource Group 1:
        ResourceGroup = top-rg
        Priority      = High
        Action        = start
        Source        = Automation
        NodeList      = {}
        ActiveStatus  = Active
        Token         = 8f5697eb5f84c0f044995b3d00040a5b
        UserID        =
        MoveStatus    = None
```

```
Resource Group 2:
        ResourceGroup = top-rg
        Priority      = low
        Action        = stop
        Source        = Operator
        NodeList      = {}
        ActiveStatus  = InActive
        Token         = 8f5697eb5f84c0f044995dad0007b338
        UserID        =
        MoveStatus    = None
```

lsrg コマンドの出力では、グループがオンラインの状態を保っていることを示しています。

```
lnxcm3x:# lsrg -g top-rg | grep State
Displaying Resource Group information:
For Resource Group "top-rg".

Resource Group 1:
        NominalState         = Offline
        OpState              = Online
        TopGroupNominalState = Offline
```

グループをオフラインにするために、オペレーターは高優先順位の停止要求を出します。

```
rgreq -p high -o stop top-rg
```

lsrgreq コマンドの出力では、この要求がオペレーターの前の低優先順位要求を置き換え、エンドツーエンド自動化からの開始要求に優先し、アクティブ要求になったことを示しています。

```
lnxcm3x:# lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
For Resource Group "top-rg".

Resource Group 1:
        ResourceGroup = top-rg
        Priority      = High
        Action        = start
        Source        = Automation
        NodeList      = {}
        ActiveStatus  = InActive
        Token         = 8f5697eb5f84c0f044995b3d00040a5b
        UserID        =
        MoveStatus    = None

Resource Group 2:
        ResourceGroup = top-rg
        Priority      = High
        Action        = stop
        Source        = Operator
        NodeList      = {}
        ActiveStatus  = Active
        Token         = 8f5697eb5f84c0f044996004000368b1
        UserID        =
        MoveStatus    = None
```

グループは、停止要求により停止します。

```
lnxcm3x:# lsrg -g top-rg | grep State
Displaying Resource Group information:
For Resource Group "top-rg".

Resource Group 1:
        NominalState         = Offline
        OpState              = Offline
        TopGroupNominalState = Offline
```

「top-rg」グループをもう一度オンラインにする場合にオペレーターが行う必要があるのは、最後の停止要求のキャンセルのみで、追加の開始要求は必要ありません。

```
rgreq -o cancel top-rg
```

下記の出力例では、エンドツーエンド自動化からの要求が再度アクティブになるため、「top-rg」グループが要求のキャンセル後に開始したことを示しています。

```
lnxcm3x: # lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
For Resource Group "top-rg".

Resource Group 1:
        ResourceGroup = top-rg
        Priority      = High
        Action        = start
        Source        = Automation
        NodeList      = {}
        ActiveStatus  = Active
        Token         = 8f5697eb5f84c0f044995b3d00040a5b
        UserID        =
        MoveStatus    = None

lnxcm3x:# lsrg -g top-rg | grep State
Displaying Resource Group information:
For Resource Group "top-rg".

Resource Group 1:
Resource Group 1:
        NominalState         = Offline
        OpState              = Online
        TopGroupNominalState = Offline
```

リカバリー・リソース・マネージャーの管理

リカバリー・リソース・マネージャーを管理するために、System Automation for Multiplatforms は、リカバリー・リソース・マネージャー (IBM.RecoveryRM) を RSCT サブシステムとして追加します。

このタスクについて

このデーモンは自動化エンジンであり、リソースの開始時と停止時、およびクラスターのノードでのリソースの配置時に決定されます。自動化ポリシーに記述されているルールに従って機能して、リソースを特定の順序で開始および停止し、オプションでその他の制約および要件を監視します。クラスターまたはノードをオンラインにすると、RecoveryRM デーモンは自動的に始動します。

デーモンの状況およびプロセス ID を取得するには、以下を実行します。

```
lssrc -s IBM.RecoveryRM
```

出力:

```
Subsystem         Group            PID     Status
IBM.RecoveryRM   rsct_rm          18283   active
```

他のすべての RSCT サブシステムと同様に、RecoveryRM デーモンは、以下のコマンドを使用して停止および始動できます。

```
stopsrc -s IBM.RecoveryRM
startsrc -s IBM.RecoveryRM
```

RecoveryRM デーモンは各クラスター・ノード上で実行されますが、その中でマスター・デーモンは 1 つのみです。このデーモンは、必要なすべての決定を 導き出す役割を担います。RecoveryRM マスターのロケーションを検出するには、以下のコマンドを使用します。

```
lssrc -ls IBM.RecoveryRM | grep Master
Master Node Name     : node03 (node number = 3)
```

出力に、マスター・デーモンが実行されているノードが表示されます。この例では、マスター・デーモンは、ノード node03 で実行されています。

他のノード上の RecoveryRM デーモンは、マスター・デーモンまたはそのノードが使用不可になった場合のホット・スタンバイとして機能します。この場合は、別の RecoveryRM デーモンがマスターになります。

マスター機能の引き継ぎは、syslog に記録されますが、その他の点では System Automation for Multiplatforms の自動化機能を中断することなくほぼ透過的に実行されます。

リソースの診断

リソースに関する情報を取得する場合は、**samdiag** コマンドを使用します。

このタスクについて

System Automation for Multiplatforms によって管理されるリソースに関する情報を取得するには、**samdiag** コマンドを使用できます。**samdiag** コマンドについて詳しくは、「*System Automation for Multiplatforms* リファレンス・ガイド」を参照してください。このコマンドは主に、システム上で何が起こり、その理由が何であるかが ユーザーにとって明白でない状態に使用されます。

注 : System Automation for Multiplatforms のリリース 1 で提供されていた **samdiag** コマンドは、マスター・デーモンが実行されていたノードでのみ実行可能でした。このため、同じクラスター内にリリース 1 およびリリース 2 のデーモンが混在する場合、およびマスター・デーモンがリリース 1 のノードにある場合は、リリース 2 のデーモンで **samdiag** を実行するとエラーが生成されます。

apacherg というリソース・グループに関する情報を入手するには、以下のコマンドを使用します。

```
samdiag -g apacherg
```

出力:

```
Diagnosis::Resource: apacherg/ResGroup/IBM.ResourceGroup
    type: CHARM Resource Group
  Status -
      Observed: Offline         - SoftDown
        Desired: Offline         - Requested Offline
       (Nominal: Offline         - Nominal State: Offline)
     Automation: Idle            - CharmBase trigger linked
      Startable: Yes             - Resource is startable
        Binding: Unbound         - Binding status initialized
       Compound: Satisfactory    - Satisfactory

 Resource Based Quorum: None            -
   Members and Memberships:
      +---bind/HasMember                  ---> RA/Float/IBM.Test
   Group Constraint: None
   Binding Constraints:
   Flags:
     None
   Orders:
     Outstanding Order: None            - Resource is Unavailable
   Dependencies:
    Start: Satisfied
      +---InCluster                      ---> Cluster
     Stop: Satisfied
   Binding exceptions:
     There are unbound members.
   Static Relationships:
      +---InCluster                      ---> Cluster
   Dynamic Relationships:
      +---bind/HasMember                 ---> RA/Float/IBM.Test"
```

以下に、 例で入手されたいくつかの情報の解釈を示します。

- ObservedState は、OpState および NominalState と同じ値を示す必要があります。そうなっていない場合は、お客様の地域を担当するサポート・センターに連絡してください。
- DesiredState と NominalState の値が異なる場合は、リソース・グループに対して 要求が発行されたことを示します。
- AutomationState は、「Busy」または「Idle」のいずれかです。「Busy」は、System Automation for Multiplatforms デーモン (IBM.RecoveryRM) が、別のリソース・マネージャーがリソースを開始または停止するのを待機中であることを意味します。これが完了した後は、AutomationState は「Idle」に変更されます。そうなっていない場合は、お客様の地域を担当するサポート・センターに連絡してください。

- Startable の状態が「No」の場合、例えば DependsOn 関係などのいくつかの関係が 正しく設定されていないことを示します。
- BindingState「Unbound」は、Observedstate が 「Offline」であることに伴うものです。これは、リソース・グループがオフラインであることを示します。これがオンラインに設定できるようにする前に、 正しい構成要素を選択するバインディング・ステップを実行する必要があります。その後、BindingState が「Bound」に設定されます。 オフラインであるリソース・グループの BindingState が「Bound」である場合はエラーです。
- CompoundState「Satisfactory」は、ObservedState と DesiredState が同一であることを示します。CompoundStates が「Inhibited」、「Denied」、または「Broken」の場合は、満たされていない関係、または「障害になった」リソースなどのエラーを示します。
- Resource Based Quorum が「None」の場合、グループのすべての浮動リソースが 独自の Resource Based Quorum をサポートしていないこと、および Resource Based Quorum フラグが 設定されていないことを意味します。
- 「Bind/HasMember」は、リソース・グループ apacherg とその浮動メンバー RA/Float/IBM.Test 間の関係を説明しています。バインディング・ステップの 実行時には、それぞれの「Bind/HasMember」関係について構成要素が 選択されます。このリソース・グループがオンラインに設定される前に、この構成要素が一時的に リソース・グループにバインドされます。
- Outstanding Order は、AutomationState を参照します。AutomationState が「idle」で ない場合、保留中のコマンドがここに示されます。
- 開始/停止の Dependencies は、ポリシーによってリソースの開始が妨げられるときを示します。
- Binding exceptions には、BindingState のより詳細な説明が示されます。
- Static relationships は、すべての構成要素リソースおよびリソース・グループが クラスターのメンバーであることを意味します。
- Dynamic relationships は、バインディング・ステップによって発生する一時的な関係です。

リソースの動的検証

リソース定義は、常に作成時に検証されます。 構成変更が発生した場合は、リソースを再度検証して、無効になるのを回避する必要があります。

このタスクについて

リソースの検証は、構成時にリソースを System Automation for Multiplatforms に対して定義する際に実行されます。その際、 問題が発生した場合や新規リソースの定義に失敗した場合、 ユーザーは即時に通知を受けます。

ただし、リソースが定義された後、構成変更が発生した場合はこの限りではありません。System Automation for Multiplatforms は構成変更後に通知を受け、それらの変更を受け入れて対処します。その結果として 1 つ以上のリソースが無効になることがあります。

この事態は、例えば、**mkrsrc** コマンドでリソースを定義する場合、**chrsrc** コマンドでリソースの値を変更する場合、または **rmrsrc** コマンドで定義済みリソースを除去する場合に起こる可能性があります。

そのような構成変更を検証し、リソースの妥当性をユーザーに伝えるために、リソース・クラス IBM.ResourceGroup、IBM.ManagedResource、IBM.Equivalency、および IBM.ManagedRelationship には動的属性 ConfigValidity が含まれています。ConfigValidity には、 リソースが無効である理由を説明するストリングが含まれます。

lsrsrc -Ad コマンドを使用して、 リソースの他の動的属性の値とともに ConfigValidity の値を表示します。

以下の検証が実行されます。

- リソース・グループの AllowedNode 属性が空である
  - ノードが除去されるとき、同値に空のメンバー・リストを含んでしまうことがあります。リソース・グループがその「AllowedNode」属性としてこの空の同値を使用すると、そのリソース・グループは

無効になります。これが発生した場合、リソース・グループの「ConfigValidity」動的属性には「許可されたノード・リストが空です」というストリングが含まれます。

- ネストされたリソース・グループの AllowedNode の共通部分がない
  - 連結されたリソース・グループにおいては、ネストされたすべての内部リソース・グループ とそれらを含むリソース・グループは、少なくとも 1 つのノードを共通して持っている必要があります。 共通するノードが 1 つのみで、そのノードが除去された場合は、すべてのリソース・グループが 無効になります。これが発生した場合、リソース・グループの「ConfigValidity」動的属性には、「連結され、ネストされたリソース・グループに共通ノードがありません」というストリングが 含まれます。
- リソースを稼働させるノードがない
  - リソース・グループにおいて、リソース・グループの AllowedNode とメンバー・リソースの NodeNameList との間で共通する ノードが 1 つのみである場合があります。このノードが 除去されると、リソース・グループは無効になります。これが発生した場合、リソース・グループの「ConfigValidity」動的属性には「リソースを開始するための共通ノードがありません」というストリングが含まれます。
- 関係を満たすノードがない
  - DependsOn 関係において、暗黙的なコロケーションがある場合、ソース・リソースの NodeNameList とターゲット・リソース の NodeNameList は、少なくとも 1 つのノードを共通して持っている必要があります。このノードが除去されると、関係は無効になります。これが発生した場合、管理対象関係の「ConfigValidity」動的属性には「ソースおよびターゲット間に共通ノードがありません」というストリングが含まれます。
- AntiCollocated 関係を満たすことができない - 1
  - リソース・グループ内の 2 つの必須浮動リソースが互いに AntiCollocated 関係を 保持しており、ノードの除去によりリソース・グループの AllowedNode に 1 つのノードのみが残った場合、リソース・グループは無効になります。これが発生した場合、リソース・グループおよび AntiCollocated 管理対象関係の「ConfigValidity」動的属性には、AntiCollocated 関係を満たすことができないとのメッセージが含まれます。
- AntiCollocated 関係を満たすことができない - 2
  - ノードの除去により、2 つの浮動リソースの構成要素が 同じノード上に 1 つのみ残されました。しかしこの 2 つの浮動リソースは AntiCollocated 関係を保持しています。これが発生した場合、AntiCollocated 管理対象関係の「ConfigValidity」動的属性には、AntiCollocated 関係を満たすことができないとのメッセージが含まれます。
- 伝搬される無効性
  - 無効な内部リソース・グループがあると、それを含むすべてのリソース・グループが無効になります。これが 発生した場合、影響を受けるリソース・グループの「ConfigValidity」動的属性には、 「囲みリソース・グループが無効です」というストリングが含まれます。

リソースの開始および停止

以下のコマンド・パターンは、`StartCommand`、`StopCommand`、および `MonitorCommand` のそれぞれの戻りコードに応じた、相互の反応の様子を示します。

アプリケーションの開始

アプリケーションの状況は MonitorCommand によってモニターされます。 例えば、MonitorCommand が戻す RC=1 は、アプリケーションがオンラインであることを 意味します。StartCommand はアプリケーションを始動します。以下のシナリオは、 アプリケーションの開始に関する StartCommand、StopCommand、および MonitorCommand の 対話パターンを示します。

アプリケーションの正常開始

次の図は、アプリケーションの正常開始におけるコマンド対話パターンを 示します。

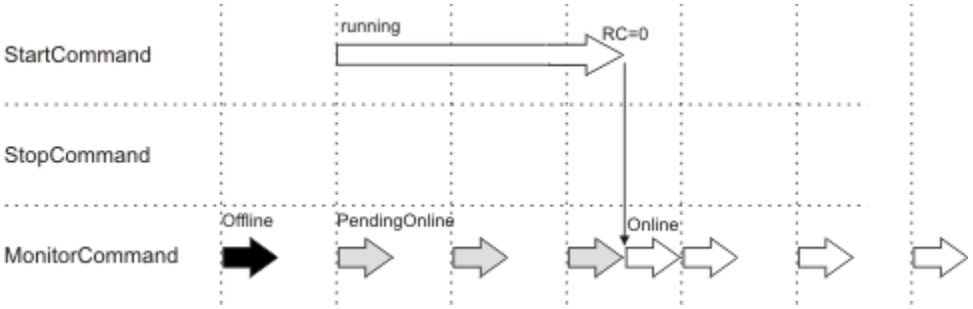


図 *88.* アプリケーションの正常開始

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、アプリケーションの OpState は「オンラインの保留中」です。
   - MonitorCommand が RC=5 (オンラインの保留中) を戻す場合。
   - MonitorCommand が RC=2 (オフライン) を戻す場合、GbResRM リソース・マネージャー は OpState を暗黙で設定します。
3. StartCommand が完了して RC=0 (成功) を戻し、アプリケーションが 正常に開始された場合、MonitorCommand は RC=1 (オンライン) を戻し、 アプリケーションはオンラインです。

注 : StartCommand が戻す RC=0 (成功) は、StartCommand の正常終了 を意味します。MonitorCommand が RC=1 (オンライン) を報告する場合のみ、 アプリケーションはオンラインです。

アプリケーションの開始の異常停止

次の図は、アプリケーションの異常停止におけるコマンド対話パターンを 示します。StartCommand はエラーで終了しました。

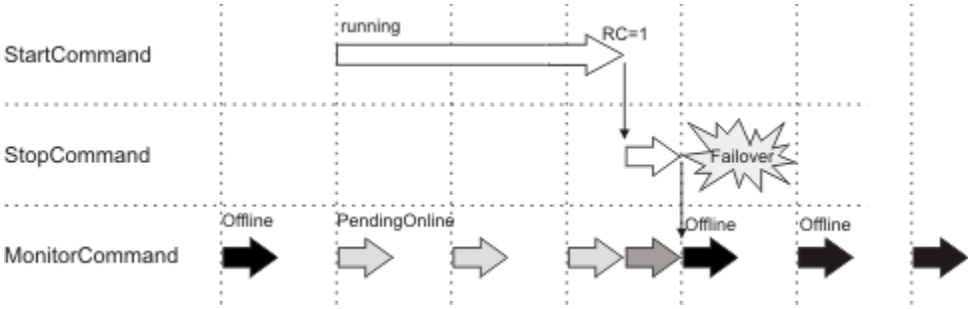


図 *89.* アプリケーションの異常停止

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、 MonitorCommand は RC=5 (オンラインの保留中) を戻し、アプリケーションは 「オンラインの保留中」状態です。
3. StartCommand が RC=1 (失敗) を戻します。アプリケーションがオンライン でなく、StartCommand が失敗で終了したため、MonitorCommand は「オフラインの保留中」を戻します。
4. アプリケーションがオンラインになる前に StartCommand がエラーで終了された 場合は、System Automation for Multiplatforms が StopCommand を実行して、フェイルオーバー手順が実行されます。

注 :

1. StopCommand は StartCommand の異常終了の直後に実行されます。 StopCommand を実行する前に、まずアプリケーションの状況を確認することを 検討してください。

不整合なコマンド状況

次の図は、不整合なコマンド状況に対するコマンド対話パターンを 示します。

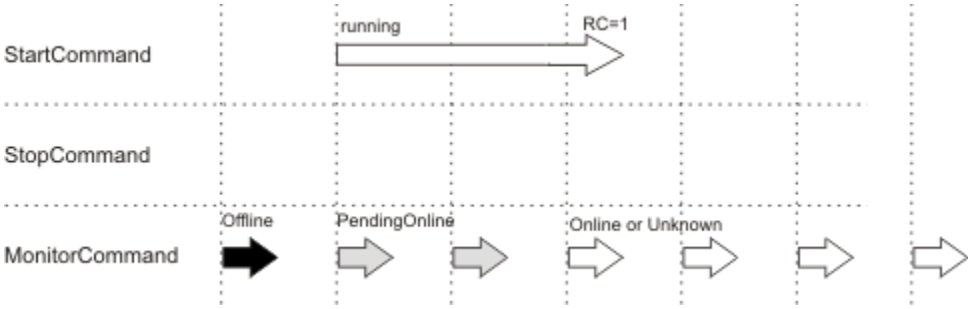


図 *90.* 不整合なコマンド状況

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、 MonitorCommand は RC=5 (オンラインの保留中) を戻し、アプリケーションは 「オンラインの保留中」状態です。
3. StartCommand が完了して RC=1 (失敗) を戻す前に、MonitorCommand が RC=1 (オンライン) または RC=0 (不明) を戻し、アプリケーションがオンラインになります。StartCommand がまだ完了していないという事実は無視され、アプリケーションが正常開始されたかのように全体の状態が扱われます。

注 :

1.「不明」状態を回避するために、MonitorCommandTimeout は慎重に定義してください。

コマンド・タイムアウト

次の図は、StartCommand のタイムアウトに対するコマンド対話パターンを 示します。

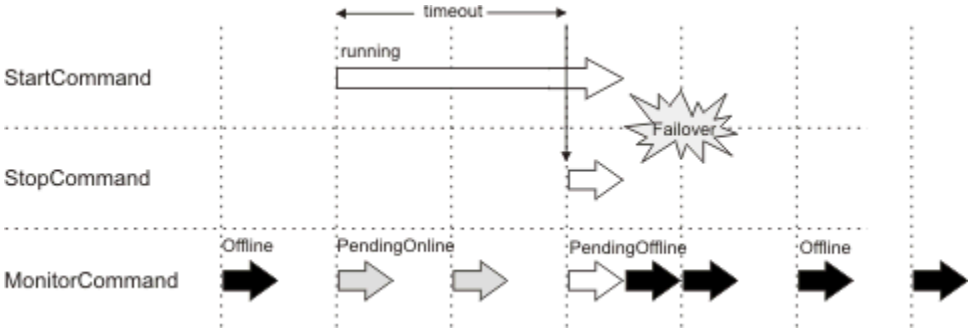


図 *91. StartCommand* のタイムアウト

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、 MonitorCommand は RC=5 (オンラインの保留中) を戻し、アプリケーションは 「オンラインの保留中」状態です。
3. StartCommand が完了する前に、間隔がタイムアウトになりました。System Automation for Multiplatforms は StartCommand を 停止します。StopCommand が実行されて、フェイルオーバーが開始されます。 アプリケーションの状態は、「オンラインの保留中」から「オフラインの保留中」に切り替わり、その後「オフライン」状態に切り替わります。

注 :

1. StopCommand は StartCommand が停止された直後に実行されます。 StopCommand を実行する前に、まずアプリケーションの状況を確認することを 検討してください。

StartCommand の再試行

次の図は、StartCommand の再試行に対するコマンド対話パターンを 示します。

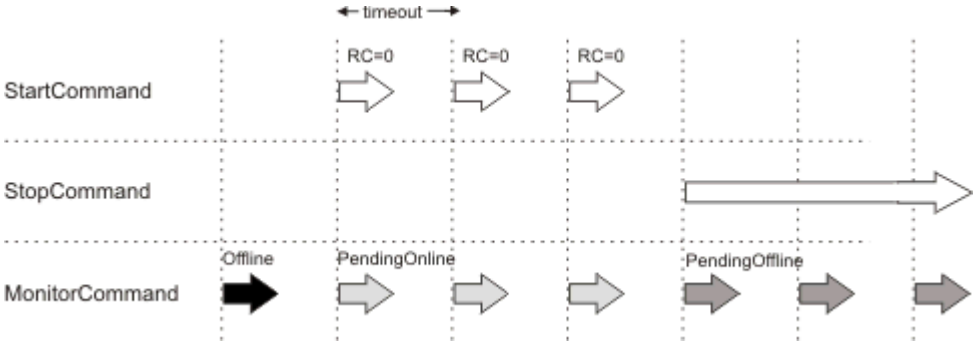


図 *92. StartCommand* の再試行

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、 MonitorCommand は RC=5 (オンラインの保留中) を戻し、アプリケーションは 「オンラインの保留中」状態です。
3. 前の StartCommand が RC=0 (成功) を戻し、アプリケーションがオンラインでない ため、StartCommand が再試行されます。
4. 定義されている数 (RetryCount) の再試行によってアプリケーションがオンライン にならない場合は、StopCommand が実行されてフェイルオーバーが開始されます。
5. アプリケーションがオフラインになる前は MonitorCommand は「オフラインの保留中」状態を戻します。

注 :

1. アプリケーションがオンラインになると RetryCount に 0 が再設定されます。

MonitorCommand の「オフラインに失敗」

次の図は、「オフラインに失敗」状態を戻す MonitorCommand に対する コマンド対話パターンを示します。

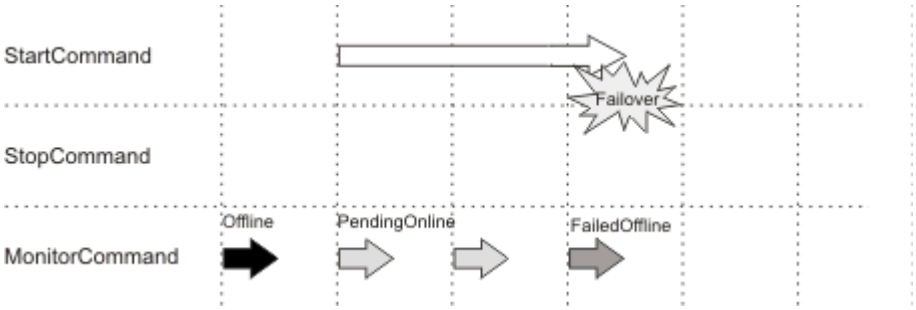


図 *93. MonitorCommand* の「オフラインに失敗」

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=2 (オフライン)、アプリケーションはオフライン状態です。
2. StartCommand が開始されます。StartCommand が実行されている間、 MonitorCommand は RC=5 (オンラインの保留中) を戻し、アプリケーションは 「オンラインの保留中」状態です。
3. StartCommand が正常に完了する前に MonitorCommand が RC=3 (オフラインに失敗) を戻します。
4. System Automation for Multiplatforms は、フェイルオーバー手順を即時に開始します。StopCommand は実行されません。

注 :

1. フェイルオーバーが開始される前に追加のステップが必要な場合は、それらのステップ を MonitorCommand に追加してください。MonitorCommand では、リソースが 実際にオフラインの場合は、3 (オフラインに失敗) のみを戻す必要があります。 そうしないと、リソースまたはリソースの一部がノードで実行されたままに なります。

アプリケーションの停止

StopCommand はアプリケーションを停止します。以下のシナリオは、 アプリケーションの停止に関する StartCommand、StopCommand、および MonitorCommand の 対話パターンを示します。

アプリケーションの正常停止

次の図は、アプリケーションの正常停止におけるコマンド対話パターンを 示します。

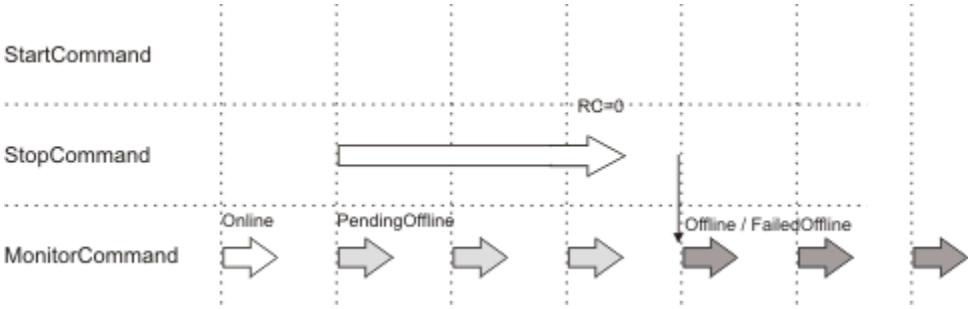


図 *94.* アプリケーションの正常停止

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=1、アプリケーションはオンライン状態です。
2. StopCommand が開始されます。StopCommand が実行されている間、MonitorCommand は RC=6 を戻し、アプリケーションは 「オフラインの保留中」状態です。
3. StopCommand が完了して RC=0 (成功) を戻し、アプリケーションが 正常に停止された場合、MonitorCommand は RC=2 を戻し、 アプリケーションは「オフライン」または「オフラインに失敗」です。

注 :

- MonitorCommand が「オフライン」または「オフラインに失敗」を戻した 場合、System Automation for Multiplatforms は、次のプロセスを開始します。
- MonitorCommand が「オフラインに失敗」を戻した場合は、このノードに対する フェイルオーバー処理は実行できません。

アプリケーションの異常停止

次の図は、アプリケーションの異常停止におけるコマンド対話パターンを 示します。

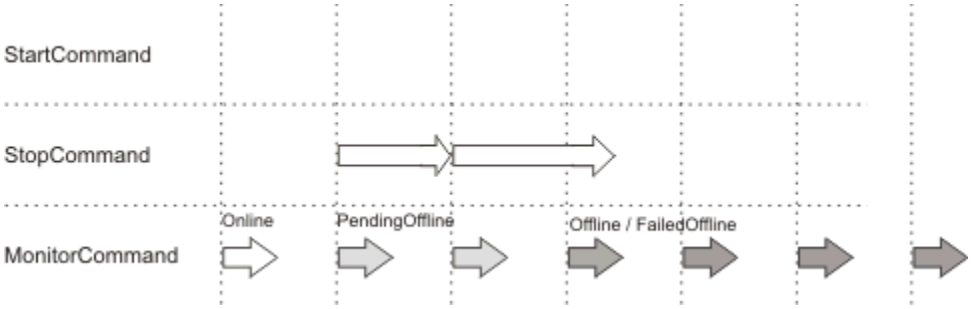


図 *95. StopCommand* が完了されないうちのアプリケーションの異常停止

コマンド・イベントのシーケンス:

1. MonitorCommand: RC=1、アプリケーションはオンライン状態です。
2. StopCommand が開始されます。StopCommand が実行されている間、MonitorCommand は RC=6 を戻し、アプリケーションは 「オフラインの保留中」状態です。
3. StopCommand が完了して RC=0 (成功) を戻し、アプリケーションが 正常に停止された場合、MonitorCommand は RC=2 を戻し、 アプリケーションは「オフライン」または「オフラインに失敗」です。

注 :

- MonitorCommand が「オフライン」または「オフラインに失敗」を戻した 場合、System Automation for Multiplatforms は、次のプロセスを開始します。
- MonitorCommand が「オフラインに失敗」を戻した場合は、このノードに対する フェイルオーバー処理は実行できません。

ボリューム・グループに応じたファイル・システムのマウント

コマンド samwhy を使用したアプリケーション障害の調査

samwhy コマンドは、システム自動化によって制御されるアプリケーションのアプリケーション障害を検出および分析する、使いやすい簡易ツールです。

コマンド samwhy は、アプリケーションの状態をオペレーターが把握するのに役立ち、またシステム自動化がそれに対して行った動作の理由を説明します。samwhy は、イベントの履歴およびシステム自動化の自動化決定の履歴を使用して、理解しやすい可読形式の出力を提供します。 出力では、指定した時間に samwhy によって検出されたアプリケーション障害がリストされ、システム自動化がトリガーした対応アクションのログが記録されます。

使用方法:

```
samwhy [-dhv] [-t hh[:mm[:ss]] | <#>h|m|s] [Resource_class[:Resource_name[:node]]]

Options:
    -d, --detail   Show detailed error information.
    -h, -?         Show brief help information.
    -t TIME        期間を指定します。 Available formats:
                     <#>h|m|s       show last <#> hours|minutes|seconds
                     hh[:mm[:ss]]   show last hours:minutes:seconds
    -v             Show build date and version.

    --help         Show extended help information.
    --nocolor      Do not use colors in the output.
```

以下の段落では、シナリオの例について説明し、対応する samwhy の出力を示します。

セットアップ: 1 つのリソース・グループ (group) と 1 つの IBM.Application リソース (res) が含まれている、2 つのノードで構成されたシステム自動化クラスター。

シナリオ:

- アプリケーション res が nodeA でダウンします。
- システム自動化は、nodeA でアプリケーション res を再始動しようとします。
- nodeA でのアプリケーション res の開始に失敗します。
- アプリケーション res は nodeB に移動され、nodeB で開始されます。

コマンド lssam の対応する出力は、以下のとおりです。

```
nodeA:/tmp # lssam
Online IBM.ResourceGroup:group Control=MemberInProblemState Nominal=Online
        '- Online IBM.Application:res Control=MemberInProblemState
                |- Failed offline IBM.Application:res:nodeA
                '- Online IBM.Application:res:nodeB
```

このシナリオにおける samwhy の出力は、以下のとおりです。

```
nodeA:# ./samwhy
Found 1 error in domain samwhy in the last 24 hours:
-----------------------------------------------------------------------------------------------------
12/16/13 16:01:51 Restart failed for IBM.Application:res on node nodeA. Failover to node nodeB started.
```

詳細な出力は、以下のとおりです。

```
nodeA:# ./samwhy -d
12/16/13 16:01:51 IBM.Application:res failed to Restart in place on node nodeA because the StartCommand
    returned rc:265 TimeOut.
  Restart in place had been triggered by monitor at 16:01:46.630952 returning Offline, while previous
    monitor had returned Online.
  The last monitor for the resource returned Offline on node nodeA.
  Failover of IBM.Application:res from node nodeA to node nodeB started at 12/16/13 16:01:51.741225.
  The last monitor for the resource returned Online on node nodeB.
```

samwhy は、システム自動化クラスターのすべてのノード上のリカバリー・リソース・マネージャーおよびグローバル・リソース・マネージャーによってトレース・ファイルに書き込まれたイベントからデータを収集します。 リカバリー・リソース・マネージャーは、start、stop、move などの要求のトレース項目を書き込みます。 グローバル・リソース・マネージャーは、start、stop、monitor などのリソース・アクションのトレース項目を書き込みます。

samwhy は、すべてのトレース項目を取得してマージし、タイム・スタンプに従ってソートし、障害およびその理由について検査します。samwhy は出力をコンソールに書き込みます。コンソールには、見つかったすべてのアプリケーション障害および各障害の説明が表示されます。また、システム自動化が障害を解決するためにトリガーしたアクションも表示されます。

リソース・グループの移動

他のリソースに影響を与えることなく、クラスター・ノード間で単一のリソース・グループを移動できます。

このタスクについて

個々のリソース・グループを、現在同じノード上で稼働しているすべてのリソースに影響を与えることなく、別のクラスター・ノードに移動できます。これは、例えば、単一または小数のリソース・グループを別のノードに移動するときに、作業負荷とパフォーマンスの目的が既に達成されているような、ロード・バランシングのシナリオに適しています。

リソースの配置を調整するには、**rgreq -o move** コマンドを使用します。

移動範囲

移動範囲とは、最上位リソース・グループのすべてのメンバーです。移動の対象になる 1 つ以上のリソースに依存するリソースは、影響を受ける可能性があります (停止される、開始されるなど)。単一の管理対象リソースに対して移動要求を発行 することはできません。

このタスクについて

最上位リソース・グループの MemberLocation 属性が collocated に設定されている場合は、 **rgreq** コマンドを実行してもノード・リストは表示されません。この場合、すべてのリソースは同一ノードにあり、このノードから移動されます。リソース・グループが 連結されていない場合は、**rgreq** コマンドの -n オプションに 1 つ以上のノードのリストを指定する必要があります。これらのノードからすべてのリソースが移動されます。

移動要求の動作は以下のとおりです。

- 固定リソースのみから成るリソース・グループに対して発行された移動要求は受け入れられません。
- オンラインではないリソース・グループに対して発行された移動要求は拒否されます。

- リソース・グループは単一の移動のみを処理できます。
- 同一リソース・グループの移動を再試行すると拒否されます。
- 異なるリソース・グループに対する複数の移動要求は順に処理されます (1 つの移動要求が処理されている間、他の移動要求はキューに入れられます)。

移動範囲内のリソースに同値に対する Dependency がある場合は、同一の同値に対する Dependency を持つその他のすべてのリソースも移動範囲に追加され、その結果、移動操作時に停止および再始動されることになります。この動作を回避するには、同値のメンバーが同じ場合でも、関連のないリソースに別の同値を作成する必要があります。

別のリソース・グループに対して DependsOn 関係を持つリソース・グループを移動の対象とすると、このリソース・グループは停止されますが、別のリソース・グループに影響はありません。したがって、移動する必要のあるリソース・グループが同じノード上で再始動されるだけです。ただし、以下の制限があります。

- 移動が機能するのは、DependsOn 関係のターゲットであるリソース・グループを移動する場合です。
- DependsOn 関係のソースであるリソース・グループを移動できるようにするには、その優先順位の値を別のリソース・グループより 11 以上大きくする必要があります。

移動要求の処理

移動要求を処理するには、ノード除外アプローチが使用されます。移動要求で指定されたノードで現在実行されている、移動スコープ内のリソース、またはこれらのリソースに依存する、移動スコープ内のリソースのみが停止されます。

このタスクについて

キューに入れられる移動要求の場合は、システムに入力されると即時に、これらのすべての移動要求に対してこのオフライン・フェーズが発生します。後続のオンライン・フェーズのみが順次処理されます。詳しくは、「167 ページの『移動範囲』」を参照してください。

移動のオンライン・フェーズでは、バインド・プログラムがすべてのリソース・グループ・メンバーの新規ノード配置を実行し、リソース・グループが再始動されます。

移動元ノードのリストを受け入れると最上位リソース・グループの必須メンバーを再始動できないことが判明した場合は、リストは無視され、リソースが再始動されます。同様に、稼働していた元のシステムでのみリソースを再始動する場合、これが必須リソースであれば、元のシステムで再始動されます。

移動要求は、移動アクションが実行されると 自動的に除去されます。移動するリソース・グループの MoveStatus 動的属性は、移動の進行状況を示す値を示します。

ただし、移動操作が完了していなくても、リソースが元のノードに戻されない場合は、移動要求のキャンセルが必要になることがあります。移動要求をキャンセルするには、以下のコマンドを使用します。

```
rgreq -o movecancel
```

移動と関係

このタスクについて

最上位リソース・グループのメンバーに対する移動要求の実行に加え、移 動された最上位グループの外側にその他のリソース・グループおよびリソース・グルー プ・メンバー (またはこのいずれか) が存在することがあります。この場合、定義されている関係の制約に基づいて 配置を調整する必要があります。これは以下の関係に適用されます。

- Collocation
- AntiCollocation
- 依存
- DependsOnAny

- StopAfter
- ForcedDownBy

さらに、移動後には Affinity 関係と AntiAffinity 関係が満たされないこ とがあります。

移動とフェイルバック

このタスクについて

フェイルバック・リソースを含むグループに対する移動要求を実行した場合、それらの リソースは、リソースのホーム・ノードに即時にフェイルバックされる可能性があります。フェイルバック が望ましくない場合には、2 つのオプションがあります。

- グループ・メンバーの SelectFromPolicy 属性を order または any に 変更して、ホーム・ノードへのフェイルバック機能を非アクティブ化します。
- *System Automation for Multiplatforms* インストールと構成のガイドの説明に従って、ホーム・ノードを除外します。

ホーム・ノードへの フェイルバック機能について詳しくは、62 ページの『ホーム・ノードへのフェイルバック機能』 を 参照してください。

リソース・グループとリソースのロックおよびアンロック

このセクションのトピックでは、リソース・グループとリソースをロックして、それらを自動化から除外する方法について説明します。

リソース・グループおよび個々のリソース・グループ・メンバーを現行状態で凍結して、自動化されないようにするには、ロック要求を使用します。 ロックされたリソース・グループとリソースの自動化を再開するには、アンロック要求を使用します。ロック要求とアンロック要求はコマンド行から実行します。

- リソース・グループをロックするには、以下のコマンドを使用します。

```
rgreq -o lock
```

- 特定のリソース・グループ・メンバーをロックするには、以下のコマンドを使用します。

```
rgmbrreq -o lock
```

- ロック要求は永続的です。この要求は、リソースの要求リストに無期限に保存されます。 要求リストから除去するには、リソースをアンロックする必要があります。リソースのアンロック時にロック要求がアクティブになっているときは、リストの次のランキングの要求が活動化されます。
- ロックされたリソース・グループをアンロックし、ロック要求を要求リストから除去するには、以下のコマンドを使用します。

```
rgreq -o unlock
```

- ロックされたリソース・グループ・メンバーをアンロックし、ロック要求を要求リストから除去するには、 以下のコマンドを使用します。

```
 rgmbrreq -o unlock
```

ロック要求のスコープ

ロック要求のスコープは、以下の方法で決定できます。

- ロックされたリソースが、ロック要求のスコープ内にある。
- リソース・グループ自体が、ロック要求のスコープ内にある場合は、このグループのすべてのメンバーがそのスコープ内にある。
- StartAfter、DependsOn、または DependsOnAny 関係のターゲットが、ロック要求のスコープ内にある場合は、ソースもそのスコープ内にある。

- StopAfter 関係のソースが、ロック要求のスコープ内にある場合は、ターゲットもそのスコープ内にある。

要求の優先順位の変更

要求を優先順位付けする場合、System Automation for Multiplatforms は、優先順位と要求のソース属性の値を使用します。「ソース」属性と要求の優先順位は、要求の実行依頼方法に従ってデフォルト値に設定されます。 **rgreq** および **rgmbrreq** コマンドを発行するときに、-S オプションを使用してソースを指定し、-p オプションを使用して要求の優先順位を設定することによって、デフォルト値を指定変更できます。要求の優先順位付け方法の詳細については、171 ページの『要求の優先順位付け』を参照してください。

リソース・グループの要求リストまたはリソース・グループ・メンバーに含めることができる各ソースからの要求は 1 つのみであるため、同じソースからのロック要求は相互に置き換わります。 要求リストを表示する場合は、**lsrgreq** コマンドを使用できます。

例のシナリオ

この例のシナリオで、オペレーターが要求を発行およびキャンセルして、リソース・グループの操作状態を変更する方法、ならびに、**lsrgreq** コマンドを使用して、リソースの要求リストを表示し、要求の成否を追跡する方法を説明します。 以下の例で使用されているコマンドの詳細な説明については、「*Tivoli System Automation for Multiplatforms* リファレンス・ガイド 」を参照してください。

リソース・グループ「top-rg」の NominalState は「オンライン」ですが、その OpState は「オフライン」です。

```
lnxcm3x:# lsrg -g top-rg | grep State
Displaying Resource Group information:
For Resource Group "top-rg".

Resource Group 1:
        NominalState         = Online
        OpState              = Offline
        TopGroupNominalState = Online
```

lsrgreq コマンドの出力では、ロック要求は開始できないため、グループはオフラインであることを示しています。

```
lnxcm3x:# lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
For Resource Group "top-rg".

Resource Group 1:
        ResourceGroup = top-rg
        Priority      = High
        Action        = lock
        Source        = Operator
        NodeList      = {}
        ActiveStatus  = Inactive
        Token         = 8f5697eb5f84c0f044995b3d00040a5b
        UserID        =
        MoveStatus    = None
```

グループ「top-rg」をオンラインにする試みで、オペレーターはグループをアンロックします。

```
rgreq -o unlock top-rg
```

ここで「top-rg」の要求リストは、以下のようになります。

```
lnxcm3x:# lsrgreq -L -g top-rg
Displaying Resource Group request information:
All request information
No requests were found.
```

要求の優先順位付け

このセクションのトピックでは、これらの要求がどのように優先順位付けされるかについて説明します。

要求のソースおよびデフォルト優先順位

要求の「ソース」属性が設定されたデフォルト値、および要求のデフォルト優先順位は、要求の実行依頼方法 (実行依頼者)、ならびにそれが出された対象のリソース (「ターゲット」リソース) のタイプによって異なります。 171 ページの表 27 にデフォルト値を示します。

表 27. 開始および停止要求のソースおよび優先順位

実行依頼者	ターゲット・リソース	ソース属性値	デフォルトの優先順位
第 1 レベル自動化モードの System Automation Application Manager オペレーション・コンソール	リソースまたはリソース・グループ	Operator	high
System Automation for Multiplatforms コマンド行インターフェース	リソースまたはリソース・グループ	Operator	低 可能性がある値: 低、高、強制実行
外部スケジューラー (例えば、Tivoli® Workload Scheduler またはクーロン・ジョブ)	リソースまたはリソース・グループ	ExtSched	高 可能性がある値: 低、高、強制実行
System Automation Application Manager エンドツーエンド自動化エンジン	System Automation for Multiplatforms リソースまたはリソース・グループを参照する、エンドツーエンド自動化リソース参照	Automation	high
エンドツーエンド自動化モードの System Automation Application Manager オペレーション・コンソール	System Automation for Multiplatforms リソースまたはリソース・グループを参照する、エンドツーエンド自動化リソース参照	Automation	high
System Automation Application Manager のコマンド行インターフェース	System Automation for Multiplatforms リソースまたはリソース・グループを参照する、エンドツーエンド自動化リソース参照	Automation	high

要求の優先順位付け方法

このタスクについて

リソース・グループに対して実行依頼される要求の優先順位付けは、以下の方法で行われます。

- すべての要求は、リソース・グループの NominalState よりも優先順位が高い。
- 要求の優先順位が異なる場合は、この優先順位によって要求の優先順位ランクを決める。

- 要求の優先順位が同じである場合は、「ソース」属性を評価して、各要求の優先順位ランクを決める。 172 ページの図 96 で、「ソース」属性値による要求の優先順位ランキングへの影響の与え方を説明します。

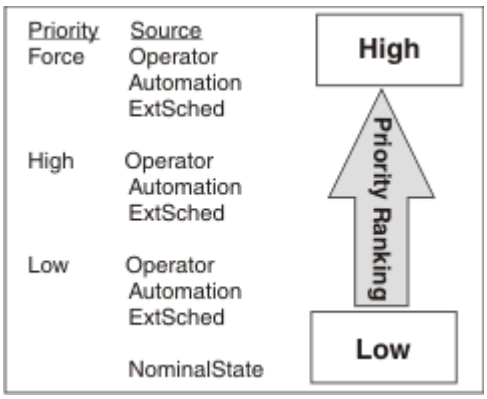


図 96. 要求の優先順位ランキング

172 ページの図 96 は、以下のことを示しています。

– ソース「オペレーター」からの要求がソース「自動化」および「ExtSched」からの要求に優先する。
– ソース「自動化」からの要求がソース「ExtSched」からの要求に優先する。

つまり、ソース「自動化」からの要求は、オプション `-p high` を使用してコマンド行要求を発行し、却下できます。 ソース「オペレーター」からの要求を却下するには、オプション `-p force` を指定する必要があります。

System Automation Application Manager が管理するリソースに対して要求を発行すると、参照される第 1 レベルの自動化リソースに対するその結果の要求は、常にエンドツーエンド自動化マネージャーによって発行されます。このため、優先順位は常に「高」に設定され、そのソース属性値は「自動化」に設定されます。これは、エンドツーエンド自動化リソースに対する元の要求が、エンドツーエンド自動化マネージャーによって発行されたか、オペレーターによって発行されたかに関係なく当てはまります。

IBM Support Assistant の使用

以下は英語のみの対応となります。IBM Support Assistant は、任意のワークステーションにインストールできる、フリーのスタンドアロン・アプリケーションです。IBM Support Assistant を使用することで、製品、サポート、および教育リソースを検索する時間が節約され、問題管理レコード (PMR) または Electronic Tracking Record (ETR) を開く必要がある場合に 情報を収集するために役立ちます。これらのレコードは問題の追跡に使用できます。

次に、ご使用の IBM 製品に対応する製品固有のプラグイン・モジュールをインストール して、このアプリケーションを機能強化できます。Tivoli System Automation for Multiplatforms 用の製品固有プラグインは、以下のリソースを提供します。

- サポート・リンク
- 教育リンク
- 問題管理レポートを送信する機能
- トレース収集機能

IBM Support Assistant および Tivoli System Automation for Multiplatforms プラグインのインストール

IBM Support Assistant V 4.1 をインストールするには、以下のステップを実行します。

- IBM Support Assistant Web サイトにアクセスします。

  ```
  www.ibm.com/software/support/isa/
  ```

- ご使用のプラットフォームに対応するインストール・パッケージをダウンロードします。IBM のユーザー ID (例えば、MySupport または developerWorks® ユーザー ID) およびパスワードを使用してサインインする必要があることに注意してください。IBM ユーザー ID をお持ちでない場合は、登録処理 (無料) を完了することにより入手できます。
- インストール・パッケージを一時ディレクトリーに解凍します。
- インストール・パッケージに含まれている「*Installation and Troubleshooting Guide*」の指示に従って、IBM Support Assistant をインストールします。

Tivoli System Automation for Multiplatforms のプラグインをインストールするには、以下の手順を実行します。

1. IBM Support Assistant アプリケーションを始動します。IBM Support Assistant は、システムに構成されているデフォルトの Web ブラウザーに表示される Web アプリケーションです。
2. IBM Support Assistant 内の「**アップデーター(Updater)**」タブをクリックします。
3. 「**新規製品およびツール (New Products and Tools)**」タブをクリックします。製品ファミリーごとにプラグイン・モジュールがリストされます。
4. 「**Tivoli**」 > 「**Tivoli Tivoli System Automation for Multiplatforms** 」 を選択します。
5. インストールする機能を選択し、「**インストール**」をクリックします。ライセンス情報および使用法の説明を必ずお読みください。
6. IBM Support Assistant を再始動します。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、 サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、 またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を 保有している場合があります。本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムと その他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie New York 12601-5400
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用すること ができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定の プログラム契約の契約条項、IBM プログラムのご使用条件、 またはそれと同等の条項に基づいて、IBM より提供されます。

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、 商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含む すべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、 強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、 改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものでは ありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。 それらの Web サイトは、お客様の責任でご使用ください。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は 表示されない場合があります。

商標

- IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、ibm.com/trademark をご覧ください。
- Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。
- Microsoft、Windows、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の商標または登録商標です。
- Linux は、Linus Torvalds の米国およびその他の国における登録商標です。
- Red Hat およびすべての Red Hat ベースの商標は、Red Hat, Inc. の米国およびその他の国における商標または登録商標です。
- UNIX は The Open Group の米国およびその他の国における登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

C

D

E

F

I

L

M

N

O

P

R

S

V

[特殊文字]

部品番号:
プログラム番号: 5724-M00

(1P) P/N:

SA88-7250-05