Sun GlassFish Communications Server 1.5 高可用性 (HA) 管理ガイド

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

Part No: 820–7388–10
2009 年 1 月

090310@21990

目次

表目次

例目次

はじめに

本書では、融合負荷分散、HTTP 負荷分散、クラスタ化、セッション持続性、およびフェイルオーバーを含む、Communications Server の高可用性機能について説明します。

ここでは、Sun GlassFish™ Communications Server のマニュアルセット全体に関する情報と表記規則について説明しています。

Communications Server のマニュアルセット

Communications Server マニュアルの URL (Uniform Resource Locator) は、`http://docs.sun.com/coll/1343.8` です。Communications Server への導入としては、次の表に示されている順序でマニュアルを参照してください。

表 P–1　Communications Server のマニュアルセットの内容

マニュアル名	説明
『Documentation Center』	タスクや主題ごとに整理された Communications Server のマニュアルのトピック。
『リリースノート』	ソフトウェアとマニュアルに関する最新情報。サポートされているハードウェア、オペレーティングシステム、Java™ Development Kit (JDK™)、およびデータベースドライバの包括的な表ベースの概要を含みます。
『クイックスタートガイド』	Communications Server 製品の使用を開始するための手順。
『Installation Guide』	ソフトウェアとそのコンポーネントのインストール。
『アプリケーション配備ガイド』	アプリケーションおよびアプリケーション コンポーネントの Communications Server への配備。配備記述子に関する情報を含みます。
『開発者ガイド』	Communications Server 上で動作することを目的とし、Java EE コンポーネントおよび API のオープン Java スタンダードモデルに準拠した、Java Platform, Enterprise Edition (Java EE プラットフォーム) アプリケーションの作成と実装。開発者ツール、セキュリティー、デバッグ、ライフサイクルモジュールの作成に関する情報を含みます。
Java EE 5 Tutorial	Java EE 5 プラットフォームテクノロジと API を使用した Java EE アプリケーションの開発。

表 P–1 Communications Server のマニュアルセットの内容 (続き)

マニュアル名	説明
『Java WSIT Tutorial』	Web サービス相互運用性テクノロジ (WSIT) を使用した Web アプリケーションの開発。WSIT テクノロジを使用する方法、時期、および理由と、各テクノロジがサポートする機能およびオプションについて説明します。
『管理ガイド』	設定、監視、セキュリティー、資源管理、および Web サービス管理を含む Communications Server のシステム管理。
『高可用性 (HA) 管理ガイド』	クラスタの設定、ノードエージェントの使用、およびロードバランサの使用。
『Administration Reference』	Communications Server 設定ファイル `domain.xml` の編集。
『パフォーマンスチューニングガイド』	パフォーマンスを向上させるための Communications Server の調整。
『Reference Manual』	Communications Server で使用できるユーティリティーコマンド。マニュアルページのスタイルで記述されています。`asadmin` コマンド行インタフェースも含みます。

関連マニュアル

その他のスタンドアロンの Sun GlassFish サーバー製品については、次のマニュアルを参照してください。

- メッセージキュー documentation (`http://docs.sun.com/coll/1343.4`)
- Identity Server documentation (`http://docs.sun.com/app/docs/prod/ident.mgmt#hic`)
- Directory Server documentation (`http://docs.sun.com/coll/1224.1`)
- Web サーバー documentation (`http://docs.sun.com/coll/1308.3`)

Communications Server とともに提供されるパッケージの Javadoc™ ツールリファレンスの場所は `http://glassfish.dev.java.net/nonav/javaee5/api/index.html` です。さらに、次のリソースが役立つことがあります。

- Java EE 5 Specifications (`http://java.sun.com/javaee/5/javatech.html`)
- Java EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

NetBeans™ 統合開発環境 (IDE) でのエンタープライズアプリケーション開発については、`http://www.netbeans.org/kb/55/index.html` を参照してください。

Communications Server に含まれる Java DB データベースの詳細は、`http://developers.sun.com/javadb/` を参照してください。

GlassFish Samples プロジェクトは、さまざまな Java EE テクノロジの使用方法の例を示すサンプルアプリケーションの集合です。GlassFish Samples は Java EE Software Development Kit (SDK) に付属しています。また、`https://glassfish-samples.dev.java.net/` の GlassFish Samples プロジェクトページからも入手できます。

デフォルトのパスおよびファイル名

次の表は、このマニュアルで使用されているデフォルトのパス名とファイル名について説明したものです。

表 P–2 デフォルトのパスおよびファイル名

プレースホルダ	説明	デフォルト値
as-install	Communications Server のベースインストールディレクトリを表します。	Solaris™ および Linux プラットフォームへのインストールで、ルートユーザーでない場合: *user's-home-directory*/SUNWappserver Solaris および Linux プラットフォームへのインストールで、ルートユーザーである場合: /opt/SUNWappserver Windows のすべてのインストールの場合: *SystemDrive*:\Sun\AppServer
domain-root-dir	すべてのドメインを含むディレクトリを表します。	すべてのインストールの場合: *as-install*/domains/
domain-dir	ドメインのディレクトリを表します。 設定ファイルには、次のように表される *domain-dir* があります。 ${com.sun.aas.instanceRoot}	*domain-root-dir*/*domain-dir*
instance-dir	サーバーインスタンスのディレクトリを表します。	*domain-dir*/*instance-dir*
samples-dir	サンプルアプリケーションを含むディレクトリを表します。	*as-install*/samples
docs-dir	マニュアルを含むディレクトリを表します。	*as-install*/docs

文字の表記ルール

この表は、このマニュアルで使用される文字体裁の種類について説明したものです。

表 P–3 文字の表記ルール

スタイル	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、および画面上のコンピュータ出力	.loginファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% you have mail.
AaBbCc123	ユーザーが入力する文字 (画面上のコンピュータ出力と対照的に表示)	machine_name% **su** パスワード:
AaBbCc123	実際の名前または値に置換されるプレースホルダ	ファイルを削除するコマンドは、 rm ファイル名です。
AaBbCc123	書名、新しい用語、および強調する用語 (オンラインでは、一部の強調項目は太字で表示される)	ユーザーズガイドの第 6 章を参照してください。 キャッシュとは、ローカルに格納されているコピーです。 ファイルを保存しないでください。

記号の表記ルール

この表は、このマニュアルで使用される記号について説明したものです。

表 P–4 記号の表記ルール

記号	説明	例	意味
[ ]	省略可能な引数やコマンドオプションが含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必須コマンドオプションの選択項目が含まれています。	-d {y|n}	-d オプションには、y 引数または n 引数のいずれかを使用する必要があります。
${ }	変数参照を示します。	${com.sun.javaRoot}	com.sun.javaRoot 変数の値を参照します。
-	同時に実行する複数のキーストロークを結び付けます。	Control-A	コントロールキーを押しながら A キーを押します。
+	連続で複数のキーストロークを行います。	Ctrl + A + N	Control キーを押して離してから、次のキーを押します。

表 P–4 記号の表記ルール (続き)

記号	説明	例	意味
→	グラフィカルユーザーインタフェースでのメニュー項目の選択を示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから、「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sun の Web サイトには、次に示す関連情報が示されています。

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

第三者の Web サイト参照

このマニュアル内で参照している第三者の URL は、追加の関連情報を提供します。

注 – このマニュアル内で引用する第三者の Web サイトの可用性について Sun は責任を負いません。こうしたサイトやリソース上の、またはこれらを通じて利用可能な、コンテンツ、広告、製品、その他の素材について、Sun は推奨しているわけではなく、Sun はいかなる責任も負いません。こうしたサイトやリソース上の、またはこれらを経由して利用可能な、コンテンツ、製品、サービスを利用または信頼したことに伴って発生した (あるいは発生したと主張される) いかなる損害や損失についても、Sun は一切の責任を負いません。

このマニュアルに関するコメント

弊社では、マニュアルの改善に努めており、お客様からのコメントおよびご忠告をお受けしております。

コメントを共有するには、http://docs.sun.comを参照し、「フィードバック」をクリックしてください。このオンラインフォームでは、マニュアルのタイトルと Part No. もご記入ください。Part No. は、7 桁か 9 桁の番号で、マニュアルのタイトルページまたは最初のページに記載されています。

◆ ◆ ◆ 第 1 章

Communications Server の高可用性

この章では、クラスタプロファイルおよびエンタープライズプロファイルで利用できる Sun GlassFish Communications Server の高可用性機能について説明します。

この章の内容は、次のとおりです。

高可用性の概要

高可用性アプリケーションおよびサービスは、ハードウェアやソフトウェアの障害には関係なく、機能を継続的に提供します。このようなアプリケーションは、99.999% の時間利用可能であることから、ファイブナインの信頼性を実現していると言われることがあります。

Communications Server では、次の高可用性機能を提供します。

融合ロードバランサ

融合ロードバランサは HTTP/HTTPS および SIP/SIPS 要求の両方を受け入れ、クラスタ内のアプリケーションサーバーインスタンスに転送します。ネットワーク障害のためにインスタンスが失敗して使用不可になるか、または応答しなくなると、ロードバランサは要求を既存の使用可能なマシンにリダイレクトします。ロードバランサはまた、障害が起きたインスタンスが復旧したことを認識し、それに応じて負荷を再配分することもできます。Communications Server は融合ロードバランサを提供します。クラスタ (Communications Server インスタンスの) ま

たはスタンドアロンインスタンスは、専用ロードバランサになります。クラスタ内の各インスタンスは負荷分散にかかわることができます。この場合、クラスタは「自己負荷分散」です。

ロードバランサによって、ワークロードが複数の物理マシンに分散されるため、全体的なシステムスループットが向上します。HTTP および SIP 要求のフェイルオーバーを通して、より高い可用性も提供されます。融合ロードバランサは、システム配備のスケーラビリティーの実現にも役立ちます。

状態を持たない単純なアプリケーションであれば、負荷分散されたクラスタで十分なこともあります。しかし、セッション状態を持ったミッションクリティカルなアプリケーションの場合は、負荷分散されたクラスタを複製セッション持続性とともに使用します。

負荷分散にかかわるサーバーインスタンスとクラスタは、同種の環境を確保する必要があります。

融合ロードバランサの負荷分散およびフェイルオーバーの設定については、第 2 章「融合負荷分散の設定」を参照してください。

高可用性 JMS (Java Message Service)

Java Message Service (JMS) API は、Java EE アプリケーションおよびコンポーネントに対して、メッセージの作成、送信、受信、および読み取りを可能にするメッセージング標準です。この API によって、緩やかに結合され、信頼性が高く、非同期の分散通信が可能となります。Sun Java System Message Queue (MQ) は JMS を実装し、Communications Server と密接に統合されているため、MQ を使用してメッセージ駆動型 Bean (MDB) などの JMS に依存するコンポーネントを作成できます。

接続のプールおよびフェイルオーバーと、MQ クラスタを通じて、JMS の高可用性が実現されます。詳細については、第 7 章「Java Message Service 負荷分散とフェイルオーバー」を参照してください。

接続プールとフェイルオーバー

Communications Server は JMS 接続プールとフェイルオーバーをサポートします。Communications Server は JMS 接続を自動的にプールします。デフォルトでは、Communications Server は、指定されたホストリストから主 MQ ブローカをランダムに選択します。フェイルオーバーが発生すると、MQ は負荷を別のブローカに透過的に転送し、JMS セマンティクスを保持します。

JMS 接続のプールおよびフェイルオーバーの詳細については、113 ページの「接続プールとフェイルオーバー」を参照してください。

MQ クラスタ

MQ Enterprise Edition は、ブローカクラスタと呼ばれる、相互に接続した複数のブローカインスタンスをサポートします。ブローカクラスタによって、クライアント接続はクラスタ内のすべてのブローカに分散されます。クラスタ化することで、水平方向のスケーラビリティーが提供され、可用性が向上します。

MQ クラスタの詳細については、115 ページの「MQ クラスタと Communications Server の併用」を参照してください。

RMI-IIOP 負荷分散とフェイルオーバー

RMI-IIOP 負荷分散では、IIOP クライアント要求が別のサーバーインスタンスまたはネームサーバーに分散されます。目標は、負荷をクラスタ間に均等に拡散して、スケーラビリティーを実現することです。また、IIOP 負荷分散を EJB のクラスタリングおよび可用性と結合すれば、EJB フェイルオーバーも実現されます。

クライアントがオブジェクトに対して JNDI 検索を実行すると、ネームサービスは、原則的に要求を特定のサーバーインスタンスにバインドします。それ以降、そのクライアントからの検索要求はすべて、同じサーバーインスタンスに送信されます。こうして、すべての `EJBHome` オブジェクトは、同じターゲットサーバーにホストされます。また、それ以降に取得された Bean 参照もすべて、同じターゲットホスト上に作成されます。JNDI 検索の実行時に、すべてのクライアントがターゲットサーバーのリストをランダムに選択するため、これにより負荷分散が効果的に実現されます。ターゲットサーバーインスタンスが停止すると、検索または EJB メソッド呼び出しは、別のサーバーインスタンスに処理が引き継がれます。

RMI-IIOP 負荷分散とフェイルオーバーは、透過的に発生します。アプリケーションの配備中に、特別な手順は必要ありません。アプリケーションクライアントが配備される Communications Server インスタンスがクラスタに参加する場合、Communications Server は、クラスタ内で現在アクティブなすべての IIOP 端点を自動的に検出します。ただし、端点の 1 つで障害が発生した場合に備えて、クライアントにはブートストラップ目的で少なくとも 2 つの端点を指定しておくことをお勧めします。

RMI-IIOP 負荷分散およびフェイルオーバーの詳細については、第 8 章「RMI-IIOP 負荷分散とフェイルオーバー」を参照してください。

詳細情報

ハードウェア要件の評価、ネットワーク構成の計画、およびトポロジの選択を含む、高可用性配備の計画については、『Sun GlassFish Enterprise Server 2.1 配備計画ガイド』を参照してください。また、このマニュアルでは、次に示すような概念への高レベルな導入も提供しています。

- ノードエージェント、ドメイン、クラスタなどの Communications Server コンポーネント
- クラスタ内の IIOP 負荷分散
- メッセージキューのフェイルオーバー

高可用性機能を利用するアプリケーションの開発の詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』を参照してください。

高可用性サーバーおよびアプリケーションの調整

高可用性とともに最適なパフォーマンスを得るためにアプリケーションや Communications Server を設定および調整する方法については、docs.sun.com で『パフォーマンスチューニングガイド』を参照してください。このドキュメントでは、次のようなトピックが説明されています。

- 持続性の頻度および持続性のスコープの調整
- ステートフルセッション Bean のチェックポイントの設定
- JDBC 接続プールの設定
- セッションサイズ
- 最適なパフォーマンスを得るためのロードバランサの設定

障害からの回復

Sun Cluster の使用

Sun Cluster では、ドメイン管理サーバー、ノードエージェント、Communications Server インスタンスおよびメッセージキューの自動フェイルオーバーを提供しています。詳細については、docs.sun.com で『Sun Cluster Data Service for Sun Java System Application Server Guide for Solaris OS』を参照してください。

標準の Ethernet 相互接続および Sun Cluster 製品のサブセットを使用します。この機能は、Java ES に含まれています。

手動での復旧

さまざまな方法を使用して、個別のサブコンポーネントを手動で復旧できます。

- 22 ページの「メッセージキューの回復」

ドメイン管理サーバーの回復

ドメイン管理サーバー (DAS) が失われて影響があるのは、管理だけです。たとえ DAS が到達不可能であっても、Communications Server クラスタおよびアプリケーションは、それまでどおり稼働し続けます。

DAS を復旧するには、次のいずれかの方法を使用します。

- `asadmin` でバックアップコマンドを定期的に実行して、定期的なスナップショットを作成します。ハードウェア障害後に、新しいマシンに同じネットワーク ID で Application Server をインストールし、以前に作成されたバックアップから `asadmin` で復元を実行します。詳細については、23 ページの「ドメイン管理サーバーの再作成」を参照してください。
- 共有されている強固なファイルシステム (NFS など) 上で、ドメインのインストールと設定をします。主 DAS マシンで問題が発見されると、2 番目のマシンが同じ IP アドレスで稼働し、手動による介入、またはユーザーが指定した自動化によって引き継がれます。Sun クラスタでは、DAS フォールトトレラントを実現する場合と同様の手法を使用します。
- Communications Server のインストールおよびドメインルートディレクトリを zip 形式で圧縮します。それを新しいマシンで復元し、同じネットワーク ID を割り当てます。ファイルベースのインストールを使用している場合は、これがもっとも簡単な方法である可能性があります。
- DAS バックアップから復元します。AS8.1 UR2 パッチ 4 の指示を参照してください。

ノードエージェントおよびサーバーインスタンスの回復

ノードエージェントおよびサーバーインスタンスを回復する方法は 2 つあります。

バックアップの **zip** ファイルを保持する。ノードエージェントおよびサーバーインスタンスをバックアップする明確なコマンドはありません。ノードエージェントディレクトリの内容を持つ zip ファイルを単に作成します。障害発生後に、同じホスト名と IP アドレスを持つ新しいマシンで保存済みのバックアップを解凍します。インストールディレクトリの場所、OS なども同じものを使用します。ファイルベースのインストール、パッケージベースのインストール、または復元されたバックアップイメージがマシン上に存在する必要があります。

手動での回復。同じ IP アドレスを持つ新しいホストを使用する必要があります。

1. Communications Server をノードエージェントとともにマシンにインストールします。
2. ノードエージェントを再作成します。サーバーインスタンスを作成する必要はありません。

3. 同期によって、設定およびデータが DAS からコピーされ更新されます。

メッセージキューの回復

メッセージキュー (MQ) の設定およびリソースは、DAS に格納され、インスタンスに同期できます。その他のデータおよび設定情報は、すべて MQ ディレクトリ (通常は `/var/imq` の下) にあるため、必要に応じてこれらのディレクトリをバックアップおよび復元します。新しいマシンには、あらかじめ MQ インストールが含まれている必要があります。マシンを復元するときは、それまでどおり MQ ブローカが起動していることを必ず確認してください。

Netbackup の使用

注 – この手順は、Sun QA によって確認されていません。

各マシンのイメージを保存するには、Veritas Netbackup を使用します。BPIP の場合は、Web サーバーおよび Application Server とともに 4 つのマシンをバックアップします。

復元されたそれぞれのマシンについて、元のマシンと同じ設定 (同じホスト名、IP アドレスなど) を使用します。

Communications Server のようなファイルベース製品では、関連するディレクトリだけをバックアップおよび復元します。ただし、Web サーバーイメージのようなパッケージベースのインストールでは、マシン全体をバックアップおよび復元する必要があります。パッケージは、Solaris パッケージデータベースにインストールされます。そのため、ディレクトリだけをバックアップし、続いて新しいシステムに復元すると、パッケージデータベースには情報のない、「展開された」Web サーバーになります。このため、今後のパッチ適用やアップグレードで問題が発生する可能性があります。

Solaris パッケージデータベースを手動でコピーおよび復元しないでください。代替方法は、Web サーバーなどのコンポーネントのインストール後にマシンのイメージをバックアップすることです。これをベースライン tar ファイルと呼びます。Web サーバーに変更を加えた場合は、たとえば `/opt/SUNWwbsvr` の下にあるこれらのディレクトリをバックアップします。復元するには、ベースライン tar ファイルから開始し、次に変更した Web サーバーディレクトリを上書きコピーします。同様に、MQ でもこの手順を使用できます (BPIP のパッケージベースのインストール)。元のマシンをアップグレードまたはパッチ適用する場合、新しいベースライン tar ファイルを作成する必要があります。

DAS のあるマシンがダウンすると、復元するまで利用できない時間が発生します。

DAS は、中央リポジトリです。サーバーインスタンスを復元して再起動すると、DAS にある情報とのみ同期されます。そのため、すべての変更は asadmin または管理コンソール を使用して実行する必要があります。

ドメイン管理サーバーの再作成

ドメイン管理サーバー (DAS) をホストしているマシンで障害が発生した場合、以前に DAS をバックアップしたことがあれば DAS を再作成できます。DAS の作業コピーを再作成するには、次の環境が必要です。

- 元の DAS を含む、1 番目のマシン (machine1)。
- アプリケーションを実行していてクライアントに提供しているサーバーインスタンスとクラスタを含む、2 番目のマシン (machine2)。1 番目のマシン上の DAS を使用して、クラスタが設定されています。
- 1 番目のマシンがクラッシュしたときには、3 番目のバックアップマシン (machine3) に DAS が再作成される必要があります。

注 – 1 番目のマシンからの DAS のバックアップを保持している必要があります。現在のドメインをバックアップするには、`asadmin backup-domain` を使用します。

▼ DAS を移行する

ドメイン管理サーバーを 1 台目のマシン (machine1) から 3 台目のマシン (machine3) に移行するには、次の手順が必要です。

1 **1 台目のマシンと同様に、Communications Server を 3 台目のマシンにインストールします。**

これは、DAS が 3 番目のマシンに正しく復元され、パスの競合が起きないようにするために必要です。

a. **コマンド行 (対話型) モードを使用して、Communications Server 管理パッケージをインストールします。**

対話型コマンド行モードを有効にするには、次のように `console` オプションを使用してインストールプログラムを呼び出します。

`./`*`bundle-filename`* `-console`

コマンド行インタフェースを使用してインストールするには、ルート権限が必要です。

b. **デフォルトのドメインをインストールするオプションを選択解除します。**

バックアップされたドメインの復元は、同じアーキテクチャーおよびまったく同じインストールパスを持つ 2 台のマシンでのみサポートされます (すなわち両方のマシンが同じ *as-install* と *domain-root-dir* を使用する)。

2 バックアップの **ZIP** ファイルを **1** 番目のマシンから **3** 番目のマシンの *domain-root-dir* にコピーします。

ファイルを FTP 転送することもできます。

3 **ZIP** ファイルを **3** 番目のマシンに復元します。

```
asadmin restore-domain --filename domain-root-dir/sjsas_backup_v00001.zip
--clienthostname machine3 domain1
```

注 – `---clienthostname` オプションを指定することによって、`domain.xml` ファイル内の `jmx-connector` 要素の `client-hostname` プロパティーを変更する必要性を回避します。

任意のドメインをバックアップできます。ただし、ドメインの再作成時に、ドメイン名を元のドメインと同じ名前にしてください。

4 **3** 番目のマシンの *domain-root-dir*`/domain1/generated/tmp` ディレクトリのアクセス権を変更し、**1** 番目のマシンの同じディレクトリのアクセス権と一致させます。

このディレクトリのデフォルトのアクセス権は次のとおりです。 `drwx------` (または 700)。

次に例を示します。

```
chmod 700 domain-root-dir/domain1/generated/tmp
```

この例では、`domain1` をバックアップしていることとします。ドメインを別の名前でバックアップしている場合は、上記の `domain1` をバックアップしているドメインの名前で置き換えるようにしてください。

5 **3** 番目のマシン上の *domain-root-dir*`/domain1/config/domain.xml` ファイルで、`jms-service` 要素の `host` 属性の値を更新します。

この属性の元の設定は次のとおりです。

```
<jms-service... host=machine1.../>
```

この属性の設定を次のように変更します。

```
<jms-service... host=machine3.../>
```

6 **machine3** で復元されたドメインを起動します。

```
asadmin start-domain --user admin-user --password admin-password domain1
```

DAS は稼働中のすべてのノードエージェントと通信し、DAS と通信するための情報をノードエージェントに提供します。ノードエージェントはこの情報を使用して DAS と通信します。

7 **DAS** の再起動時に稼働していないすべてのノードエージェントについて、**machine2** で *as-install*/nodeagents/ *nodeagent*/agent/config/das.properties の agent.das.host プロパティー値を変更します。

この手順は、DAS の再起動時に稼働しているノードエージェントについては不要です。

8 **machine2** でノードエージェントを再起動します。

注 – asadmin start-instance コマンドを使用してクラスタインスタンスを起動し、復元されたドメインと同期できるようにします。

◆ ◆ ◆ 第 2 章

融合負荷分散の設定

この章では、融合負荷分散について説明します。ここで説明する内容は次のとおりです。

この章では、Communications Server に含まれている融合ロードバランサを使用して、HTTP、HTTPS、SIP、および SIPS メッセージの負荷を分散する方法について説明します。

もう 1 つの 負荷分散オプションは、Communications Server で Sun Secure Application Switch を使用し、ハードウェアベースの負荷分散ソリューションを構築するというものです。このソリューションの設定については、Clustering and Securing Web Applications: A Tutorial (`http://developers.sun.com/prodtech/appserver/reference/techart/load-balancing.html`) を参照してください。

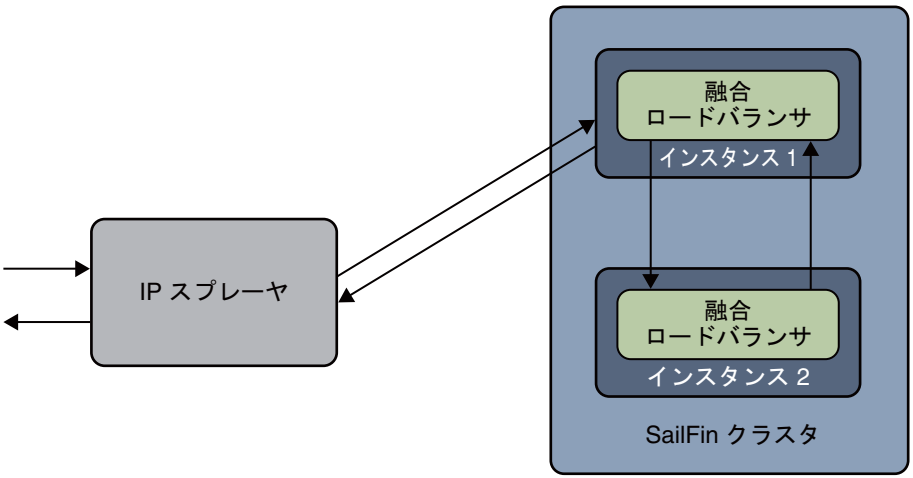
融合ロードバランサの動作

ロードバランサの目的は、スタンドアロンまたはクラスタ化された複数のインスタンスの間でワークロードを均等に分散させ、それにより、システムの全体的なスループットを向上させることです。

融合ロードバランサにより、Java EE アプリケーションサーバーに配備されるサービスの高可用性を実現できます。負荷分散処理の間、それまでサービスを提供していたインスタンスが稼働していないか、または正常な状態でないために要求を処理できないことが検出された場合、融合ロードバランサはセッション要求を同じクラスタの別のサーバーインスタンスにフェイルオーバーします。

注 – ロードバランサは、8K バイトを超える URI や URL を処理しません。

次の図は、ロードバランサの動作を表しています。



1. IP スプレーヤがクライアント要求を受信します。

   注 – IP スプレーヤはクラスタ内のすべてのインスタンスに要求を均等に分散させるもので、ハードウェア IP スプレーヤが使用されることもあります。IP スプレーヤなどのネットワーク要素は、融合ロードバランサの前面に出て、トランスポートレベルでクラスタのトラフィックを分散させます。

2. IP スプレーヤはクラスタ内の任意の SailFin インスタンスを選択して、そのインスタンスに要求を転送します。この例および図では、要求はインスタンス 1 に転送されています。
3. インスタンス 1 の融合ロードバランサはインスタンス (この場合はインスタンス 2) を選択して、要求を処理します。

   要求を転送するためのサーバーインスタンスの選択は、設定されている29 ページの「融合負荷分散アルゴリズム」に基づいて行われます。このキーは、スティッキネスを維持するためのヘッダーまたはパラメータとして要求に追加されます。
4. この例では、インスタンス 1 の融合ロードバランサは要求をインスタンス 2 に転送します。融合ロードバランサがこの要求を処理するためにインスタンス 1 を選択した場合、ステップ 5 は省略されます。

5. インスタンス 2 の融合ロードバランサは要求を受信し、要求がすでに別のインスタンスからプロキシされていることを検出します。さらなる処理がない場合、インスタンス 2 は要求をコンテナに受け渡して、処理できるようにします。インスタンス 2 の融合ロードバランサは応答をインスタンス 1 の融合ロードバランサに戻します。
6. インスタンス 1 の融合ロードバランサは応答をクライアントに戻します。
7. セッションが確立されると、応答にスティッキ情報が設定され、後続の要求にはこのスティッキ情報が保持されます。クライアントからの後続の要求には、ヘッダー/パラメータにスティッキ情報が含まれます。インスタンス 1 はこの要求を受信した場合でも、スティッキ情報を検出して、要求をインスタンス 2 の融合ロードバランサに転送します。

   HTTP および HTTPS のスティッキネスを保つために、ロードバランサはクッキーまたは (ブラウザがクッキーをサポートしていない場合) URL リライティングを使用します。SIP/SIPS セッションの場合、ロードバランサは `BEKey` および `BERoute` などのパラメータを使用します。たとえば、融合ロードバランサは `BERoute` パラメータを送信要求の一部として VIA ヘッダーに貼り付けます。

融合負荷分散アルゴリズム

ロードバランサは次のアルゴリズムのいずれかを自動的に使用します。

- ラウンドロビンアルゴリズム — ロードバランサは新規メッセージを処理するためのインスタンスをラウンドロビン方式で選択します。
- コンシステントハッシュ - ロードバランサは新規メッセージを処理するためのインスタンスを、要求から抽出したハッシュキーに基づいて選択します。ハッシュキーは、(提供されている場合) DCR ファイルで指定されたルールを使用して抽出されます。DCR ファイルが提供されていない場合、ハッシュキーはデフォルトのヘッダーを使用して抽出されます。

  DCR ファイル `data-centric-rules.xml` は、融合アプリケーションまたは純粋な SIP アプリケーションからの HTTP/HTTPS および SIP/SIPS メッセージの両方に対してコンシステントハッシュを適用するためのルールを規定します。このファイルが指定されている場合、このファイル内の命令はデフォルトのヘッダーを使用してハッシュキーを抽出するメカニズムよりも優先されます。DCR ファイルが提供されていない場合、SIP および HTTP メッセージが同じセッションの一部であっても、異なるインスタンスによって処理されることがあります。融合アプリケーションを配備する場合、必ず DCR ファイルを提供してください。デフォルトのヘッダーを使用した場合、SIP メッセージの正しい負荷分散を実現できない場合もあります。したがって、純粋な SIP アプリケーションの場合でも、DCR ファイルを提供することをお勧めします。このファイルの詳細については、35 ページの「設定の編集」 および 39 ページの「データ中心ルールファイル」 を参照してください。

Web アプリケーション向けの融合負荷分散アルゴリズム

純粋な Web アプリケーションに属する HTTP および HTTPS メッセージに対して、融合ロードバランサはデフォルトでは「スティッキラウンドロビン」アルゴリズムを使用します。新規要求がロードバランサに送信されると、単純なラウンドロビン方式に基づいてアプリケーションサーバーインスタンスに転送されます。要求がセッションベースのアプリケーション向けである場合、セッションが作成されることがあります。このような場合、応答にスティッキ情報が含まれ、後続のメッセージで戻されます。同じセッションベースのアプリケーション向けの同一クライアントから送信された後続のメッセージは、割り当てメッセージまたはスティッキメッセージと見なされ、ロードバランサによって同じインスタンス (そのインスタンスが健全と判定された場合) にルーティングされます。スティッキ (sticky: 粘着性の) ラウンドロビンという名前が付いているのはそのような理由からです。セッションベースでないアプリケーションへの要求や、セッションベースのアプリケーションに対する最初の要求は新規要求です。

SIP アプリケーション向けの融合負荷分散アルゴリズム

純粋な SIP アプリケーションに属する SIP および SIPS メッセージに対して、融合ロードバランサはデフォルトでは「コンシステントハッシュ」アルゴリズムを使用します。DCR ファイルの中に SIP/SIPS 要求と一致するルールがある場合、ハッシュキーはそのルールを使用して抽出されます。DCR ファイルの中に SIP または SIPS 要求と一致するルールまたは命令がない場合、ハッシュキーは要求の `from-tag,call-id` パラメータを使用して生成されます。

融合アプリケーション向けの融合負荷分散アルゴリズム

融合ロードバランサは、融合アプリケーションからの HTTP/HTTPS および SIP/SIPS メッセージに対して適切なアルゴリズムを次のように適用します。

- DCR ファイルが指定されていない場合、HTTP/HTTPS メッセージには「スティッキラウンドロビン」アルゴリズムを適用し、SIP/SIPS メッセージには「コンシステントハッシュ」アルゴリズムを適用します。
- DCR ファイルが指定されている場合、HTTP/HTTPS メッセージおよび SIP/SIPS メッセージの両方に「コンシステントハッシュ」アルゴリズムを適用します。

  融合アプリケーションに属する HTTP および HTTPS メッセージに対しては、DCR ファイルの中に HTTP/HTTPS 要求と一致するルールがある場合、そのルールを使用してハッシュキーが抽出されます。DCP ファイルの中に HTTP/HTTPS 要求と一致するルールまたは命令がない場合、HTTP 要求のポートおよびリモートホストを使用してハッシュキーが抽出されます。

  SIP および SIPS メッセージに対しては、DCR ファイルの中に SIP/SIPS 要求と一致するルールがある場合、ハッシュキーはそのルールを使用して抽出されます。DCR ファイルの中に SIP または SIPS 要求と一致するルールまたは命令がない場合、ハッシュキーは要求の `from-tag,call-id` パラメータを使用して生成されます。

融合ロードバランサの配備

ロードバランサは、目標や環境に応じて、以下の節で説明している各種の方法で設定できます。

- 31 ページの「自己負荷分散クラスタの使用」
- 31 ページの「専用負荷分散サーバーインスタンスの使用」

自己負荷分散クラスタの使用

開発環境または本稼働環境では、1 つのクラスタに含まれるすべてのサーバーインスタンスが、専用の負荷分散インスタンスをまったく使用することなく、要求のリダイレクトおよび処理の両方にかかわるように指定できます。 これは、ターゲットと LB ターゲットが同じクラスタである「自己負荷分散」クラスタです。

フロントエンドハードウェア IP スプレーヤは、クラスタ内のすべてのインスタンスに要求を均等に分散させます。ハードウェア IP スプレーヤを使用しない場合、要求はクラスタ内のどのサーバーインスタンスにも転送できます。そのインスタンスの融合ロードバランサコンポーネントにより、要求は確実にクラスタ全体に分散されます。ただし、そのインスタンスはシングルポイント障害です。ハードウェア IP スプレーヤがあると、高可用性を実現できます。

専用負荷分散サーバーインスタンスの使用

開発環境では、1 つ以上のスタンドアロンサーバーインスタンスを、クラスタへの要求のリダイレクト、または要求を処理するその他のスタンドアロンインスタンスへの要求のリダイレクトを実行する専用ロードバランサとして指定できます。これらの専用ロードバランサは、ロードバランサの「ターゲット」と呼ばれます。

要求を処理するクラスタまたはインスタンスはロードバランサの「*LB* ターゲット」と呼ばれます。特定のロードバランサの LB ターゲットにはクラスタまたはスタンドアロンインスタンスを使用できますが、クラスタとインスタンスを混ぜることはできません。

- クラスタ化されたサーバーインスタンスを LB ターゲットとして使用 - ロードバランサを配備する最も一般的な方法は、サーバーインスタンスのクラスタを LB ターゲットとして使用する方法です。デフォルトでは、クラスタ内のすべてのインスタンスは同じ設定を持ち、同じアプリケーションが配備されています。ロードバランサは、サーバーインスタンスの間でワークロードを分散させ、正常でないインスタンスから正常なインスタンスへのフェイルオーバーを要求します。複数のクラスタがある場合、要求はクラスタ間で負荷分散されますが、要求のフェイルオーバーは単一クラスタ内のインスタンス間でのみ行われます。
- 複数のスタンドアロンインスタンスを LB ターゲットとして使用 - 複数のスタンドアロンインスタンスを LB ターゲットとして使用し、これらの LB ターゲット間で要求を負荷分散およびフェイルオーバーするようにロードバランサを設定することもできます。ただし、この設定では、それぞれのスタンドアロンインスタンス

に同種の環境が確保され、同じアプリケーションが配備されていることを手動で確認する必要があります。クラスタでは自動的に同種の環境が維持されるため、ほとんどの状況では、クラスタの使用がより適切で、より容易な方法です。

1 つのクラスタ内のすべてのインスタンスは、相互の IP アドレスに対して同様のビューが必要です。たとえば、インスタンス 1 およびインスタンス 2 という 2 つのインスタンスを持つクラスタについて考えてみます。インスタンス 1 がインスタンス 2 の IP アドレスを調べたときに、a.b.c.d を取得したとします。インスタンス 2 が自らの IP アドレスを調べたときにも、a.b.c.d (インスタンス 1 から見た場合と同じ IP アドレス) として認識される必要があります。ユーザーのマシンにあるホストファイルが正しく設定されていることを確認してください。

注 – 専用ロードバランサは本稼働環境ではサポートされません。

融合負荷分散の設定

この節では、ロードバランサを設定する方法について説明します。次の項目が含まれています。

- 32 ページの「融合負荷分散を設定するための前提条件」
- 32 ページの「融合負荷分散を設定するための手順」

融合負荷分散を設定するための前提条件

ロードバランサを設定する前に、次の手順を実行する必要があります。

- クラスタプロファイルを使用して、負荷分散にかかわる Communications Server クラスタまたはサーバーインスタンスを作成します。詳細は、第 3 章「Communications Server でのクラスタの設定」を参照してください。
- グループ管理サービス (GMS) が有効になっていることを確認します。デフォルトでは有効になっています。詳細は、52 ページの「グループ管理サービス」を参照してください。
- 該当するクラスタまたはサーバーインスタンスのノードエージェントを作成します。詳細は、第 4 章「ノードエージェントの設定」を参照してください。
- これらのクラスタまたはインスタンスに対してアプリケーションを配備します。詳細は、『Sun GlassFish Communications Server 1.5 Application Deployment Guide』を参照してください。

融合負荷分散を設定するための手順

ユーザーの環境で負荷分散を設定するには、管理コンソールまたは `asadmin` コマンドを使用します。以降の節では、さらに詳しい情報を示します。

▼ 管理コンソールを使用して融合負荷分散を設定する

1 ロードバランサを作成します。

左側の区画で「融合ロードバランサ」をクリックし、「新規」をクリックします。「新しい融合ロードバランサ」ページで、ロードバランサ名を入力し、ロードバランサの役割を果たすターゲットクラスタまたはインスタンスも選択します。

任意で、次の設定を指定できます。

- 設定ファイル名 — 融合ロードバランサの設定ファイルの名前を指定します。 設定ファイルのデフォルトのパスおよび名前は、*domain-dir*/config/*cluster-config*/converged-loadbalancer.xml です。
- 変更を自動的に適用 — 設定変更をターゲットサーバーインスタンスに自動的に適用するかどうかを指定します。 デフォルトでは、この設定はオフになっています。この設定を true にして、融合ロードバランサの設定ファイルを生成します。
- 自己負荷分散 — ターゲットクラスタを自己負荷分散にするかどうかを指定します。デフォルトでは、この設定はオンになっています。本稼働環境では、自己負荷分散ターゲットクラスタのみがサポートされます。

2 自己負荷分散でないロードバランサの場合、ロードバランサが管理できるように、クラスタまたはスタンドアロンサーバーインスタンスに参照を追加します。

左側の区画で「融合ロードバランサ」ノードをクリックし、ノードの下に表示されるリストから目的のロードバランサをクリックします。「融合ロードバランサの LB ターゲット」タブを開いて、「LB ターゲットの管理」をクリックし、「LB ターゲットの管理」ページで必要な LB ターゲットを選択します。

クラスタまたはスタンドアロンインスタンスを LB ターゲットとして選択できます。クラスタとスタンドアロンインスタンスの組み合わせを LB ターゲットとして選択することはできません。

注 – このステップは、本稼働環境ではサポートされません。

---

3 ロードバランサを作成したときにクラスタがすでに起動している場合、ロードバランサを起動するには、クラスタを再起動する必要があります。

参照　詳細については、管理コンソールのオンラインヘルプを参照してください。

▼ asadmin コマンドを使用して融合負荷分散を設定する

1 つの asadmin コマンド、create-converged-lb を使用すると、次のステップを実行できます。

1 ロードバランサ設定を作成します。

asadmin create-converged-lb-config コマンドを使用します。

2 ロードバランサが管理するクラスタまたはスタンドアロンサーバーインスタンスへの参照を追加します。

`asadmin create-converged-lb-ref` コマンドを使用します。

注 – クラスタ配備の設定および定義の繰り返し処理を実行するときは、`--autocommit` を false に設定することをお勧めします。デフォルトで、このオプションは false に設定されています。これは主に中間融合ロードバランサファイルの生成を防止することが目的です。この設定がないと、`domain.xml` の変更のたびに中間ファイルが生成され、融合ロードバランサ自体の設定の表示に影響を及ぼします。クラスタ配備定義が展開の安定段階に達した時点で、`--autocommit` を true に設定します。

例 2–1 融合ロードバランサの作成

次の一連の `asadmin` コマンドでは、クラスタ、ノードエージェント、および自己負荷分散融合ロードバランサを設定します。

```
asadmin> create-cluster cluster1
    Command create-cluster executed successfully.
asadmin>create-node-agent --user admin --passwordfile pass.txt
--host host1 nodeagent1
    Command create-node-agent executed successfully.
asadmin>create-node-agent --user admin --passwordfile pass.txt
--host host1 nodeagent2
    Command create-node-agent executed successfully.
asadmin>create-instance --user admin --passwordfile pass.txt --nodeagent nodeagent1
--cluster cluster1 cluster1_instance1
   Command create-instance executed successfully.
asadmin> create-instance --user admin --passwordfile pass.txt
--nodeagent nodeagent2 --cluster cluster1 cluster1_instance2
   Command create-instance executed successfully.
asadmin> create-converged-lb --user admin --passwordfile pass.txt
--configfile clb.xml --autocommit=true --lbenableallinstances=true
--target cluster1 clb-1
   Command create-converged-lb executed successfully.
asadmin> start-node-agent nodeagent1
   Command start-node-agent started successfully.
asadmin> start-node-agent nodeagent2
   Command start-node-agent started successfully.
asadmin> start-cluster cluster1
 cluster1_instance1 is running, does not require restart
 cluster1_instance2 is running, does not require restart
 Command start-cluster executed successfully.
```

ロードバランサを作成したときにクラスタがすでに起動している場合、ロードバランサを起動するには、クラスタを再起動する必要があります。

参照　これらの asadmin コマンドの詳細については、『Sun GlassFish Communications Server 1.5 Reference Manual』を参照してください。

融合ロードバランサの設定

次の節では、ロードバランサ設定を変更および使用する方法についてさらに詳しく説明します。

管理コンソールでのロードバランサ設定の編集

融合ロードバランサの作成後、次の節の説明に従うと、管理コンソールで融合ロードバランサの設定を編集できます。

設定の編集

管理コンソールでロードバランサを設定するには、左側の区画で「融合ロードバランサ」ノードをクリックし、ノードの下に表示されるリストから目的のロードバランサをクリックします。「設定」タブを開きます。「融合ロードバランサ設定を編集」ページが表示されます。

次の表では、ロードバランサ設定の設定について説明します。

表 2–1　ロードバランサ設定の設定

設定	説明
ポリシータイプ	負荷分散アルゴリズムを、HTTP ポリシーおよび SIP ポリシーまたは DCR ファイルのどちらで決定するか指定します。
HTTP ポリシー	HTTP 要求のルーティングに使用されるポリシーを指定します。唯一の許可値は round-robin です。この場合、ロードバランサはクラスタのサーバーインスタンスを指定の順序で循環します。

表 2–1 ロードバランサ設定の設定 (続き)

設定	説明
SIP ポリシー	SIP 要求のルーティングに使用されるポリシーを指定します。ハッシュキーの取得のためにコンシステントハッシュポリシーを適用するパラメータを指定します。これには、ハッシュ対象のパラメータ名の単一値またはコンマ区切り値を使用できます。複数のパラメータを指定する場合、コンシステントハッシュを適用するためにパラメータの連結値が使用されます。デフォルトは `from-tag,call-id` です。
DCR ファイルのアップロード	HTTP および SIP 要求のデータ中心ルールを格納する DCR ファイルを指定します。デフォルトでは、このファイルは指定されていません。このファイルが指定されている場合、デフォルトのパスおよび名前は *domain-dir*`/`*cluster*`/config/data-centric-rules.xml` です。データ中心ルールファイルを指定する融合ロードバランサ設定がクラスタまたはスタンドアロンサーバーインスタンスをまったく参照しない場合、デフォルトのパスおよび名前は *domain-dir*`/config/data-centric-rules.xml`です。 このファイルが指定されている場合、このファイルの命令は「HTTP ポリシー」および「SIP ポリシー」設定よりも優先されます。このファイルの詳細については、39 ページの「データ中心ルールファイル」を参照してください。 HTTP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーはリモートホストおよびポートを使用して生成されます。SIP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーは `from-tag,call-id` を使用して生成されます。
プロパティー	プロパティーの名前および値を設定できます。

ロードバランサの詳細編集

融合ロードバランサの作成後、融合ロードバランサ設定を変更できます。

管理コンソールでロードバランサの詳細を編集するには、左側の区画で「融合ロードバランサ」ノードをクリックし、ノードの下に表示されるリストから目的のロードバランサをクリックします。「ターゲット」タブを開きます。「Edit Load Balancer Details」を選択します。「Edit Load Balancer Details」ページが表示されます。

次の表では、ロードバランサの詳細設定について説明します。

表 2–2 ロードバランサの設定

設定	説明
変更を自動的に適用	設定変更をターゲットサーバーインスタンスに自動的に適用するかどうかを指定します。

表 2–2 ロードバランサの設定 (続き)

設定	説明
設定ファイル名	融合ロードバランサの設定ファイルの名前を指定します。設定ファイルのデフォルトのパスおよび名前は *domain-dir*/config /*cluster-config* です。この場合、*cluster-config* は、クラスタの設定に固有の設定リポジトリディレクトリです。デフォルトでは、設定ファイルの名前は *clb-name*_CLB_CONFIG.xml です。
要求プールサイズ	融合ロードバランサのプロキシによって作成およびプールされた要求オブジェクトの数を指定します。
再試行の送信	データ送信に失敗したときにリモートインスタンスに関してプロキシが行う再試行の回数を指定します。
読み取りタイムアウト	プロキシがソケットチャンネルでクライアントからのデータを待つ時間 (単位：ミリ秒) を指定します。

自己負荷分散の編集 (ロードバランサターゲットの詳細)

ロードバランサの作成後、ロードバランサターゲットクラスタを自己負荷分散にするかどうかを変更できます。

管理コンソールでロードバランサターゲットの詳細を編集するには、左側の区画で「融合ロードバランサ」ノードをクリックし、ノードの下に表示されるリストから目的のロードバランサをクリックします。「LB ターゲット」タブを開きます。「LB ターゲットの詳細を編集」を選択します。「LB ターゲットの詳細を編集」ページが表示されます。このページでは、「自己負荷分散」だけが編集可能な設定です。この設定は有効または無効にすることができます。

注 – 自己負荷分散を無効にしたロードバランサは、本稼働環境ではサポートされません。

ロードバランサ設定の編集

ロードバランサの作成後、`asadmin set` コマンドを使用すると、ロードバランサ設定を編集できます。

融合ロードバランサ設定を編集するには、次のコマンドを使用します。

```
asadmin set config-name-config.availability-service.converged-load-balancer.setting
```

次に例を示します。

```
asadmin set c1-config.availability-service.converged-load-balancer.config-file=myclb.xml

asadmin set c1-config.availability-service.converged-load-balancer.auto-commit=true
```

```
asadmin set c1-config.availability-service.converged-load-balancer.converged-lb-config-name=myclb-config
```

融合ロードバランサプロキシ設定を編集するには、次のコマンドを使用します。

```
asadmin set config-name-config.availability-service.converged-load-balancer.proxy.setting
```

次に例を示します。

```
asadmin set c1-config.availability-service.converged-load-balancer.proxy.request-pool-size=50
asadmin set c1-config.availability-service.converged-load-balancer.proxy.send-retry-count=3
asadmin set c1-config.availability-service.converged-load-balancer.proxy.read-time-out=1500
```

融合ロードバランサプロキシ設定ポリシーの設定を編集するには、次のコマンドを使用します。

```
asadmin set domain.converged-lb-config.clb-config.converged-lb-policy.setting
```

次に例を示します。

```
asadmin set domain.converged-lb-configs.myclb-config.converged-lb-policy.http=round-robin

asadmin set domain.converged-lb-configs.myclb-config.converged-lb-policy.sip=from-tag,call-id
```

クラスタの `self-loadbalance` 設定を編集するには、次のコマンドを使用します。

```
asadmin set domain.converged-lb-config.clbcfg.converged-lb-cluster-ref.cluster.self-loadbalance=setting
```

次に例を示します。

```
asadmin set domain.converged-lb-configs.myclb-config.converged-lb-cluster-ref.clust1.self-loadbalance=true
```

注 – 自己負荷分散を無効にしたロードバランサは、本稼働環境ではサポートされません。

`asadmin set` コマンドの詳細については、『Sun GlassFish Communications Server 1.5 Reference Manual』を参照してください。

サーバーインスタンスの負荷分散の有効化または無効化

サーバーインスタンスを停止する前に、要求が別のインスタンスにフェイルオーバーされるように、そのインスタンスの負荷分散を無効にする必要があります。サーバーインスタンスまたはクラスタの負荷分散を無効にするには、`asadmin disable-converged-lb-server` コマンドを使用します。

次に例を示します。

```
asadmin disable-converged-lb-server --user admin --passwordfile pass.txt cluster1
```

asadmin enable-converged-lb-server コマンドを使用すると、サーバーインスタンスまたはクラスタの負荷分散を有効にすることができます。デフォルトでは、新規のインスタンスまたはクラスタを作成したとき、lb-enabled オプションは false に設定されます。ユーザーはインスタンスまたはクラスタの負荷分散を明示的に有効にする必要があります。

次に例を示します。

```
asadmin enable-converged-lb-server --user admin --passwordfile pass.txt cluster1
```

asadmin disable-converged-lb-server コマンドおよび asadmin enable-converged-lb-server の詳細については、『Sun GlassFish Communications Server 1.5 Reference Manual』を参照してください。

融合ロードバランサのログメッセージレベルの変更

融合ロードバランサメッセージのデフォルトのログレベルは、INFO に設定されています。この設定を変更するには、asadmin set コマンドを使用して javax.enterprise.system.container.clb プロパティーを変更します。たとえば、次のようにすると、c1-config のログレベルを FINE に変更できます。

```
asadmin set c1-config.log-service.module-log-levels.property.clb=FINE
```

```
asadmin set c1-config.log-service.module-log-levels.property.sip=FINE
```

データ中心ルールファイル

データ中心ルールファイルのデフォルトのパスおよび名前は、*domain-dir*/*cluster*/config/ data-centric-rules.xmlです。データ中心ルールファイルを指定する融合ロードバランサ設定がクラスタまたはスタンドアロンサーバーインスタンスをまったく参照しない場合、デフォルトのパスおよび名前は *domain-dir*/config/data-centric-rules.xmlです。ファイル形式を検証する DTD ファイルは、*as-install*/lib/dtds/ sun-data-centric-rule_1_0.dtd です。

データ中心ルールファイルが指定されている場合、コンシステントハッシュアルゴリズムにより、要求の転送先のサーバーインスタンスが特定されます。サーバーインスタンスの選択は、ハッシュキーに基づいて行われます。キーは、データ中心ルールに従って、受信される SIP および HTTP 要求から抽出されます。これらの

ルールは、LB ターゲットに配備されたすべてのアプリケーションに適用され、操作中に変更できます。そのような変更は、新規の初期要求のみに影響を及ぼします。

データ中心ルール言語は、要求をサーブレットにマッピングするためのトリガー言語に似ています。ルール言語は、SIP および HTTP キー抽出をサポートする変数および条件から構成されます。サーブレットマッピング言語と対照的に、データ中心ルールファイルは、non-null (true) または null (false) のいずれかの値を返す必要があります。

個々の受信される SIP および HTTP 要求は、現在のルールセットと合わされます。一致する最初のルールがキー抽出に使用されます。条件が non-null 値を返すまで、ルールは順次評価されます。OR 式の場合、最初の non-null sub-result が返されます。AND 式の場合、最後の sub-result が返されます。一致するルールがない場合、キーは null に設定されます。

HTTP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーはリモートホストおよびポートを使用して生成されます。SIP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーは `from-tag,call-id` を使用して生成されます。

次はデータ中心ルールファイルの例です。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE user-centric-rules PUBLIC "-//Sun Microsystems Inc.//DTD Sailfin 1.0//EN"
"http://www.sun.com/software/appserver/dtds/sun-data-centric-rule_1_0.dtd">
<user-centric-rules>
  <sip-rules>
    <if>
      <session-case>
        <equal>ORIGINATING</equal>
        <if>
          <header name="P-Asserted-Identity"
              return="request.P-Asserted-Identity.uri.resolve.user">
            <exist/>
          </header>
          <if>
            <header name="P-Asserted-Identity"
                return="request.from.uri.resolve.user">
              <notexist/>
            </header>
            <if>
              <header name="P-Asserted-Identity"
                  return="request.to.uri.resolve.user">
                <notexist/>
              </header>
            </if>
          </if>
```

```
        </if>
      </session-case>
      <else return="request.uri.resolve.user" />
    </if>
  </sip-rules>
  <http-rules>
    <or>
      <request-uri return="match.resolve.user">
        <match>/users/([^/]+)</match>
      </request-uri>
      <and>
        <request-uri>
          <match>^/css/</match>
        </request-uri>
        <or>
          <request-uri parameter="referredBy"
              return="parameter.requestUri.uri.resolve.user">
            <exist/>
          </request-uri>
          <request-uri parameter="referredBy"
              return="parameter.from.uri.resolve.user">
            <notexist/>
          </request-uri>
        </or>
      </and>
    </or>
  </http-rules>
</data-centric-rules>
```

次の節では、`data-centric-rules.xml` ファイルの要素について説明します。

トップレベル要素

トップレベル要素は SIP/SIPS および HTTP/HTTPS 要求のルールを決定します。

user-centric-rules

これは `data-centric-rules.xml` ファイルのトップレベルまたはルート要素です。

サブ要素

次のサブ要素のいずれかまたは両方を指定できます: 41 ページの「sip-rules」、42 ページの「http-rules」

sip-rules

SIP および SIPS 要求のデータ中心ルールを決定します。

上位要素

41 ページの「user-centric-rules」

サブ要素

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

http-rules

HTTP および HTTPS 要求のデータ中心ルールを決定します。

上位要素

41 ページの「user-centric-rules」

サブ要素

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

演算子要素

演算子要素は、複数のルール間での決定を指定します。 `or`、`and`、および `if` 要素は再帰的です。これらの要素の分岐およびレベルは、各分岐の最も低いレベルに44 ページの「条件要素」の 1 つが含まれている限り、何個でも指定できます。

or

含まれている条件の少なくとも 1 つが non-null 値と評価されるときに限り、true と評価されます。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

and

含まれている条件のすべてが non-null 値と評価されるときに限り、true と評価されます。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

if

1 つの条件を含みます。条件が non-null 値と評価された場合、評価は `if` 分岐で続行されます。条件が null と評価された場合、評価はオプションの `else` 分岐で続行されます。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

44 ページの「else」 サブ要素はオプションです。指定する場合には、最後に置く必要があります。

else

親の `if` 要素の条件が null と評価された場合に、代替文を指定します。

上位要素

43 ページの「if」

属性

次の属性が必須です。大文字と小文字は区別されます。

`return` 戻り値として評価される変数を指定します。47 ページの「変数」を参照してください。

条件要素

条件要素は、ルール決定の基になるデータの種類を指定します。すべての属性値は大文字と小文字が区別されます。

header

要求ヘッダーに基づいてルールを指定します。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素が厳密に 1 つ必要です。

46 ページの「exist」、47 ページの「notexist」

属性

次の属性が必要です。

`name` 要求ヘッダーの名前を指定します。

`return` 戻り値として評価される変数を指定します。47 ページの「変数」を参照してください。

request-uri

要求 URI とそのパラメータに基づいてルールを指定します。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素が厳密に 1 つ必要です。

46 ページの「exist」、47 ページの「notexist」、47 ページの「match」

属性

次の属性は、サブ要素が `exist` または `notexist` である場合は必須で、サブ要素が `match` である場合はオプションです。

`parameter` 要求 URI パラメータを指定します。

`return` 戻り値として評価される変数を指定します。47 ページの「変数」を参照してください。

session-case

SIP セッションの `call` パラメータに基づいてルールを指定します。IMS/3GPP (Internet Protocol Multimedia Subsystem Third Generation Partnership Project: インターネットプロトコルマルチメディアサブシステム / 第 3 世代移動体通信システムの標準化プロジェクト) 環境でのみ関連性があります。詳細は、47 ページの「equal」サブ要素の説明を参照してください。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

47 ページの「equal」 サブ要素は必須で、先頭に置く必要があります。

次のサブ要素はすべてオプションです。使用できる数および順序に制限はありません。

42 ページの「or」、43 ページの「and」、43 ページの「if」、44 ページの「header」、45 ページの「request-uri」、45 ページの「session-case」、46 ページの「cookie」

cookie

HTTP クッキーに基づいてルールを指定します。

上位要素

41 ページの「sip-rules」、42 ページの「http-rules」、42 ページの「or」、43 ページの「and」、43 ページの「if」

サブ要素

次のサブ要素が厳密に 1 つ必要です。

46 ページの「exist」、47 ページの「notexist」

属性

次の属性が必要です。

`name` クッキーの名前を指定します。

`return` 戻り値として評価される変数を指定します。47 ページの「変数」を参照してください。

条件型要素

条件型要素は要求データをルールによって予測されているデータと比較します。すべての比較で、大文字と小文字は区別されます。

exist

要求に存在する `header`、 `request-uri`、または `cookie` として変数が評価されるように指定します。47 ページの「変数」を参照してください。

上位要素

44 ページの「header」、45 ページの「request-uri」、46 ページの「cookie」

notexist

要求に存在しない header、 request-uri、または cookie として変数が評価されるように指定します。47 ページの「変数」を参照してください。

上位要素

44 ページの「header」、45 ページの「request-uri」、46 ページの「cookie」

equal

呼び出しのどの部分 (呼び出しの発信側または終端側) が処理されるかを指定します。 IMS/3GPP 環境でのみ関連性があります。指定できる値は次のとおりです。

- EXTERNAL — SIP 要求にルートヘッダーがない、またはルートヘッダーに call パラメータがないことを指定します。
- ORIGINATING — ルートヘッダーに call=orig が含まれることを指定します。
- TERMINATING — ルートヘッダーに call=term_registered が含まれることを指定します。
- TERMINATING_UNREGISTERED — ルートヘッダーに call=term_unregistered が含まれることを指定します。

上位要素

45 ページの「session-case」

match

request-uri が一致する必要のある正規表現を指定します。戻り値は、正規表現の前方参照を行うグループの一致です。正規表現の詳細については、http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html を参照してください。

match 要素では、オプションの接頭辞を戻り値に付加できます。

上位要素

45 ページの「request-uri」

変数

データ中心ファイルの name、parameter、および return 属性と 46 ページの「exist」および 47 ページの「notexist」 要素では、変数が使用されます。 文字列 resolve を含

む変数には、TEL URI の ENUM 検索を実行することで取得された値が含まれます。一部の変数には、置換可能なテキストが含まれます。たとえば、`request.`*header* 変数では、*header* は SIP または HTTP ヘッダーの名前に置き換えられます。SIP および HTTP 要求を一致させるための構文は多少異なります。

次の SIP 変数がサポートされます。

```
request.uri
request.uri.scheme
request.uri.user
request.uri.host
request.uri.port
request.method

request.uri.resolve
request.uri.resolve.user
request.uri.resolve.host

request.header
request.header.uri
request.header.uri.scheme
request.header.uri.user
request.header.uri.host
request.header.uri.port
request.header.uri.display-name

request.header.uri.resolve
request.header.uri.resolve.user
request.header.uri.resolve.host

request.header.match
request.header.match.resolve.user

match
match.resolve.user
```

次の HTTP 変数がサポートされます。

```
request.header
request.header.uri
request.header.uri.user
request.header.uri.host
request.header.uri.resolve
request.header.uri.resolve.user
request.header.uri.resolve.host

parameter.parameter
```

```
parameter.parameter.uri
parameter.parameter.uri.user
parameter.parameter.uri.host
parameter.parameter.uri.resolve
parameter.parameter.uri.resolve.user
parameter.parameter.uri.resolve.host

match
match.resolve.user

cookie.cookie-name
```

HTTP 変数 `parameter.` *parameter*`.uri.resolve.user` の解決は複雑です。この変数は HTTP 要求のパラメータ値と一致します。この値は 1 つの `name-addr` か、またはそのシーケンス (コンマ区切り) です。`name-addr` 要素は、使用可能なユーザー中心ハッシュキーが見つかるまで解決されます。分解の順序は次のとおりです。

1. `name-addr` に `user=phone` パラメータが含まれている場合、TEL URL として解決されます。それ以外の場合、URI のユーザー部が抽出されます。したがって、ENUM で解決できない電話番号エンティティーを指定している場合、または SIP URI にユーザー部がない場合、SIP URI の分解は失敗することがあります。
2. すべての SIP URI が考察された後、2 回目の試行が行われ、TEL URL は左から右に読み取られます。使用可能なユーザー中心キーが見つかった時点で、評価はただちに停止します。
3. 分解がすべての試行で失敗した場合、ユーザー中心キーは見つかりません。HTTP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーはリモートホストおよびポートを使用して生成されます。SIP 要求が DCR ファイルルールにまったく一致しない場合、ハッシュキーは `from-tag,call-id` を使用して生成されます。

たとえば、変数が `parameter.from.uri.resolve.user` で HTTP 要求が `GET ...?...&from=...&...HTTP/1.1`である場合、結果は次の表の値に基づきます。実際には、例示されている文字の一部は、URL エンコード処理の必要がある場合があります (例: < が `%3C` のように表示されることがある)。

表 2–3 `from` パラメータ値の例

`from` パラメータの値	ユーザー中心キー
`<sip:server.xx.yy>`	なし
`<sip:alice@server.xx.yy>`	`alice`
`<tel:+1-333-555>,<sip:+1-22-22@server.xx.yy;user=phone>`	ENUM から

ロードバランサと Enterprise Server および Communications Server

融合ロードバランサは、Communications Server ディストリビューションの一部です。HTTP 負荷分散プラグインは、Sun GlassFish Enterprise Server with HADB bundle にバンドルされます。HTTP 負荷分散プラグインは、`http://glassfish.dev.java.net` から別途ダウンロードすることもできます。

HTTP ロードバランサおよび融合ロードバランサ

純粋な Web アプリケーションエンタープライズ配備の場合、HTTP 負荷分散プラグインを Sun GlassFish Enterprise Server 2.1 とともに使用します。Sun GlassFish Communication Server with the Converged Load Balancer は、SIP および融合アプリケーションのエンタープライズ配備を対象にしています。

◆ ◆ ◆ 第 3 章

Communications Server でのクラスタの設定

この章では、Communications Server クラスタの使用法について説明します。次の節で構成されています。

クラスタの概要

クラスタは、同じアプリケーション、リソース、および設定情報を共有するサーバーインスタンスの集まりに名前を付けたものです。異なるマシン上のサーバーインスタンスを 1 つの論理クラスタにまとめ、それらのインスタンスを 1 つの単位として管理できます。マルチマシンクラスタのライフサイクルは、DAS を使用して容易に制御できます。

インスタンスはクラスタにグループ化できます。アプリケーションを単一配備のクラスタのすべてのインスタンスに分散できます。クラスタは動的です。インスタンスが追加または削除されると、変更は自動的に処理されます。

クラスタにより、水平方向のスケーラビリティー、負荷分散、およびフェイルオーバー保護が使用可能になります。定義により、クラスタ内のすべてのインスタンスに対してリソースとアプリケーションの設定は同じになります。あるサーバーインスタンスまたはクラスタ内のあるマシンに障害が起きると、ロードバランサは障害を検出し、障害の起きたインスタンスからクラスタ内のほかのインスタンスにトラフィックをリダイレクトし、ユーザーセッションの状態を回復します。クラスタ内のすべてのインスタンス上には同一のアプリケーションとリソースがあるため、インスタンスはクラスタ内のほかのどのインスタンスにも処理を継続させることができます。

クラスタインスタンスはリングトポロジの形式に構成されています。リング内の各メンバーは、リング内の次のメンバー (レプリカパートナー) にインメモリー状態

データを送信し、前のメンバーから状態データを受信します。いずれかのメンバーで状態データが更新されると、リング全体でレプリケートされます。リングトポロジでメンバーに障害が発生すると、リングは切断されます。グループ管理サービス (GMS) で、メンバーの障害を認識できます。その場合、レプリケーションフレームワークはクラスタのトポロジを作り直して、メンバーにその変更を通知します。メンバーはそのレプリカパートナーが消えたことを認識すると、稼働中のメンバーから新しいパートナーを選択します。

グループ管理サービス

グループ管理サービス (GMS) は、クラスタ内のインスタンスに対して有効になっている基盤コンポーネントです。GMS を有効にすると、クラスタインスタンスで障害が発生した場合にクラスタおよびドメイン管理サーバー (DAS) が障害を認識し、障害発生時に必要な処理を実行することができます。Communications Server の多くの機能は、GMS に依存します。たとえば、GMS は IIOP フェイルオーバー、インメモリーレプリケーション、トランザクションサービス、およびタイマーサービスの各機能によって使用されます。

クラスタ内の複数のサーバーインスタンスが異なるマシンに配置されている場合は、それらのマシンが同じサブネット上にあることを確認してください。

注 – GMS 機能は開発者プロファイルでは利用できません。クラスタプロファイルおよびエンタープライズプロファイルでは、デフォルトで GMS が有効です。

GMS は Shoal フレームワークの中心的なサービスです。Shoal の詳細については、Project Shoal ホームページ (`https://shoal.dev.java.net/`)を参照してください。

▼ クラスタに対して **GMS** を有効または無効にする

1. ツリーコンポーネントで、「クラスタ」を選択します。
2. クラスタの名前をクリックします。
3. 「一般情報」で、必要に応じて「ハートビート有効」チェックボックスが選択または選択解除されていることを確認します。
   - **GMS** を有効にするには、「ハートビート有効」チェックボックスが選択されていることを確認します。
   - **GMS** を無効にするには、「ハートビート有効」チェックボックスが選択解除されていることを確認します。

4. **GMS** を有効にしており、**GMS** で使用するデフォルトのポートおよび **IP** アドレスを変更する必要がある場合は、これらの設定値を変更します。

5. 「保存」をクリックします。

GMS の設定

使用環境に合わせて GMS を設定します。これは、GMS が障害を確認する頻度を決定する設定値を変更することによって行います。たとえば、障害検出試行の間のタイムアウト、障害が疑われるメンバーの再試行の回数、または、クラスタのメンバーを確認するときのタイムアウトを変更できます。

次の例では、get コマンドを使用して、*cluster-config-name* に関連付けられているすべてのプロパティーを取得します。

```
asadmin get cluster2-config.group-management-service.*

cluster2-config.group-management-service.fd-protocol-max-tries = 3
cluster2-config.group-management-service.fd-protocol-timeout-in-millis = 2000

cluster2-config.group-management-service.merge-protocol-max-interval-in-millis
= 10000

cluster2-config.group-management-service.merge-protocol-min-interval-in-millis
= 5000

cluster2-config.group-management-service.ping-protocol-timeout-in-millis = 5000

cluster2-config.group-management-service.vs-protocol-timeout-in-millis = 1500
```

▼ 管理コンソールを使用して GMS 設定を設定する

1. 管理コンソールで、**Communications Server** ノードに移動します。

2. 「設定」 –> *cluster_name* -config —> 「グループ管理サービス」の順にクリックします。

例 3–1 asadmin get および set コマンドによる GMS 設定の変更

管理コンソールの代わりに、asadmin get および set コマンドを使用できます。

```
asadmin> list  cluster2-config.*
cluster2-config.admin-service
cluster2-config.admin-service.das-config
cluster2-config.admin-service.jmx-connector.system
```

```
cluster2-config.admin-service.jmx-connector.system.ssl
cluster2-config.availability-service
cluster2-config.availability-service.jms-availability
cluster2-config.availability-service.sip-container-availability
cluster2-config.diagnostic-service
cluster2-config.ejb-container
cluster2-config.ejb-container-availability
cluster2-config.ejb-container.ejb-timer-service
...
...
...
...
cluster2-config.web-container-availability

asadmin> get  cluster2-config.group-management-service.*
cluster2-config.group-management-service.fd-protocol-max-tries = 3
cluster2-config.group-management-service.fd-protocol-timeout-in-millis = 2000
cluster2-config.group-management-service.merge-protocol-max-interval-in-millis = 10000
cluster2-config.group-management-service.merge-protocol-min-interval-in-millis = 5000
cluster2-config.group-management-service.ping-protocol-timeout-in-millis = 5000
cluster2-config.group-management-service.vs-protocol-timeout-in-millis = 1500

asadmin>set  cluster2-config.group-management-service.fd-protocol-max-tries=4
 cluster2-config.group-management-service.fd-protocol-max-tries = 4

asadmin> get  cluster2-config.group-management-service.*
 cluster2-config.group-management-service.fd-protocol-max-tries = 4
cluster2-config.group-management-service.fd-protocol-timeout-in-millis = 2000
cluster2-config.group-management-service.merge-protocol-max-interval-in-millis = 10000
cluster2-config.group-management-service.merge-protocol-min-interval-in-millis = 5000
cluster2-config.group-management-service.ping-protocol-timeout-in-millis = 5000
cluster2-config.group-management-service.vs-protocol-timeout-in-millis = 1500
```

ロードバランサを作成したときにクラスタがすでに起動している場合、ロードバランサを起動するには、クラスタを再起動する必要があります。

クラスタの操作

- 61 ページの「EJB タイマーを移行する」

▼ クラスタを作成する

1. ツリーコンポーネントで、「クラスタ」ノードを選択します。
2. 「クラスタ」ページで、「新規」をクリックします。
   「クラスタの作成」ページが表示されます。または「新しいクラスタ」ページが表示されます。
3. 「名前」フィールドで、クラスタの名前を入力します。
   この名前は、次の条件を満たす必要があります。
   - 大文字と小文字、数字、下線、ハイフン、およびピリオド (.) だけで構成される
   - すべてのノードエージェント名、サーバーインスタンス名、クラスタ名、および設定名の間で一意である
   - 「`domain`」ではない
4. 「設定」フィールドで、ドロップダウンリストから設定を選択します。
   - 共用設定を使用しないクラスタを作成するには、`default-config` を選択します。
     「選択している設定のコピーを作成します」ラジオボタンを選択済みにしておきます。デフォルト設定のコピーは、*cluster_name*`-config` という名前になります。
   - 共用設定を使用するクラスタを作成するには、ドロップダウンリストから設定を選択します。
     「選択している設定を参照します」ラジオボタンを選択して、指定した既存の共用設定を使用するクラスタを作成します。
5. オプションとして、サーバーインスタンスを追加できます。
   クラスタ作成後にサーバーインスタンスを追加することも可能です。

   サーバーインスタンスは複数のマシンに常駐できます。すべてのサーバーインスタンスは、DAS と通信可能なノードエージェントに関連付ける必要があります。クラスタのサーバーインスタンスを作成する前に、まず 1 つまたは複数のノードエージェントまたはノードエージェントのプレースホルダを作成します。75 ページの「ノードエージェントのプレースホルダを作成する」を参照してください。

   サーバーインスタンスを作成するには、次のようにします。

   a. 「作成するサーバーインスタンス」セクションで、「追加」をクリックします。

b. 「インスタンス名」フィールドにインスタンスの名前を入力します。

c. 「ノードエージェント」ドロップダウンリストからノードエージェントを選択します。

6 「了解」をクリックします。

7 表示される「クラスタを正常に作成」ページで「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-cluster`

参照
- 57 ページの「クラスタを設定する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」
- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタ用のリソースを設定する」
- 60 ページの「クラスタを削除する」

クラスタ、サーバーインスタンス、およびノードエージェントを管理する方法の詳細については、65 ページの「ノードエージェントの配備」を参照してください。

▼ クラスタのサーバーインスタンスを作成するには

始める前に クラスタのサーバーインスタンスを作成する前に、まずノードエージェントまたはノードエージェントのプレースホルダを作成します。75 ページの「ノードエージェントのプレースホルダを作成する」を参照してください。

1 ツリーコンポーネントで、「クラスタ」ノードを開きます。

2 クラスタのノードを選択します。

3 「インスタンス」タブをクリックして、「クラスタ化されたサーバーインスタンス」ページを表示します。

4 「新規」をクリックして、「クラスタ化されたサーバーインスタンスの作成」ページまたは「新しいクラスタ化されたサーバーインスタンス」ページを表示します。

5 「名前」フィールドで、サーバーインスタンスの名前を入力します。

6 「ノードエージェント」ドロップダウンリストからノードエージェントを選択します。

7 「了解」をクリックします。

参考 同機能を持つ asadmin コマンド

`create-instance`

参照
- 63 ページの「ノードエージェントとは」
- 55 ページの「クラスタを作成する」
- 57 ページの「クラスタを設定する」
- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタ用のリソースを設定する」
- 60 ページの「クラスタを削除する」
- 58 ページの「クラスタ内のサーバーインスタンスを設定するには」

▼ クラスタを設定する

1 ツリーコンポーネントで、「クラスタ」ノードを開きます。

2 クラスタのノードを選択します。
「一般情報」ページで、次のタスクを実行できます。
- 「クラスタの起動」をクリックして、クラスタ化されたサーバーインスタンスを起動します。
- 「クラスタの停止」をクリックして、クラスタ化されたサーバーインスタンスを停止します。
- 「EJB タイマーを移行」をクリックして、停止されたサーバーインスタンスからクラスタ内の別のサーバーインスタンスに EJB タイマーを移行します。

参考 同機能を持つ asadmin コマンド

`start-cluster`、`stop-cluster`、`migrate-timers`

参照
- 55 ページの「クラスタを作成する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」
- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタ用のリソースを設定する」
- 60 ページの「クラスタを削除する」
- 61 ページの「EJB タイマーを移行する」

▼ クラスタ化されたインスタンスを起動、停止、および削除する

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. サーバーインスタンスを含むクラスタのノードを開きます。
3. 「インスタンス」タブをクリックして、「クラスタ化されたサーバーインスタンス」ページを表示します。
   このページで、次のタスクを実行できます。
   - インスタンスのチェックボックスを選択して「削除」、「起動」、または「停止」をクリックし、指定したすべてのサーバーインスタンスに対して選択したアクションを実行します。
   - インスタンスの名前をクリックして、「一般情報」ページを表示します。

▼ クラスタ内のサーバーインスタンスを設定するには

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. サーバーインスタンスを含むクラスタのノードを開きます。
3. サーバーインスタンスノードを選択します。
4. 「一般情報」ページでは、次の操作を行えます。
   - 「インスタンスを起動」をクリックして、インスタンスを起動します。
   - 「インスタンスの停止」をクリックして、実行するインスタンスを停止します。
   - 「JNDI ブラウズ」をクリックして、実行中のインスタンスの JNDI ツリーをブラウズします。
   - 「ログファイルを表示」をクリックして、サーバーのログビューアを開きます。
   - 「ログをローテーション」をクリックして、インスタンスのログファイルをローテーションします。このアクションは、ログファイルのローテーションをスケジュールします。実際のローテーションは、次にログファイルがエントリに書き込まれたときに行われます。
   - 「トランザクションの回復」をクリックして、未完了のトランザクションを回復します。
   - 「プロパティー」タブをクリックして、インスタンスのポート番号を変更します。

- 「監視」タブをクリックして、監視プロパティーを変更します。

参照
- 55 ページの「クラスタを作成する」
- 57 ページの「クラスタを設定する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」
- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタ用のリソースを設定する」
- 60 ページの「クラスタを削除する」
- 『Sun GlassFish Communications Server 1.5 管理ガイド』の「Communications Server のトランザクションからの回復方法を設定する」

▼ クラスタ用のアプリケーションを設定する

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「アプリケーション」タブをクリックして、「アプリケーション」ページを表示します。

   このページで、次のタスクを実行できます。

   - 「配備」ドロップダウンリストから、配備するアプリケーションのタイプを選択します。表示される「配備」ページで、アプリケーションを指定します。
   - 「フィルタ」ドロップダウンリストから、リストに表示するアプリケーションのタイプを選択します。
   - アプリケーションを編集するには、アプリケーション名をクリックします。
   - アプリケーションの横にあるチェックボックスを選択して、「有効」または「無効」を選択し、クラスタのアプリケーションを有効または無効にします。

参照
- 55 ページの「クラスタを作成する」
- 57 ページの「クラスタを設定する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」
- 60 ページの「クラスタ用のリソースを設定する」
- 60 ページの「クラスタを削除する」

▼ クラスタ用のリソースを設定する

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「リソース」タブをクリックして、「リソース」ページを表示します。
   このページで、次のタスクを実行できます。
   - クラスタ用の新規リソースを作成します。「新規」ドロップダウンリストから、作成するリソースのタイプを選択します。リソースを作成するときには、必ずクラスタをターゲットとして指定します。
   - リソースをグローバルに有効または無効にします。リソースの横にあるチェックボックスを選択して、「有効」または「無効」をクリックします。このアクションはリソースを削除しません。
   - 特定のタイプのリソースのみを表示します。「フィルタ」ドロップダウンリストから、リストに表示するリソースのタイプを選択します。
   - リソースを編集します。リソース名をクリックします。

参照
- 55 ページの「クラスタを作成する」
- 57 ページの「クラスタを設定する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」
- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタを削除する」

▼ クラスタを削除する

1. ツリーコンポーネントで、「クラスタ」ノードを選択します。
2. 「クラスタ」ページで、クラスタ名の横にあるチェックボックスを選択します。
3. 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

```
delete-cluster
```

参照
- 55 ページの「クラスタを作成する」
- 57 ページの「クラスタを設定する」
- 56 ページの「クラスタのサーバーインスタンスを作成するには」

- 59 ページの「クラスタ用のアプリケーションを設定する」
- 60 ページの「クラスタ用のリソースを設定する」

▼ EJB タイマーを移行する

サーバーインスタンスが異常に、または突然実行を停止した場合、そのサーバーインスタンス上にインストールされた EJB タイマーを、クラスタ内の実行中サーバーインスタンスに移動する必要があります。このためには、次の手順を実行します。

1. ツリーコンポーネントで、「クラスタ」ノードを開きます。
2. クラスタのノードを選択します。
3. 「一般情報」ページで、「**EJB** タイマーを移行」をクリックします。
4. 「**EJB** タイマーを移行」ページで、次の操作を行います。
   a. 「ソース」ドロップダウンリストから、タイマーの移行元である停止されたサーバーインスタンスを選択します。
   b. (省略可能)「送信先」ドロップダウンリストから、タイマーを移行する先の実行中サーバーインスタンスを選択します。
      このフィールドを空のままにした場合、実行中のサーバーインスタンスがランダムに選択されます。
   c. 「了解」をクリックします。
5. 送信先サーバーインスタンスを停止して再起動します。
   ソースサーバーインスタンスが実行中の場合、または送信先サーバーインスタンスが停止中の場合は、管理コンソールにエラーメッセージが表示されます。

参考 同機能を持つ asadmin コマンド

`migrate-timers`

参照
- 57 ページの「クラスタを設定する」
- EJB タイマーサービスの設定に関する管理コンソールオンラインヘルプ

◆ ◆ ◆ 第 4 章

4 ノードエージェントの設定

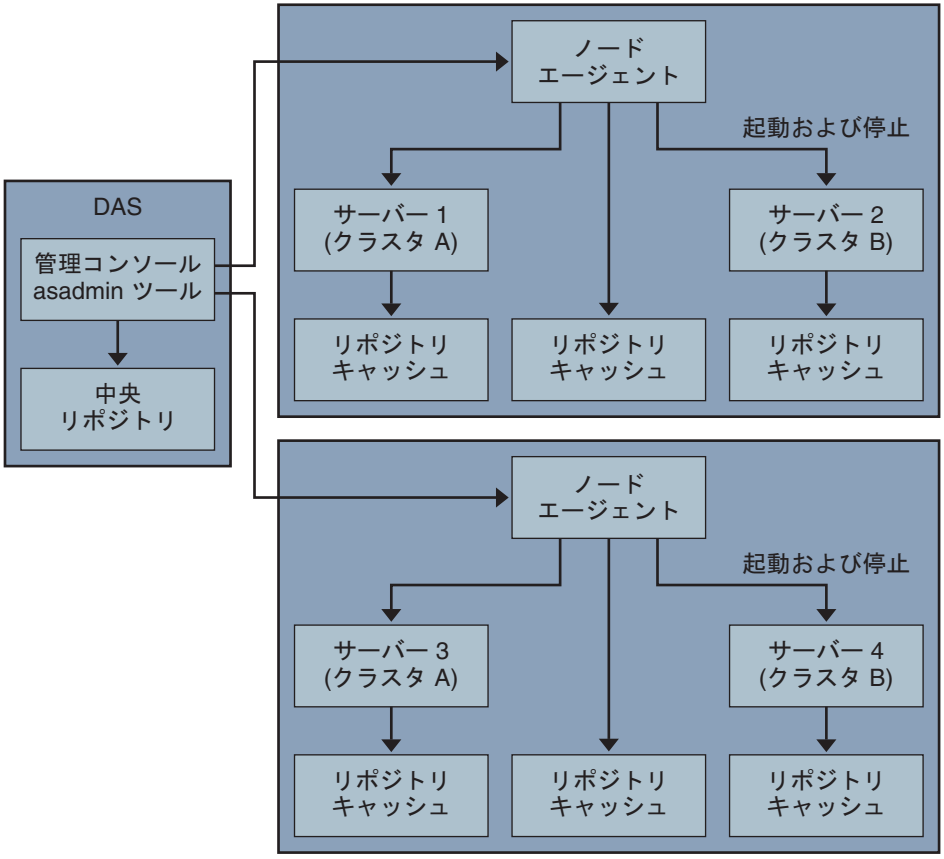
この章では、Communications Server のノードエージェントについて説明します。次の節で構成されています。

ノードエージェントとは

ノードエージェントは、ドメイン管理サーバー (DAS) をホストするマシンを含む、サーバーインスタンスをホストするすべてのマシンに必要な軽量プロセスです。ノードエージェントは次の機能を実行します。

- ドメイン管理サーバーの指示により、サーバーインスタンスの起動、停止、作成、または削除を行います。
- 障害の発生したサーバーインスタンスを再起動します。
- 障害の発生したサーバーのログファイルを表示します。
- 各サーバーインスタンスのローカル設定リポジトリとドメイン管理サーバーの中央リポジトリを同期化します。各ローカルリポジトリには、そのサーバーインスタンスまたはノードエージェントに関する情報のみが含まれます。

  次の図は、ノードエージェントの全体的なアーキテクチャーを示しています。



Application Server をインストールすると、マシンのホスト名を持つノードエージェントがデフォルトで作成されます。このノードエージェントは、実行する前に、ローカルマシン上で手動で起動する必要があります。

ノードエージェントを実行していない場合でも、サーバーインスタンスを作成および管理できます。ただし、ノードエージェントを使用してサーバーインスタンスを起動および停止するには、ノードエージェントが実行中である必要があります。

ノードエージェントは 1 つのドメインを処理します。マシンが複数のドメインで実行されるインスタンスをホストする場合は、複数のノードエージェントを実行する必要があります。

ノードエージェントの障害発生後のサーバーインスタンスの動作

ソフトウェアの障害またはその他のエラーによって、ノードエージェントが予期せず停止する場合があります。この状況では、ノードエージェントが管理していたすべてのサーバーインスタンスは管理されなくなります。ただし、そのようなサーバーインスタンスは稼動を続けており、DAS からもアクセス可能なままです。それらのサーバーインスタンスについての情報はまだ Communications Server の管理インタフェースから入手可能であり、それらのサーバーインスタンスに配備されたアプリケーションへのアクセスもまだ可能です。

ノードエージェントを再起動しても、管理対象外となったサーバーインスタンスは管理されていない状態のままです。ノードエージェントはこれらのサーバーインスタンスの管理を再開しません。ソフトウェアの障害やその他のエラーによって、管理対象外のサーバーインスタンスが予期せず停止した場合、ノードエージェントはそのサーバーインスタンスを再起動できません。

管理対象外のサーバーインスタンスが稼働を続ける必要がある場合、ノードエージェントによるそのサーバーインスタンスの管理を再開することはできません。管理対象外となったサーバーインスタンスの管理を再開するただ 1 つの方法は、ノードエージェントを再起動したあとでサーバーインスタンスを停止し、再起動することです。

ノードエージェントの配備

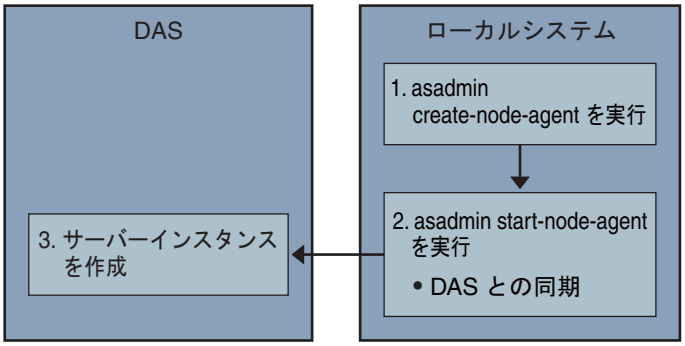
次の 2 とおりの方法で、ノードエージェントの設定および配備ができます。

- オンライン配備: 用いるトポロジがわかっていて、すでにドメイン用のハードウェアが設置されている場合。
- オフライン配備: 完全な環境を設定する前に、ドメインとサーバーインスタンスを設定する場合。

▼ ノードエージェントをオンラインで配備する

すでにドメインのトポロジがわかっていて、ドメイン用のハードウェアが設置されている場合は、オンライン配備を使用します。

次の図は、ノードエージェントのオンライン配備の概要を示しています。



始める前に　ドメイン管理サーバーをインストールして起動します。ドメイン管理サーバーが起動し、実行中になったら、オンラインまたはオフライン配備を開始します。

1. サーバーインスタンスをホストするすべてのマシンにノードエージェントをインストールします。

   インストーラまたは `asadmin create-node-agent` コマンドを使用します。マシンに複数のエージェントが必要な場合は、`asadmin create-node-agent` を使用してエージェントを作成します。

   詳細については、75 ページの「ノードエージェントの作成」を参照してください。

2. `asadmin start-node-agent` コマンドを使用して、ノードエージェントを起動します。

   起動すると、ノードエージェントはドメイン管理サーバー (DAS) と通信します。それが DAS に到達すると、DAS にノードエージェントに対する設定が作成されます。設定が作成されると、管理コンソールでノードエージェントを表示できます。

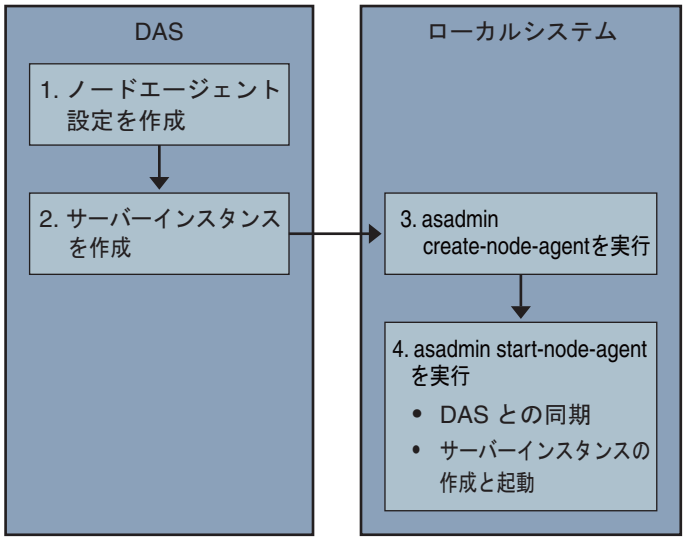
   詳細については、78 ページの「ノードエージェントの起動」を参照してください。

3. ドメインを設定します。 サーバーインスタンスを作成し、クラスタを作成して、アプリケーションを配備します。

▼ ノードエージェントをオフラインで配備する

個々のローカルマシンを設定する前に、オフライン配備を使用してドメイン内にノードエージェントを配備します。

次の図は、オフライン配備の概要を示しています。



始める前に　ドメイン管理サーバーをインストールして起動します。ドメイン管理サーバーが起動し、実行中になったら、オンラインまたはオフライン配備を開始します。

1. ドメイン管理サーバーにプレースホルダノードエージェントを作成します。
   詳細については、75 ページの「ノードエージェントのプレースホルダを作成する」を参照してください。

2. サーバーインスタンスとクラスタを作成して、アプリケーションを配備します。
   サーバーインスタンスを作成するときは、まだ使用されていないポート番号を割り当てるようにしてください。設定がオフラインで実行されるため、作成時にはドメインでポートの競合をチェックすることができません。

3. サーバーインスタンスをホストするすべてのマシンにノードエージェントをインストールします。
   インストーラまたは `asadmin create-node-agent` コマンドを使用します。ノードエージェントには、以前に作成したプレースホルダノードエージェントと同じ名前を付ける必要があります。

   詳細については、75 ページの「ノードエージェントの作成」を参照してください。

4. `asadmin start-node-agent` コマンドを使用して、ノードエージェントを起動します。
   ノードエージェントが起動すると、ドメイン管理サーバーにバインドされ、以前にノードエージェントに関連付けられたサーバーインスタンスを作成します。

   詳細については、78 ページの「ノードエージェントの起動」を参照してください。

ノードエージェントとドメイン管理サーバーとの同期化

設定データは、ドメイン管理サーバーのリポジトリ (中央リポジトリ) に格納されると同時に、ノードエージェントのローカルマシンにもキャッシュされるため、これらの 2 つは同期化する必要があります。キャッシュの同期化は、常に管理ツールでの明示的なユーザーのアクションによって実行されます。

ここでは、次の内容について説明します。

ノードエージェントの同期化

はじめてノードエージェントが起動すると、中央リポジトリの最新情報の要求をドメイン管理サーバー (DAS) に送信します。ノードエージェントが DAS に正常に接続され、設定情報を取得すると、ノードエージェントは DAS にバインドされます。

注 – デフォルトでは、`asadmin start-node-agent` コマンドを使用すると、DAS と同期化せずに、リモートサーバーインスタンスが自動的に起動します。DAS によって管理されている中央リポジトリと同期化しているリモートサーバーインスタンスを起動する場合は、`asadmin start-node-agent` コマンドの `--startinstances=false` オプションを指定します。次に、`asadmin start-instance` コマンドを使用してリモートサーバーインスタンスを起動します。

DAS にプレースホルダノードエージェントを作成した場合、ノードエージェントがはじめて起動するときに、ノードエージェントは DAS の中央リポジトリから設定を取得します。最初の起動時に、DAS が実行されていないため、ノードエージェントが DAS に到達できない場合、ノードエージェントは停止し、バインドされないままの状態になります。

ドメインのノードエージェントの設定が変更された場合、ノードエージェントを実行するローカルマシンのノードエージェントと自動的に通信します。

DAS のノードエージェント設定を削除すると、次に同期するときにノードエージェント自体が停止し、削除待ちとしてマーク付けされます。ローカルの `asadmin delete-node-agent` コマンドを使用して、ノードエージェントを手動で削除します。

サーバーインスタンスの同期化

管理コンソールまたは asadmin ツールを使用してサーバーインスタンスを明示的に起動する場合、サーバーインスタンスは中央リポジトリと同期化されます。この同期が失敗すると、サーバーインスタンスは起動しません。

ノードエージェントが、管理コンソールまたは asadmin ツールによる明示的な要求なしにサーバーインスタンスを起動する場合、サーバーインスタンスのリポジトリキャッシュは同期しません。サーバーインスタンスは、キャッシュに格納された設定によって実行されます。リモートサーバーインスタンスのキャッシュ内にファイルを追加または削除してはいけません。

リモートサーバーインスタンスの設定は、キャッシュとして扱われ (nodeagents/*na1*/*server1* の下にあるすべてのファイル)、Application Server によって所有されます。極端な例を挙げれば、ユーザーがリモートサーバーインスタンスのすべてのファイルを削除し、ノードエージェントを再起動すると、リモートサーバーインスタンス (*server1* など) は再作成され、すべての必要なファイルは同期化されます。

次のファイルおよびディレクトリは Application Server によって同期が保たれます。

表 4–1 リモートサーバーインスタンス間で同期化されるファイルとディレクトリ

ファイルまたはディレクトリ	説明
applications	配備されているすべてのアプリケーション。このディレクトリ (およびサブディレクトリ) は、サーバーインスタンスから参照されるアプリケーションに基づいて同期化されます。ノードエージェントはどのアプリケーションも参照しないので、アプリケーションを同期化しません。
config	ドメイン全体に対する設定ファイルを格納します。実行時の一時ファイル (admch、admsn、secure.seed、.timestamp、__timer_service_shutdown__.dat など) を除いたこのディレクトリ内のすべてのファイルは、同期化されます。
config/*config_name*	*config_name* という名前の設定を使用してすべてのインスタンスによって共有されるファイルを格納するためのディレクトリ。domain.xml で定義されるすべての設定に対して、このようなディレクトリが 1 つずつ存在することになります。このディレクトリ内のすべてのファイルが、*config_name* を使用しているサーバーインスタンスと同期化されます。

表 4–1 リモートサーバーインスタンス間で同期化されるファイルとディレクトリ (続き)

ファイルまたはディレクトリ	説明
config/*config_name*/lib/ext	Java 拡張クラスを (zip または jar アーカイブとして) 置くことができるフォルダ。これは、*config_name* という名前の設定を使用してサーバーインスタンスに配備されたアプリケーションによって使用されます。これらの jar ファイルは、Java 拡張メカニズムを使用してロードされます。
docroot	HTTP ドキュメントルート。既定の設定では、ドメイン内のすべてのサーバーインスタンスが同じ docroot を使用します。それらのサーバーインスタンスに異なる docroot を使用させるためには、仮想サーバーの docroot プロパティーを設定する必要があります。
generated	Java EE アプリケーションやモジュール用に生成されたファイル。たとえば、EJB スタブ、コンパイル済みの JSP クラス、セキュリティーポリシーファイル。このディレクトリは、applications ディレクトリと同様に同期化されます。したがって、サーバーインスタンスによって参照されるアプリケーションに対応するディレクトリのみが同期化されます。
lib、lib/classes	ドメイン全体に配備されたアプリケーションに使用される共通の Java クラスファイルまたは jar および zip アーカイブを置くことができるフォルダ。これらのクラスは、Application Server のクラスローダーを使用してロードされます。クラスローダーでのロード順序は次のとおりです。 lib/classes、lib/*.jar、 lib/*.zip
lib/ext	ドメイン全体に配備されたアプリケーションによって使用される Java 拡張クラスを (jar または zip アーカイブとして) 置くことができるフォルダ。これらの jar ファイルは、Java 拡張メカニズムを使用してロードされます。
lib/applibs	依存する jar ファイルを domains/<*domain_name*>lib/applibs の下に配置し、libraries オプションを使用して jar ファイルへの相対パスを指定します。 たとえば、asadmin deploy --libraries commons-coll.jar,X1.jar foo.ear のようにします。
java-web-start	このディレクトリ (およびサブディレクトリ) の各部分が、サーバーインスタンスから参照されるアプリケーションに基づいて同期化されます。

ライブラリファイルの同期化

アプリケーションの --libraries 配備時属性を使用して、アプリケーションの実行時の依存関係を指定することができます。相対パス (jar の名前のみ) を指定すると、Application Server は指定したライブラリを *domain-dir* /lib/applibs 内で見つけようとします。

ライブラリをドメイン全体で使用できるようにするには、JAR ファイルを *domain-dir*/lib または *domain-dir*/lib/classes に配置することができます。(詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』の「Using the Common Class Loader」を参照してください。) 通常この方法は、JDBC ドライバや、ドメイン内のすべてのアプリケーションによって共有されているその他のユーティリティーライブラリに対してあてはまります。

クラスタ全体またはスタンドアロンのサーバー全体で使用する場合は、jar ファイルを *domain-dir*/domain1/config/*xyz-config*/lib ディレクトリにコピーします。次に、それらの jar ファイルを、*xyz-config* の classpath-suffix または classpath-prefix 要素に追加します。これによって、*xyz-config* を使用するすべてのサーバーインスタンスの jar ファイルが同期化されます。

要約すると、次のようになります。

- domains/domain1/lib - ドメイン全体スコープ、共通のクラスローダー、jar ファイルを自動的に追加。
- domains/domain1/config/cluster1, config/lib - 設定全体、classpath-prefix または classpath-suffix を更新。
- domains/domain1/lib/applibs - アプリケーションスコープ、アプリケーションのクラスローダーに自動的に追加。
- domains/domain1/config/cluster1, config/lib/ext - http://java.sun.com/j2se/1.5.0/docs/guide/extensions/extensions.html に自動的に追加。

固有の設定と設定管理

設定ファイル (domains/*domain1*/config の下) は、ドメイン全体にわたって同期化されます。スタンドアロンのサーバーインスタンス (server1) によって使用される *server1* -config 用に *server.policy* をカスタマイズする場合は、変更後の server.policy ファイルを domains/domain1/config/server1-config ディレクトリの下に配置します。

変更済みの server.policy ファイルは、スタンドアロンのサーバーインスタンス *server1* とのみ同期化されます。 jvm-option を更新することも忘れないでください。次に例を示します。

```
<java-config>
...
<jvm-options>-Djava.security.policy=${com.sun.aas.instanceRoot}/config
/server1-config/server.policy</jvm-options>
</java-config>
```

大きなアプリケーションの同期化

同期化の必要な大きなアプリケーションが使用環境に含まれる場合、または使用できるメモリーが制限されている場合は、JVM オプションを調整してメモリーの使用を制限できます。この調整によって、メモリー不足によるエラーを受信する可能性は低くなります。インスタンス同期化 JVM ではデフォルトの設定が使用されますが、JVM オプションを設定してそれらを変更することもできます。

INSTANCE-SYNC-JVM-OPTIONS プロパティーを使用して、JVM オプションを設定します。このプロパティーを設定するコマンドは次のとおりです。

```
asadmin set
domain.node-agent.node_agent_name.property.INSTANCE-SYNC-JVM-OPTIONS="JVM_options"
```

次に例を示します。

```
asadmin set
domain.node-agent.node0.property.INSTANCE-SYNC-JVM-OPTIONS="-Xmx32m -Xss2m"
```

この例では、ノードエージェントは node0、JVM オプションは -Xmx32m -Xss2m です。

詳細は、http://java.sun.com/docs/hotspot/VMOptions.html を参照してください。

注– ノードエージェントの設定にプロパティーが追加されたり変更されてもノードエージェントは自動的に同期化されないため、INSTANCE-SYNC-JVM-OPTIONS プロパティーの変更後、ノードエージェントを再起動してください。

---

doNotRemoveList フラグの使用

Application Server によって同期化されたディレクトリ (applications、generated、docroot、config、lib、java-web-start) 内のファイルを、アプリケーションによって保存して読み取る必要のある場合は、doNotRemoveList フラグを使用します。この属性は、ファイルまたはディレクトリのコンマ区切りのリストをとります。アプリケーション依存ファイルは、DAS によって管理される中央リポジトリに存在していない場合でも、サーバーの起動時には削除されません。中央リポジトリに同じファイルが存在する場合は、同期化中に上書きされます。

INSTANCE-SYNC-JVM-OPTIONS プロパティーを使用して、doNotRemoveList 属性に渡します。

次に例を示します。

```
<node-agent name="na1" ...>

...
```

```
<property name="INSTANCE-SYNC-JVM-OPTIONS"
value="-Dcom.sun.appserv.doNotRemoveList=applications/j2ee-modules
/<webapp_context>/logs,generated/mylogdir"/>

</node–agent>
```

ノードエージェントログの表示

各ノードエージェントには、固有のログファイルがあります。ノードエージェント関連の問題がある場合、次の場所にあるログファイルを参照します。

node_agent_dir /*node_agent_name*/agent/logs/server.log です。

ノードエージェントログにより、サーバーのログを参照して問題に関する詳細なメッセージを調べるように指示される場合もあります。

サーバーログの場所は以下のとおりです。

node_agent_dir/*node_agent_name*/ *server_name*/logs/server.log

node_agent_dir のデフォルトの位置は *install_dir*/nodeagents です。

ノードエージェントの操作

ノードエージェントタスクの実行方法

一部のノードエージェントタスクについては、ノードエージェントを実行するシステムでローカルに asadmin ツールを使用する必要があります。その他のタスクは、管理コンソールまたは asadmin を使用してリモートで実行できます。

次の表は、タスクとそれを実行する場所の概要です。

表 4–2 ノードエージェントタスクの実行方法

作業	管理コンソール	asadmin コマンド
ノードエージェントのプレースホルダをドメイン管理サーバーに作成します。	「新しいノードエージェントのプレイスホルダ」ページ。	create-node-agent-config
ノードエージェントを作成します。	使用不可	create-node-agent
ノードエージェントを起動します。	使用不可	start-node-agent
ノードエージェントを停止します。	使用不可	stop-node-agent
ドメイン管理サーバーからノードエージェント設定を削除します。	「ノードエージェント」ページ。	delete-node-agent-config
ローカルマシンからノードエージェントを削除します。	使用不可	delete-node-agent
ノードエージェント設定を編集します。	「ノードエージェント」ページ。	set
ノードエージェントを一覧表示します。	「ノードエージェント」ページ。	list-node-agents

ノードエージェントのプレースホルダ

既存のノードエージェントが存在しなくても、ノードエージェントのプレースホルダを使用して、サーバーインスタンスを作成および削除することができます。ノードエージェントのプレースホルダは、ノードエージェント自体がノードエージェントのローカルシステムに作成される前に、ドメイン管理サーバー (DAS) 上に作成されます。

ノードエージェントのプレースホルダの作成については、75 ページの「ノードエージェントのプレースホルダを作成する」を参照してください。

注 – プレースホルダノードエージェントを作成すると、それを使用してドメインにインスタンスを作成できます。ただし、インスタンスを起動する前に、`asadmin` コマンドを使用して、インスタンスが配置されるマシン上に実際のノードエージェントをローカルに作成し、起動する必要があります。詳細については、75 ページの「ノードエージェントの作成」および 78 ページの「ノードエージェントの起動」を参照してください。

▼ ノードエージェントのプレースホルダを作成する

ノードエージェントは、リモートマシンで実行されているサーバーインスタンスのローカルウォッチドッグです。このため、ノードエージェントはサーバーインスタンスをホストしているマシン上に作成する必要があります。この要件の結果、管理コンソールを使用して、ノードエージェントのプレースホルダのみを作成できます。このプレースホルダは、ノードエージェントが存在しない場合のノードエージェントの設定です。

プレースホルダを作成したら、ノードエージェントをホスティングするマシン上で、asadmin コマンドの create-node-agent を使用して作成を完了します。詳細については、75 ページの「ノードエージェントの作成」を参照してください。

ノードエージェントを作成および使用するために必要な手順のリストについては、65 ページの「ノードエージェントの配備」を参照してください。

1. ツリーコンポーネントで、「ノードエージェント」ノードを選択します。
2. 「ノードエージェント」ページで、「新規」をクリックします。
3. 「新しいノードエージェントのプレイスホルダ」ページで、新規ノードエージェントの名前を入力します。
   名前は、ドメインのすべてのノードエージェント名、サーバーインスタンス名、クラスタ名、および設定名の間で一意である必要があります。
4. 「了解」をクリックします。
   新規ノードエージェントのプレースホルダが「ノードエージェント」ページにリスト表示されます。

参考 同機能を持つ asadmin コマンド

create-node-agent-config

ノードエージェントの作成

ノードエージェントを作成するには、ノードエージェントを実行するマシンで、asadmin コマンドの create-node-agent をローカルに実行します。

ノードエージェントのデフォルト名は、ノードエージェントを作成するホストの名前です。

ノードエージェントのプレースホルダをすでに作成している場合は、ノードエージェントプレースホルダと同じ名前を使用して、関連したノードエージェント

を作成します。ノードエージェントのプレースホルダをまだ作成しておらず、DAS が起動していて到達可能である場合、`create-node-agent` コマンドは DAS 上にノードエージェント設定 (プレースホルダ) も作成します。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

セキュリティー保護された通信を行うように、DAS およびノードエージェントが設定される場合があります。この状況では、ノードエージェントが起動されるとき、DAS がノードエージェントに送信する証明書をノードエージェントの側で検証する必要があります。証明書を検証するために、ノードエージェントはそのローカルトラストストアから証明書を検索します。このトラストストアはマスターパスワードによって保護されています。ユーザーにパスワードの入力を求めることなくノードエージェントを起動できるようにするには、ノードエージェントの作成時に、ノードエージェントのマスターパスワードをファイルに保存します。ノードエージェントのマスターパスワードをファイルに保存しない場合、ユーザーはノードエージェントを起動するたびにマスターパスワードの入力を求められます。

注 – 状況によっては、DNS 経由で到達可能なホストの名前を指定する必要があります。詳細については、77 ページの「DNS に到達可能なホストに対してノードエージェントを作成する」を参照してください。

▼ ノードエージェントを作成する

- 次のコマンドを入力します。

  `asadmin create-node-agent --host` *das-host* **`--port`** *port-no* **`--user`** *das-user* **`[--savemasterpassword=true]`** *nodeagent*

  ユーザーにパスワードの入力を求めることなくノードエージェントを起動できるようにするには、ノードエージェントのマスターパスワードをファイルに保存します。ノードエージェントのマスターパスワードをファイルに保存するには、ノードエージェントを作成するためのコマンドで、`--savemasterpassword` オプションを `true` に設定します。

  `--savemasterpassword` を `true` に設定すると、マスターパスワードの入力を求められます。それ以外の場合、パスワードの入力を求められることはありません。

`--host` *das-host*	ドメイン管理サーバー (DAS) が稼働しているホストの名前を指定します。
`-port` *port-no*	ドメインを管理するための HTTP/HTTPS ポート番号を指定します。
`--user` *das-user*	DAS ユーザーを指定します。
nodeagent	作成するノードエージェントの名前を指定します。この名前はドメイン内で一意である必要があります。

例 4–1 ノードエージェントの作成

```
asadmin create-node-agent --host myhost --port 4848 --user admin nodeagent1
```

このコマンドは、nodeagent1 という名前のノードエージェントを作成します。ノードエージェントが通信する DAS は、マシン myhost 上で稼働しています。エージェントのドメインを管理するための HTTP ポートは 4848 です。DAS ユーザーの名前は admin です。

▼ DNS に到達可能なホストに対してノードエージェントを作成する

次の状況では、DAS が稼働しているホストが DNS 経由で到達可能である必要があります。

- ドメインがサブネット境界にまたがっている。すなわち、ノードエージェントと DAS が異なるドメイン (例: sun.com と java.com) 内にある。
- ホスト名が DNS に登録されていないDHCP マシンが使用されている。

1. ドメインを作成するための create-domain コマンドで、--domainproperties domain.hostName=*das-host-name* オプションを指定します。

   das-host-name は、DAS が稼働しているマシンの名前です。

2. ノードエージェントを作成するための create-node-agent コマンドで、次のオプションを指定します。

   - --host *das-host-name*。*das-host-name* は、手順 1 で指定した DAS ホスト名です。このオプションは、ファイル *as-install*/nodeagents/*nodeagentname*/agent/config/das.properties 内の agent.das.host プロパティーに対応します。
   - --agentproperties remoteclientaddress=*node-agent-host-name*。*node-agent-host-name* は、DAS がノードエージェントへの接続に使用するホスト名です。このオプションは、ファイル *as-install*/nodeagents/*nodeagentname*/agent/config/nodeagent.properties 内の agent.client.host プロパティーに対応します。

参考 hosts ファイルの更新によるホストの指定

別の解決法は、プラットフォームに特定のホスト名およびアドレス解決を定義する、hosts ファイルを更新し、ホスト名を正しい IP アドレスに解決することです。ただし、DHCP 使用して再接続する時に、異なる IP アドレスを割り当てられる可能性があります。その場合、各サーバーでホスト解決ファイルを更新する必要があります。

ノードエージェントの起動

ノードエージェントがサーバーインスタンスを管理できるようにするには、ノードエージェントを実行している必要があります。ノードエージェントを起動するには、ノードエージェントが存在するシステムで asadmin コマンドの start-node-agent をローカルに実行します。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

次に例を示します。

```
asadmin start-node-agent --user admin --startinstances=false nodeagent1
```

ここで、admin は管理ユーザーであり、nodeagent1 は起動しているノードエージェントです。

デフォルトでは、ノードエージェントインスタンスのキャッシュリポジトリは、ノードエージェントの再起動時に中央リポジトリから同期されません。インスタンスのキャッシュリポジトリを中央リポジトリと強制的に同期するには、asadmin start-node-agent コマンドで --syncinstances オプションを true に設定します。

注 – --syncinstances オプションを true に設定すると、すべてのインスタンスのリポジトリがノードエージェントの再起動時に同期されます。

ノードエージェントを再起動したあとで、asadmin start-instance コマンドを使用してサーバーインスタンスを起動します。

ノードエージェントの停止

実行中のノードエージェントを停止するには、ノードエージェントが存在するシステムで、asadmin コマンドの stop-node-agent を実行します。stop-node-agent は、ノードエージェントが管理するすべてのサーバーインスタンスを停止します。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

次に例を示します。

```
asadmin stop-node-agent nodeagent1
```

ここで、nodeagent1 はノードエージェントの名前です。

ノードエージェントの削除

ノードエージェントを削除する前に、ノードエージェントを停止する必要があります。ノードエージェントが起動しない場合、またはドメイン管理サーバーに正常に接続できない (バインドされない) 場合も、ノードエージェントを削除できます。

ノードエージェントのファイルを削除するには、ノードエージェントが存在するシステムで、`asadmin` コマンドの `delete-node-agent` を実行します。

コマンド構文の詳しい説明については、コマンドに関するオンラインヘルプを参照してください。

次に例を示します。

```
asadmin delete-node-agent nodeagent1
```

ここで、`nodeagent1` はノードエージェントです。

ノードエージェントを削除する場合は、管理コンソールまたは `asadmin delete-node-agent-config` コマンドのいずれかを使用して、ドメイン管理サーバーからノードエージェントの設定も削除する必要があります。

▼ ノードエージェントの一般情報を表示する

1. ツリーコンポーネントで、「ノードエージェント」ノードを選択します。
2. ノードエージェントの名前をクリックします。
   ノードエージェントがすでに存在するのに、ここに表示されない場合は、ノードエージェントのホストマシンで、`asadmin start-node-agent` を使用して、ノードエージェントを起動します。78 ページの「ノードエージェントの起動」を参照してください。
3. ノードエージェントのホスト名をチェックします。
   ホスト名が「不明なホスト」の場合、ノードエージェントはドメイン管理サーバー (DAS) と初期接続をしていません。
4. ノードエージェントの状態をチェックします。
   この状態は次のいずれかです
   - 稼動中: ノードエージェントが正常に作成され、現在実行中です。
   - 停止中:「ノードエージェントはローカルマシンで作成されているが、起動していない」、または「ノードエージェントは起動したが、その後停止した」のどちらかです。

- ランデブーを待機しています: ノードエージェントは、ローカルマシンで作成されていないプレースホルダです。

詳細については、75 ページの「ノードエージェントの作成」および 78 ページの「ノードエージェントの起動」を参照してください。

5 起動時にインスタンスを起動するかどうかを選択します。

ノードエージェントが起動するときに、ノードエージェントに関連するサーバーインスタンスが自動的に起動するようにするには「Yes」を選択します。インスタンスを手動で起動するには、「No」を選択します。

6 ノードエージェントがドメイン管理サーバーと接続したかどうかを確認します。

ノードエージェントがドメイン管理サーバーと接続していない場合、正常に起動していません。

7 ノードエージェントに関連するサーバーインスタンスを管理します。

ノードエージェントが実行中の場合、インスタンス名の横にあるチェックボックスをクリックし、「起動」または「停止」をクリックしてインスタンスを起動または停止します。

▼ ノードエージェントの設定を削除する

管理コンソールを使用すると、ドメインからノードエージェントの設定を削除することができます。設定は削除できますが、実際のノードエージェントは削除できません。ノードエージェント自体を削除するには、ノードエージェントのローカルマシンで `asadmin` コマンドの `delete-node-agent` を実行します。詳細については、79 ページの「ノードエージェントの削除」を参照してください。

ノードエージェントの設定を削除する前に、ノードエージェントの実行を停止し、関連するインスタンスを削除する必要があります。ノードエージェントを停止するには、`asadmin` コマンドの `stop-node-agent` を使用します。詳細については、78 ページの「ノードエージェントの停止」を参照してください。

1 ツリーコンポーネントで、「ノードエージェント」ノードを選択します。

2 「ノードエージェント」ページで、削除するノードエージェントの横にあるチェックボックスを選択します。

3 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-node-agent-config`

▼ ノードエージェントの設定を編集する

1. ツリーコンポーネントで、「ノードエージェント」ノードを開きます。
2. 編集するノードエージェントの設定を選択します。
3. 「起動時にインスタンスを起動」を「**Yes**」に設定し、エージェントの起動時にエージェントのサーバーインスタンスが起動されるようにします。

   このページから、手動でのインスタンスの起動または停止もできます。

   この設定がプレースホルダノードエージェント用である場合は、`asadmin create-node-agent` を使用して実際のノードエージェントを作成するときに、この設定が引き継がれます。ノードエージェントの作成については、75 ページの「ノードエージェントの作成」を参照してください。

   この設定が既存のノードエージェント用である場合、ノードエージェントの設定情報が自動的に同期されます。

▼ ノードエージェントのレルムを編集する

ノードエージェントに接続しているユーザーの認証レルムを設定する必要があります。管理ユーザーだけがノードエージェントにアクセスできます。

1. ツリーコンポーネントで、「ノードエージェント」ノードを開きます。
2. 編集するノードエージェントの設定を選択します。
3. 「認証レルム」タブをクリックします。
4. 「レルムの編集」ページで、レルムを入力します。

   デフォルトは、ノードエージェントの作成時に作成された `admin-realm` です。別のレルムを使用するには、ドメインによって制御されるすべてのコンポーネントまたは正常に通信しないコンポーネントのレルムを置き換えます。
5. 「クラス名」フィールドで、レルムを実装する **Java** クラスを指定します。

6 必要なプロパティーを追加します。
認証レルムは、特定の実装によって必要とするものが異なるプロバイダ固有のプロパティーが必要です。

▼ ノードエージェントの JMX 対応リスナーを編集するには

ノードエージェントは、JMX を使用してドメイン管理サーバーと通信します。このため、JMX 要求を待機するポートとその他のリスナー情報が必要です。

1 ツリーコンポーネントで、「ノードエージェント」ノードを開きます。

2 編集するノードエージェントの設定を選択します。

3 「**JMX**」タブをクリックします。

4 「アドレス」フィールドに、**IP** アドレスまたはホスト名を入力します。
リスナーが一意のポート値を使用してサーバーのすべての IP アドレスを待機する場合は、「`0.0.0.0`」を入力します。それ以外の場合は、サーバーの有効な IP アドレスを入力します。

5 「ポート」フィールドに、ノードエージェントの **JMX** コネクタが待機するポート番号を入力します。
IP アドレスが「`0.0.0.0`」の場合、ポート番号は一意のものである必要があります。

6 「**JMX** プロトコル」フィールドで、**JMX** コネクタがサポートするプロトコルを入力します。
デフォルトは rmi_jrmp です。

7 「すべてのアドレスを受け付ける」の横にあるチェックボックスをクリックして、すべての **IP** アドレスに接続できるようにします。
ノードエージェントは、ネットワークカードに関連付けられた特定の IP アドレスを待機するか、またはすべての IP アドレスを待機します。すべてのアドレスを許可すると、「待機するホストアドレス」プロパティーに値「`0.0.0.0`」が設定されます。

8 「レルム名」フィールドで、リスナーの認証を処理するレルムの名前を入力します。
このページの「セキュリティ」セクションで、リスナーが SSL、TLS、あるいはこの両方のセキュリティーを使用するように設定します。

安全なリスナーを設定するには、次の手順を実行します。

9 「セキュリティー」フィールドの「有効」ボックスにチェックマークを付けます。
デフォルトで、セキュリティーが有効になります。

10 クライアント認証を設定します。
このリスナーを使っている個々のクライアントにサーバーへの認証を要求する場合は、「クライアント認証」フィールドの「有効」ボックスにチェックマークを付けます。

11 証明書のニックネームを入力します。
「証明書のニックネーム」フィールドに、既存サーバーの鍵ペアと証明書の名前を入力します。

証明書および SSL の操作の詳細については、管理コンソールのオンラインヘルプを参照してください。

12 **SSL3/TLS** セクションでは次の手順を実行します。

a. リスナーで有効にするセキュリティープロトコルにチェックマークを付けます。
SSL3 と TLS のどちらか、または両方のプロトコルにチェックマークを付ける必要があります。

b. プロトコルが使用する暗号化方式群にチェックマークを付けます。
すべての暗号化方式群を有効にするには、「サポートされるすべての暗号化方式群」にチェックマークを付けます。

13 「保存」をクリックします。

▼ スタンドアロンのサーバーインスタンスを作成する

1 ツリーコンポーネントで、「スタンドアロンインタンス」ノードを選択します。

2 「新規」をクリックします。

3 「新しいスタンドアロンサーバーインスタンス」ページで、インスタンスの名前を入力します。

4 このインスタンスが作成されるノードエージェントを選択し、「**OK**」をクリックします。

◆ ◆ ◆ 第 5 章

設定の管理

この章では、Communications Server における名前付きサーバー設定の追加、変更、および使用について説明します。次の節で構成されています。

設定の使用

設定

設定とは一連のサーバー設定情報で、HTTP/SIP リスナー、ORB/IIOP リスナー、JMS ブローカ、EJB コンテナ、セキュリティー、ロギング、および監視などの設定が含まれます。アプリケーションやリソースは、名前付き設定では定義されません。

設定は管理ドメインに配置されます。ドメイン内の複数のサーバーインスタンスが同じ設定を参照したり、別個の設定を使用したりできます。

クラスタでは、クラスタのインスタンスで均質の環境が確保されるように、クラスタ内のすべてのサーバーインスタンスがクラスタの設定を継承します。

設定には数多くの必須設定情報が含まれるため、既存の名前付き設定をコピーして新しい設定を作成します。設定情報を変更しないかぎり、新規に作成された設定はコピーした設定と同じです。

クラスタまたはインスタンスが設定を使用するには 3 つの方法があります。

- 「スタンドアロン:」スタンドアロンのサーバーインスタンスまたはクラスタは、ほかのサーバーインスタンスまたはクラスタと設定を共有しません。つまり、ほかのサーバーインスタンスまたはクラスタはその名前付き設定を参照しません。スタンドアロンのインスタンスまたはクラスタは、既存の設定をコピーして名前を変更することにより作成します。
- 共有: 共有サーバーインスタンスまたはクラスタは、ほかのサーバーインスタンスまたはクラスタと設定を共有します。つまり、複数のインスタンスまたはクラスタが同じ名前付き設定を参照します。共有サーバーインスタンスまたはクラスタは、既存の設定をコピーするのではなく参照することにより作成します。
- クラスタ化: クラスタ化されたサーバーインスタンスはクラスタの設定を継承します。

関連項目

- 86 ページの「default-config 設定」
- 86 ページの「インスタンスまたはクラスタの作成時に作成された設定」
- 87 ページの「一意のポート番号と設定」
- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」

default-config 設定

`default-config` 設定は、スタンドアロンサーバーインスタンスまたはスタンドアロンクラスタの設定を作成するテンプレートとして機能する特殊な設定です。クラスタとサーバーインスタンスは、`default-config` 設定を参照できません。この設定は、新しい設定を作成するためにコピーできるだけです。デフォルト設定を編集して、コピーした新しい設定が正しく初期設定されているかどうか確認します。

詳細は、次の項目を参照してください。

- 86 ページの「インスタンスまたはクラスタの作成時に作成された設定」
- 85 ページの「設定」
- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」
- 90 ページの「設定を参照するインスタンスのポート番号を編集する」

インスタンスまたはクラスタの作成時に作成された設定

新しいサーバーインスタンスまたは新しいクラスタを作成する場合は、次のどちらかを実行します。

- 既存の設定を参照します。新しい設定は追加されません。

- 既存の設定のコピーを作成します。サーバーインスタンスまたはクラスタを追加すると、新しい設定が追加されます。

デフォルトでは、`default-config` 設定からコピーした設定を使用して新しいクラスタまたはインスタンスが作成されます。別の設定からコピーするには、新規インスタンスまたはクラスタの作成時に設定を指定します。

サーバーインスタンスの場合、新しい設定には *instance_name* `-config` という名前が付けられます。クラスタの場合、新しい設定には *cluster-name* `-config` という名前が付けられます。

詳細は、次の項目を参照してください。

- 86 ページの「default-config 設定」
- 85 ページの「設定」
- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」

クラスタ化された設定の同期

クラスタ化された設定を作成すると、Communications Server によってクラスタ設定ディレクトリがドメイン管理サーバーの *domain-root*`/`*domain-dir*`/config/cluster-config` に作成されます。このディレクトリは、クラスタ内ですべてのインスタンスの設定を同期するために使われます。

一意のポート番号と設定

同じホストマシン上の複数のインスタンスが同じ設定を参照する場合、各インスタンスは固有のポート番号で待機する必要があります。たとえば、ポート 80 の HTTP リスナーを使用する名前付き設定を 2 つのサーバーインスタンスが参照する場合、ポートの競合により、どちらかのサーバーインスタンスが起動できなくなります。一意のポートが使用されるように、個々のサーバーインスタンスが待機するポート番号を定義するプロパティーを変更します。

ポート番号に次の原則を適用します。

- 個々のサーバーインスタンスのポート番号は、最初に設定から継承されます。
- サーバーインスタンスの作成時にポートがすでに使用されている場合は、継承されたデフォルト値をインスタンスレベルでオーバーライドして、ポートの競合を防止します。
- インスタンスが設定を共有しているものと仮定します。設定にはポート番号 n があります。同じ設定を使用するマシンで新しいインスタンスを作成する場合、新しいインスタンスにはポート番号 *n*+1 が割り当てられます (使用可能な場合)。この番号が使用できない場合は、*n*+1 の次に使用可能なポートが選択されます。
- 設定のポート番号を変更する場合、そのポート番号を継承するサーバーインスタンスは変更されたポート番号を自動的に継承します。

- インスタンスのポート番号を変更し、続いて設定のポート番号を変更する場合、インスタンスのポート番号は変更されません。

  詳細は、次の項目を参照してください。
- 90 ページの「設定を参照するインスタンスのポート番号を編集する」
- 89 ページの「名前付き設定のプロパティーの編集」
- 85 ページの「設定」

名前付き設定に関連した作業

- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」
- 90 ページの「設定を参照するインスタンスのポート番号を編集する」
- 91 ページの「名前付き設定のターゲットを表示する」
- 91 ページの「名前付き設定を削除する」

▼ 名前付き設定を作成する

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 「設定」ページで、「新規」をクリックします。
3. 「設定の作成」ページで、一意の設定の名前を入力します。または「新しい設定」ページで、一意の設定の名前を入力します。
4. 設定を選択して、コピーします。

   `default-config` 設定は、スタンドアロンサーバーインスタンスまたはスタンドアロンクラスタを作成するときに使用するデフォルトの設定です。

参考 同機能を持つ asadmin コマンド

`copy-config`

参照
- 85 ページの「設定」
- 86 ページの「default-config 設定」
- 89 ページの「名前付き設定のプロパティーの編集」
- 90 ページの「設定を参照するインスタンスのポート番号を編集する」
- 91 ページの「名前付き設定のターゲットを表示する」
- 91 ページの「名前付き設定を削除する」

名前付き設定のプロパティーの編集

次の表は、設定に対して定義済みのプロパティーを示しています。

あらかじめ定義されたプロパティーはポート番号です。有効なポートの値は 1 ～ 65535 です。UNIX では、ポート 1 ～ 1024 で待機するソケットを作成するには、スーパーユーザー権限が必要です。複数のサーバーインスタンスがある場合、ポート番号は一意にする必要があります。

プロパティー名	説明
HTTP_LISTENER_PORT	http-listener-1 のポート番号。
HTTP_SSL_LISTENER_PORT	http-listener-2 のポート番号。
IIOP_SSL_LISTENER_PORT	IIOP リスナー SSL が待機する IIOP 接続用の ORB リスナーポート。
IIOP_LISTENER_PORT	orb-listener-1 が待機する IIOP 接続用の ORB リスナーポート。
JMX_SYSTEM_CONNECTOR_PORT	JMX コネクタが待機するポート番号。
IIOP_SSL_MUTUALAUTH_PORT	IIOP リスナー SSL_MUTUALAUTH が待機する IIOP 接続のORB リスナーポート。

▼ 名前付き設定のプロパティーを編集する

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。
3. 「システムプロパティー」を選択します。
4. 「システムプロパティーの設定」ページで、動的再設定を有効にするかどうかを選択します。
   有効な場合は、設定に対する変更は、サーバーを再起動することなくサーバーインスタンスに適用されます。
5. 必要に応じて、プロパティーを追加、削除、または変更します。
6. 設定に関連するすべてのインスタンスの現在のプロパティーの値を編集するには、「インスタンス値」をクリックします。

参考 同機能を持つ asadmin コマンド

`set`

参照
- 85 ページの「設定」
- 88 ページの「名前付き設定を作成する」
- 91 ページの「名前付き設定のターゲットを表示する」
- 91 ページの「名前付き設定を削除する」

▼ 設定を参照するインスタンスのポート番号を編集する

名前付き設定を参照する各インスタンスは、最初にその設定からポート番号を継承します。ポート番号はシステムで一意である必要があるため、継承されたポート番号をオーバーライドする必要があります。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。
3. 「システムプロパティー」を選択します。
   管理コンソールに「システムプロパティーの設定」ページが表示されます。
4. 編集するインスタンス変数の横にある「インスタンス値」をクリックします。
   たとえば、HTTP-LISTENER-PORT インスタンス変数の横にある「インスタンス値」をクリックすると、その設定を参照するすべてのサーバーインスタンスの HTTP-LISTENER-PORT の値が表示されます。
5. 必要に応じて値を変更して、「保存」をクリックします。

参考 同機能を持つ asadmin コマンド

`set`

参照
- 87 ページの「一意のポート番号と設定」
- 85 ページの「設定」
- 89 ページの「名前付き設定のプロパティーの編集」

▼ 名前付き設定のターゲットを表示する

「システムプロパティーの設定」ページに、設定を使用するすべてのターゲットのリストが表示されます。クラスタ設定の場合、ターゲットはクラスタです。インスタンス設定の場合、ターゲットはインスタンスです。

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 名前付き設定のノードを選択します。
3. 「システムプロパティー」を選択します。
   管理コンソールに「システムプロパティーの設定」ページが表示されます。

参照
- 87 ページの「一意のポート番号と設定」
- 85 ページの「設定」
- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」
- 91 ページの「名前付き設定を削除する」

▼ 名前付き設定を削除する

1. ツリーコンポーネントで、「設定」ノードを選択します。
2. 「設定」ページで、削除する名前付き設定のチェックボックスにチェックマークを付けます。
   `default-config` 設定は削除できません。
3. 「削除」をクリックします。

参考 同機能を持つ asadmin コマンド

`delete-config`

参照
- 85 ページの「設定」
- 88 ページの「名前付き設定を作成する」
- 89 ページの「名前付き設定のプロパティーの編集」
- 91 ページの「名前付き設定のターゲットを表示する」

◆ ◆ ◆ 第 6 章

高可用性 (HA) セッション持続性とフェイルオーバーの設定

この章では、高可用性セッション持続性の有効化と設定を行う方法について説明します。

セッション持続性とフェイルオーバーの概要

Communications Server は、HTTP セッションデータおよびステートフルセッション Bean (SFSB) セッションデータのフェイルオーバーを通して、高可用性セッション持続性を提供します。フェイルオーバーとは、サーバーインスタンスまたはハードウェアに障害が発生しても、別のサーバーインスタンスが分散セッションを引き継ぐことを意味します。

要件

分散セッションは、次の条件が満たされた場合に、複数の Sun GlassFish Communications Server インスタンスで動作できます。

- 各サーバーインスタンスが、同じセッション状態データにアクセスする。Communications Server では、HTTP セッションおよびステートフルセッション Bean のデータを格納するための、次のタイプの高可用性ストレージが用意されています。
  - クラスタ内の別のサーバーにおけるインメモリーレプリケーション。インメモリーレプリケーションは、クラスタプロファイルではデフォルトで有効です。

インメモリーレプリケーションを使用するには、「グループ管理サービス (GMS)」 を有効にする必要があります。GMS の詳細については、52 ページの「グループ管理サービス」を参照してください。

クラスタ内の複数のサーバーインスタンスが異なるマシンに配置されている場合は、次の前提条件が満たされていることを確認してください。

- GMS およびインメモリーレプリケーションが正常に機能することを保証するには、すべてのマシンが同じサブネット上に存在している必要があります。
- インメモリーレプリケーションが正常に機能することを保証するには、クラスタ内のすべてのマシンのシステムクロックができるだけ厳密に同期している必要があります。

- 各サーバーインスタンスに、同じ分散可能な Web アプリケーションが配備されていること。`web.xml` 配備記述子ファイルの `web-app` 要素に、`distributable` 要素が含まれている必要があります。
- Web アプリケーションが、高可用性セッション持続性を使用していること。分散可能でない Web アプリケーションが、高可用性セッション持続性を使用するように設定されていると、サーバーはログファイルにエラーを書き込みます。
- Web アプリケーションは、`--availabilityenabled` オプションが `true` に設定された `deploy` または `deploydir` コマンドを使用して配備されている必要があります。これらのコマンドの詳細については、`deploy(1)`および `deploydir(1)`を参照してください。

制限事項

セッションが処理を継続すると、ファイルを開くための参照やネットワーク接続はすべて失われます。アプリケーションは、この制限を念頭においてコード化する必要があります。

フェイルオーバーをサポートする分散セッションには、特定のオブジェクトしかバインドできません。サーブレット 2.4 仕様とは異なり、Sun GlassFish Communications Server は、フェイルオーバーがサポートされていないオブジェクト型が分散セッションにバインドされると `IllegalArgumentException` をスローしません。

フェイルオーバーをサポートする分散セッションには、次のオブジェクトをバインドできます。

- すべての EJB コンポーネントに対するローカルホームおよびオブジェクト参照。
- 共存ステートレスセッション Bean、ステートフルセッション Bean、またはエンティティー Bean の参照。
- 分散ステートレスセッション Bean、ステートフルセッション Bean、またはエンティティー Bean の参照。

- `InitialContext` および `java:comp/env` に対する JNDI コンテキスト。
- `UserTransaction` オブジェクト。ただし、失敗したインスタンスが再起動されない場合は、準備されたグローバルトランザクションはすべて失われ、正しくロールバックまたはコミットされない可能性もあります。
- 直列化可能な Java 型。

フェイルオーバーをサポートする分散セッションには、次のオブジェクト型をバインドできません。

- JDBC データソース
- Java Message Service (JMS) の `ConnectionFactory` および `Destination` オブジェクト
- JavaMail™ セッション
- 接続ファクトリ
- 管理対象オブジェクト
- Web サービス参照

一般に、これらのオブジェクトに対して、フェイルオーバーは機能しません。ただし、オブジェクトが直列化可能な場合など、フェイルオーバーが機能する場合もあります。

高可用性セッション持続性の設定

この節では、高可用性セッション持続性を設定する方法について、次のトピックとともに説明します。

- 95 ページの「高可用性セッション持続性を設定する」
- 96 ページの「セッション可用性の有効化」

▼ 高可用性セッション持続性を設定する

始める前に　高可用性セッション持続性は、動的配備、動的再読み込み、および自動配備とは互換性がありません。これらの機能は、本稼働環境ではなく開発環境を対象としているため、HA セッション持続性を有効にする前に無効にする必要があります。これらの機能を無効にする方法については、『Sun GlassFish Communications Server 1.5 Application Deployment Guide』を参照してください。

1. **Communications Server クラスタを作成します。**
   詳細については、55 ページの「クラスタを作成する」を参照してください。

2. **クラスタの負荷分散を設定します。**
   詳細は、Microsoft サポートページの記事

3 目的のアプリケーションサーバーインスタンス、および**Web**または**EJB**コンテナの可用性を有効にします。

次に、セッション持続性の設定を行います。次の方法のうち1つを選択します。

- 管理コンソールを使用します。97ページの「サーバーインスタンスの可用性の有効化」を参照してください。
- `asadmin`コマンド行ユーティリティーを使用します。`set(1)`を参照してください。

4 クラスタ内の各サーバーインスタンスを再起動します。

5 可用性を必要とする特定の**SFSB**の可用性を有効にします。

セッション状態にチェックポイントを設定する必要のあるメソッドを選択します。105ページの「個々のBeanの可用性の設定」を参照してください。

6 高可用性を必要とする各**Web**モジュールを分散可能**(distributable)**にします。

7 配備中に、個々のアプリケーション、**Web**モジュール、または**EJB**モジュールの可用性を有効にします。

105ページの「個々のアプリケーションまたはEJBモジュールの可用性の設定」を参照してください。

管理コンソールで、可用性を有効にするチェックボックスをチェックするか、または`--availabilityenabled`オプションを`true`にして`asadmin deploy`コマンドを実行します。

セッション可用性の有効化

セッション可用性は、次の5つの異なるスコープ(高いレベルから低いレベルへの順)で有効にすることができます。

1. サーバーインスタンス。デフォルトでは有効になっています。サーバーインスタンスのセッション可用性を有効にすると、サーバーインスタンスで実行されているすべてのアプリケーションが高可用性セッション持続性を持つことができるようになります。手順については、次の節の97ページの「サーバーインスタンスの可用性の有効化」を参照してください。
2. コンテナ(WebまたはEJB)。デフォルトでは有効になっています。コンテナレベルでの可用性の有効化については、次の節を参照してください。
   - 98ページの「Webコンテナの可用性の設定」
   - 103ページの「EJBコンテナの可用性の設定」
3. アプリケーション。デフォルトでは無効になっています。
4. スタンドアロンのWebまたはEJBモジュール。デフォルトでは無効になっています。
5. 個々のSFSB。デフォルトでは無効になっています。

可用性を指定されたスコープで有効にするには、それより上のすべてのレベルでも有効にする必要があります。たとえば、アプリケーションレベルで可用性を有効にするには、サーバーインスタンスレベルおよびコンテナレベルでも有効にする必要があります。

ある特定のレベルの可用性は、デフォルトでは 1 つ上のレベルに設定されます。たとえば、可用性がコンテナレベルで有効になっている場合、デフォルトではアプリケーションレベルで有効になります。

可用性がサーバーインスタンスレベルで無効になっている場合、ほかのすべてのレベルで有効にしても反映されません。可用性がサーバーインスタンスレベルで有効になっている場合、明示的に無効化しないかぎり、すべてのレベルで有効になります。

サーバーインスタンスの可用性の有効化

サーバーインスタンスの可用性を有効にするには、`asadmin set` コマンドを使用して、設定の `availability-service.availability-enabled` プロパティーを true に設定します。

たとえば、設定の名前が `config1` の場合は、次のように指定します。

```
asadmin set --user admin --passwordfile password.txt
--host localhost
--port 4849
config1.availability-service.availability-enabled="true"
```

▼ 管理コンソールを使用してサーバーインスタンスの可用性を有効にする

1. ツリーコンポーネントで、「設定」ノードを開きます。
2. 編集する設定のノードを展開します。
3. 「可用性サービス」ノードを選択します。
4. 「可用性サービス」ページで、「可用性サービス」ボックスにチェックマークを付けて、インスタンスレベルの可用性を有効にします。
   無効にするには、このボックスのチェックマークを外します。
5. 「保存」ボタンをクリックします。
6. サーバーインスタンスを停止し、再起動します。

HTTP セッションフェイルオーバー

Java EE アプリケーションは一般に、大量のセッション状態データを保持しています。Web ショッピングカートは、セッション状態の古典的な例です。アプリケーションはまた、頻繁に必要になるデータをセッションオブジェクトにキャッシュすることもできます。実際、ユーザーとの対話が多いほぼすべてのアプリケーションには、セッション状態の保持が必要になります。

Web コンテナの可用性の設定

注 – セッション状態データの格納にインメモリーレプリケーションを使用している場合、Web コンテナ可用性の有効化およびプロパティーの設定は `asadmin set` コマンドを使用して行う必要があります。

たとえば、`set` コマンドを使用して次のように指定します。ここで、`config1` は設定の名前です。

```
asadmin set
config1.availability-service.web-container-availability.availability-enabled="true"
```

```
asadmin set
config1.availability-service.web-container-availability.persistence-frequency="time-based"
```

▼ 管理コンソールを使用して Web コンテナの可用性を有効にする

1. ツリーコンポーネントで、目的の設定を選択します。
2. 「可用性サービス」をクリックします。
3. 「Web コンテナの可用性」タブを選択します。
   「可用性サービス」ボックスにチェックマークを付けて、可用性を有効にします。無効にするには、このボックスのチェックマークを外します。
4. 次の節の 98 ページの「Web コンテナの可用性の設定」の説明に従って、ほかの設定を変更します。
5. サーバーインスタンスを再起動します。

Web コンテナの可用性の設定

「可用性サービス」の「Web コンテナの可用性」タブを使用すると、次の可用性設定を変更できます。

持続型: 可用性が有効化されている Web アプリケーションのセッションの持続性メカニズムを指定します。使用できる値は、memory (持続性なし)、file (ファイルシステム)、および replicated (ほかのサーバー上のメモリー) です。

Web コンテナの可用性が有効にされている場合、次の表に示されているように、デフォルトの持続型はプロファイルに応じて異なります。

プロファイル	持続型
開発者	memory
クラスタ	replicated

セッションの持続性が必要となる本稼動環境では、replicated を使用します。memory および file の持続型は、高可用性セッション持続性を提供しません。

Web コンテナの可用性を無効にする場合、デフォルトの持続型は memory です。

持続性の頻度: セッション状態を格納する頻度を指定します。持続型が replicated の場合にのみ適用できます。指定できる値は次のとおりです。

- web-method - セッション状態は、各 Web 要求の終了時に、クライアントに応答を返信する前に格納されます。このモードでは、障害発生時にセッション状態を完全に更新するための最良の保証が得られます。これはデフォルトの設定です。
- time-based - セッション状態が、reapIntervalSeconds ストアプロパティーによって設定された頻度でバックグラウンドに格納されます。このモードでは、セッション状態が必ずしも完全に更新される保証は得られません。ただし、各要求後に状態が格納されないので、パフォーマンスが大幅に向上します。

持続性の範囲: 格納するセッションオブジェクトの範囲と、セッション状態を格納する頻度を指定します。持続型が replicated の場合にのみ適用できます。使用できる値は次のとおりです。

- session - 常にすべてのセッション状態が格納されます。このモードでは、セッションデータを分散可能な Web アプリケーションに正しく格納するための最良の保証が得られます。これはデフォルトの設定です。
- modified-session - セッション状態が変更された場合、すべてのセッション状態が格納されます。HttpSession.setAttribute() または HttpSession.removeAttribute() が呼び出された場合に、セッションが変更されたと見なします。属性が変更されるたびに、必ず setAttribute() を呼び出す必要があります。これは Java EE 仕様の要件ではありませんが、このモードを正しく動作させるために必要になります。
- modified-attribute - 変更されたセッション属性だけが格納されます。このモードを正しく動作させるには、次のガイドラインに従う必要があります。
  - セッション状態が変更されるたびに、setAttribute() を呼び出します。

- 属性間で相互参照しないようにします。別個の各属性キーにあるオブジェクトグラフを直列化し、別々に格納します。別個の各キーにあるオブジェクト間に相互参照がある場合は、正常な直列化および直列化復元は行われません。
- 複数の属性間、または少なくとも読み取り専用属性と変更可能な属性間でセッション状態を分散します。

シングルサインオン状態: シングルサインオン状態の持続性を有効にするには、このボックスにチェックマークを付けます。無効にするには、このボックスのチェックマークを外します。詳細については、100 ページの「セッションフェイルオーバーでのシングルサインオンの使用」を参照してください。

個々の Web アプリケーションの可用性の設定

個々の Web アプリケーションの可用性の有効化と設定を行うには、アプリケーション配備記述子ファイル `sun-web.xml` を編集します。アプリケーションの配備記述子の設定は、Web コンテナの可用性の設定より優先されます。

`session-manager` 要素の `persistence-type` 属性によって、アプリケーションが使用するセッション持続性のタイプが決定されます。高可用性セッション持続性を有効にするには、この要素を `replicated` に設定する必要があります。

`sun-web.xml` ファイルの詳細については、『Sun GlassFish Communications Server 1.5 Application Deployment Guide』の「The sun-web.xml File」を参照してください。

例

```
<sun-web-app> ...
  <session-config>
    <session-manager persistence-type="replicated">
      <manager-properties>
        <property name="persistenceFrequency" value="web-method" />
      </manager-properties>
      <store-properties>
        <property name="persistenceScope" value="session" />
      </store-properties>
    </session-manager> ...
</session-config> ...
```

セッションフェイルオーバーでのシングルサインオンの使用

単一のアプリケーションサーバーインスタンスにおいて、ユーザーがあるアプリケーションによって一度認証されると、同じインスタンス上で動作しているほかの

アプリケーションに対する個別の再認証は必要ありません。これをシングルサインオンといいます。詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』の「User Authentication for Single Sign-on」を参照してください。

HTTP セッションがクラスタ内のほかのインスタンスにフェイルオーバーした場合でも、シングルサインオンが機能し続けるようにするには、インメモリーレプリケーションを使用して、シングルサインオン情報が持続される必要があります。シングルサインオン情報を持続させるには、最初にサーバーインスタンスと Web コンテナの可用性を有効にし、次にシングルサインオン状態のフェイルオーバーを有効にします。

98 ページの「Web コンテナの可用性の設定」で説明するように、シングルサインオン状態のフェイルオーバーは、管理コンソールの「可用性サービス」の「Web コンテナの可用性」タブで有効にすることができます。`asadmin set` コマンドを使用して、設定の `availability-service.web-container-availability.sso-failover-enabled` プロパティーを true に設定することもできます。

たとえば、`set` コマンドを使用して次のように指定します。ここで、`config1` は設定の名前です。

```
asadmin set --user admin --passwordfile password.txt
--host localhost --port 4849
config1.availability-service.web-container-availability.
sso-failover-enabled="true"
```

シングルサインオングループ

単一の名前とパスワードの組み合わせによってアクセス可能なアプリケーションは、シングルサインオングループを構成します。シングルサインオングループに属するアプリケーションに対応する HTTP セッションでは、1 つのセッションがタイムアウトになった場合でも、ほかのセッションは無効化されず、引き続き有効となります。これは、1 つのセッションがタイムアウトしてもほかのセッションの可用性に影響を与えるべきではないからです。

この動作の当然の結果として、あるセッションがタイムアウトして、セッションを実行していた同じブラウザウィンドウから対応するアプリケーションにアクセスを試みる場合、再度認証を行う必要はありません。ただし、新しいセッションが作成されます。

シングルサインオングループに属するショッピングカートアプリケーションの例を挙げます。このグループにはほかに 2 つのアプリケーションが含まれます。ほかの 2 つのアプリケーションのセッションタイムアウト値は、ショッピングカートアプリケーションのセッションタイムアウト値を上回るものと仮定します。ショッピングカートアプリケーションのセッションがタイムアウトして、セッションを実行していた同じブラウザウィンドウからショッピングカートアプリケーションの実行を試

みる場合、再度認証を行う必要はありません。ただし、以前のショッピングカートは失われていて、新しいショッピングカートを作成する必要があります。ほかの 2 つのアプリケーションは、ショッピングカートアプリケーションを実行していたセッションのタイムアウト後も変わらず動作し続けます。

同様に、ほかの 2 つのアプリケーションのどちらかに対応するセッションがタイムアウトしたとします。セッションを実行していた同じブラウザウィンドウからアプリケーションに接続している間は、再度認証を行う必要はありません。

注 – この動作は、セッションがタイムアウトした場合にのみ当てはまります。シングルサインオンが有効になっていて、`HttpSession.invalidate()` を使用してセッションの 1 つを無効にする場合、シングルサインオングループに属するすべてのアプリケーションのセッションが無効になります。シングルサインオングループに属する任意のアプリケーションへのアクセスを試みる場合、再認証が必要であり、アプリケーションにアクセスするクライアントに対して新しいセッションが作成されます。

ステートフルセッション **Bean** のフェイルオーバー

ステートフルセッション Bean (SFSB) には、クライアント固有の状態が含まれています。クライアントとステートフルセッション Bean の間には、一対一の関係が存在します。作成時、EJB コンテナは各 SFSB に、クライアントにバインドするための一意のセッション ID を割り当てます。

サーバーインスタンスが失敗した場合に備えて、SFSB の状態を持続的なストアに保存することができます。SFSB の状態は、そのライフサイクル内のあらかじめ定義された時点に、持続性ストアに保存されます。これを、チェックポイント設定と呼びます。有効になっている場合、チェックポイント設定は一般に、トランザクションがロールバックする場合でも、Bean がトランザクションを完了したあとに実行されます。

ただし、SFSB が Bean 管理によるトランザクションに参加している場合、そのトランザクションは Bean メソッドの実行の途中でコミットされる可能性があります。このメソッド呼び出しの結果、Bean の状態は遷移している途中である可能性があるため、これは Bean の状態にチェックポイントを設定するのに適切なタイミングではありません。この場合、EJB コンテナは、対応するメソッドの終了時に Bean の状態にチェックポイントを設定します。ただし、メソッドの終了時に、その Bean が別のトランザクションの範囲に入っていないことが前提です。Bean 管理によるトランザクションが複数のメソッドにまたがっている場合は、後続のメソッドの終了時にアクティブなトランザクションが存在しなくなるまで、チェックポイント設定が遅延されます。

SFSB の状態は必ずしもトランザクションではなく、非トランザクションビジネスメソッドの結果として大幅に変更される可能性もあります。SFSB がこれに当てはまる

場合は、106 ページの「チェックポイントを設定するメソッドの指定」で説明しているように、チェックポイントを設定するメソッドのリストを指定することができます。

分散可能 Web アプリケーションが SFSB を参照しており、その Web アプリケーションのセッションがフェイルオーバーする場合は、EJB 参照もフェイルオーバーされます。

Communications Server インスタンスの停止中に、セッション持続性を使用している SFSB の配備が取り消されると、持続性ストア内のセッションデータがクリアされない可能性があります。これを回避するには、Communications Server インスタンスが動作している間、SFSB の配備を取り消します。

EJB コンテナの可用性の設定

▼ EJB コンテナの可用性を設定する

1. 「EJB コンテナの可用性」タブを選択します。
2. 「可用性サービス」ボックスにチェックマークを付けます。
   可用性を無効にするには、このボックスのチェックマークを外します。
3. 104 ページの「可用性の設定」の説明に従って、ほかの設定を変更します。
4. 「保存」ボタンをクリックします。
5. サーバーインスタンスを再起動します。

参考 同機能を持つ asadmin コマンド

EJB コンテナの可用性を有効にするには、`asadmin set` コマンドを使用して、設定に次の 3 つのプロパティーを設定します。

- `availability-service.ejb-container-availability.availability-enabled`
- `availability-service.ejb-container-availability.sfsb-persistence-type`
- `availability-service.ejb-container-availability.sfsb-ha-persistence-type`

たとえば、設定の名前が `config1` の場合は、次のコマンドを使用します。

```
asadmin set --user admin --passwordfile password.txt
--host localhost
--port 4849
config1.availability-service.
ejb-container-availability.availability-enabled="true"
```

```
asadmin set --user admin --passwordfile password.txt --host localhost --port
4849
config1.availability-service.
ejb-container-availability.sfsb-persistence-type="file"

asadmin set --user admin --passwordfile password.txt
--host localhost
--port 4849
config1.availability-service.
ejb-container-availability.sfsb-ha-persistence-type="replicated"
```

可用性の設定

「可用性サービス」の「EJB コンテナの可用性」タブを使用すると、次の設定を変更できます。

HA 持続型: 可用性が有効になっている SFSB のセッション持続性と非活性化メカニズムを指定します。使用できる値は、`file` (ファイルシステム) および `replicated` (ほかのサーバー上のメモリー) です。デフォルト値は `replicated` です。セッションの持続性が必要となる本稼動環境では、`replicated` を使用します。

SFSB 持続型: 可用性が有効になっていない SFSB の非活性化メカニズムを指定します。使用できる値は、`file` (デフォルト) および `replicated` です。

いずれかの持続型を `file` に設定すると、EJB コンテナによって非活性化されたセッション Bean が格納されるファイルシステムの場所が指定されます。ファイルシステムに対するチェックポイントはテストには有効ですが、本稼働環境には役立ちません。ストアプロパティーの設定については、管理コンソールのオンラインヘルプを参照してください。

HA 持続性によって、どのサーバーインスタンスが失敗した場合でも、サーバーインスタンスのクラスタは SFSB 状態を復元できます。HA ストアはまた、非活性化と活性化のストアとしても使用されます。SFSB 状態の持続性を必要とする本稼働環境では、このオプションを使用します。

可用性が無効の場合の SFSB セッションストアの設定

可用性が無効になっている場合、ローカルファイルシステムは SFSB 状態の非活性化に使用されますが、持続性には使用されません。SFSB 状態が格納される場所を変更するには、EJB コンテナのセッション格納位置の設定を変更します。ストアプロパティーの設定については、管理コンソールのオンラインヘルプを参照してください。

個々のアプリケーションまたは EJB モジュールの可用性の設定

配備中に、個々のアプリケーションまたは EJB モジュールの SFSB の可用性を有効にすることができます。

- 管理コンソールを使用して配備している場合は、可用性を有効にするチェックボックスをチェックします。
- `asadmin deploy` または `asadmin deploydir` コマンドを使用して配備している場合は、`--availabilityenabled` オプションを `true` に設定します。詳細については、`deploy(1)` および `deploydir(1)` を参照してください。

個々の Bean の可用性の設定

個々の SFSB について可用性を有効にし、チェックポイントを設定するメソッドを選択するには、`sun-ejb-jar.xml` 配備記述子ファイルを使用します。

高可用性セッション持続性を有効にするには、`ejb` 要素に `availability-enabled="true"` を設定します。SFSB キャッシュのサイズと動作を制御するには、次の要素を使用します。

- `max-cache-size`: キャッシュに保持されるセッション Bean の最大数を指定します。キャッシュがオーバーフローする (Bean の数が `max-cache-size` を超える) 場合、コンテナは一部の Bean を非活性化するか、または Bean の直列化された状態をファイルに書き出します。ファイルを作成するディレクトリは、設定 API を使用して EJB コンテナから取得されます。
- `resize-quantity`
- `cache-idle-timeout-in-seconds`
- `removal-timeout-in-seconds`
- `victim-selection-policy`

`sun-ejb-jar.xml` の詳細については、『Sun GlassFish Communications Server 1.5 Application Deployment Guide』の「The sun-ejb-jar.xml File」を参照してください。

例 6–1　可用性が有効になっている EJB 配備記述子の例

```
<sun-ejb-jar>
    ...
    <enterprise-beans>
        ...
        <ejb availability-enabled="true">
            <ejb-name>MySFSB</ejb-name>
        </ejb>
```

例 6–1　可用性が有効になっている EJB 配備記述子の例　　(続き)

```
        ...
    </enterprise-beans>
</sun-ejb-jar>
```

チェックポイントを設定するメソッドの指定

有効になっている場合、チェックポイント設定は一般に、トランザクションがロールバックする場合でも、Bean がトランザクションを完了したあとに実行されます。Bean の状態に重要な変更をもたらす非トランザクションビジネスメソッドの終了時に、SFSB のオプションのチェックポイント設定を追加で指定するには、`sun-ejb-jar.xml` 配備記述子ファイルの `ejb` 要素にある `checkpoint-at-end-of-method` 要素を使用します。

`checkpoint-at-end-of-method` 要素内の非トランザクションメソッドは、次のいずれかになります。

- SFSB のホームインタフェースで定義された `create()` メソッド。作成の直後に、SFSB の初期状態にチェックポイントを設定する場合に使用します。
- コンテナ管理によるトランザクションのみを使用している SFSB の場合は、トランザクション属性 TX_NOT_SUPPORTED または TX_NEVER でマークされた Bean のリモートインタフェースのメソッド。
- Bean 管理によるトランザクションのみを使用している SFSB の場合は、Bean 管理によるトランザクションが起動もコミットもされないメソッド。

  このリストに記述されているその他のメソッドはすべて無視されます。これらの各メソッドの呼び出しの終了時に、EJB コンテナは SFSB の状態を持続性ストアに保存します。

注 – SFSB がどのトランザクションにも参加しておらず、`checkpoint-at-end-of-method` 要素で明示的に指定されているメソッドがない場合は、この Bean に対して `availability-enabled="true"` が設定されていても、この Bean の状態にチェックポイントは設定されません。

パフォーマンスを向上させるには、メソッドの小さなサブセットを指定します。これらのメソッドは一般に、大量の処理を実行するか、または Bean の状態に重要な変更をもたらします。

例 6–2　メソッドのチェックポイント設定を指定する EJB 配備記述子の例

```
<sun-ejb-jar>
    ...
    <enterprise-beans>
```

例 6–2 メソッドのチェックポイント設定を指定する EJB 配備記述子の例 (続き)

```
        ...
        <ejb availability-enabled="true">
            <ejb-name>ShoppingCartEJB</ejb-name>
            <checkpoint-at-end-of-method>
                <method>
                    <method-name>addToCart</method-name>
                </method>
            </checkpoint-at-end-of-method>
        </ejb>
        ...
    </enterprise-beans>
</sun-ejb-jar>
```

◆ ◆ ◆ 第 7 章

Java Message Service 負荷分散とフェイルオーバー

この章では、Communications Server で使用するために Java Message Service (JMS) の負荷分散とフェイルオーバーを設定する方法について説明します。ここで説明する内容は次のとおりです。

Java Message Service の概要

Java Message Service (JMS) API は、Java EE アプリケーションおよびコンポーネントに対して、メッセージの作成、送信、受信、および読み取りを可能にするメッセージング標準です。この API によって、緩やかに結合され、信頼性が高く、非同期の分散通信が可能となります。Sun Java System Message Queue (MQ) は JMS を実装し、Communications Server と密接に統合されているため、MQ を使用してメッセージ駆動型 Bean (MDB) などのコンポーネントを作成できます。

MQ は「コネクタモジュール」を使用して Communications Server と統合されます。コネクタモジュールはリソースアダプタとしても知られており、Java EE Connector Architecture Specification 1.5 によって定義されています。Communications Server に配備された Java EE コンポーネントは、コネクタモジュールを介して統合された JMS プロバイダを使用して、JMS メッセージをやり取りします。Communications Server で JMS リソースを作成すると、バックグラウンドでコネクタリソースが作成されます。そのようにして、JMS 操作のたびにコネクタランタイムが呼び出され、バックグラウンドで MQ リソースアダプタが使用されます。

Java Message Service は、管理コンソールまたは `asadmin` コマンド行ユーティリティーから管理することができます。

詳細情報

JMS リソースの設定の詳細については、『Sun GlassFish Communications Server 1.5 管理ガイド』の第 4 章「Java Message Service (JMS) リソースの設定」を参照してください。JMS の詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』の第 18 章「Using the Java Message Service」を参照してください。コネクタ (リソースアダプタ) の詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』の第 12 章「Developing Connectors」を参照してください。

Sun GlassFish Message Queue の詳細については、メッセージキュー documentation (`http://docs.sun.com/coll/1343.4`) を参照してください。JMS API の概要については、JMS Web ページ (`http://java.sun.com/products/jms/index.html`)を参照してください。

Java Message Service の設定

「Java Message Service」設定は、Sun GlassFish Communications Server クラスタまたはインスタンスへのすべてのインバウンドおよびアウトバウンド接続に使用できます。次にあげるものを使用して、Java Message Service を設定できます。

- 管理コンソール。関連する設定で「Java メッセージサービス」コンポーネントを開きます。詳細は、『Sun GlassFish Communications Server 1.5 管理ガイド』の第 4 章「Java Message Service (JMS) リソースの設定」.を参照してください。
- `asadmin set` コマンド。次の属性を設定できます。

```
server.jms-service.init-timeout-in-seconds = 60
server.jms-service.type = LOCAL
server.jms-service.start-args =
server.jms-service.default-jms-host = default_JMS_host
server.jms-service.reconnect-interval-in-seconds = 60
server.jms-service.reconnect-attempts = 3
server.jms-service.reconnect-enabled = true
server.jms-service.addresslist-behavior = random
server.jms-service.addresslist-iterations = 3
server.jms-service.mq-scheme = mq
server.jms-service.mq-service = jms
```

次のようなプロパティーも設定できます。

```
server.jms-service.property.instance-name = imqbroker
server.jms-service.property.instance-name-suffix =
server.jms-service.property.append-version = false
```

Java Message Service のすべての属性とプロパティーを一覧表示するには、`asadmin get` コマンドを使用します。`asadmin get` の詳細については、`get(1)`を参照してください。`asadmin set` の詳細については、`set(1)`を参照してください。

JMS 接続ファクトリの設定を使用して、Java Message Service の設定をオーバーライドできます。詳細は、『Sun GlassFish Communications Server 1.5 管理ガイド』の「JMS 接続ファクトリ」を参照してください。

注 – Java Message Service の設定を変更したあとには、Communications Server インスタンスを再起動する必要があります。

JMS 管理の詳細については、『Sun GlassFish Communications Server 1.5 管理ガイド』の第 4 章「Java Message Service (JMS) リソースの設定」を参照してください。

Java Message Service の統合

MQ を Communications Server に統合する方法には、 LOCAL、REMOTE、および EMBEDDED の 3 通りがあります。管理コンソールでは、これらのモードは Java Message Service の「タイプ」属性で表されます。

LOCAL Java Message Service

「タイプ」属性が LOCAL (クラスタインスタンスのデフォルト) の場合、Communications Server はデフォルト JMS ホストとして指定された MQ ブローカを起動および停止します。MQ プロセスはアウトプロセスで (別個の VM 内で) アプリケーションサーバープロセスから起動されます。Communications Server は、追加のポートをブローカに提供します。このポートはブローカによって、RMI レジストリを起動するために使用されます。このポート番号は、そのインスタンス用に設定された JMS ポートの番号に 100 を足した数値になります。たとえば、JMS ポート番号が 37676 である場合、この追加ポートの番号は 37776 になります。

Communications Server インスタンスと Message Queue ブローカの間に 1 対 1 の関係を作成するには、タイプを LOCAL に設定し、各 Communications Server インスタンスに異なるデフォルト JMS ホストを指定します。この作業は、クラスタが Communications Server と MQ のどちらに定義されているかに関係なく行えます。

LOCAL タイプでは、「起動引数」属性を使用して MQ ブローカの起動パラメータを指定します。

REMOTE Java Message Service

「タイプ」属性が REMOTE の場合、MQ ブローカは別個に起動する必要があります。ブローカの起動については、『 Sun GlassFish Message Queue 管理ガイド』を参照してください。

この場合、Communications Server は外部的に設定されたブローカまたはブローカクラスタを使用します。また、MQ ブローカの起動と停止は Communications Server と

は別個に行い、MQ ツールを使用してブローカまたはブローカクラスタを設定および調整する必要があります。REMOTE タイプは Communications Server クラスタに最適です。

REMOTE タイプでは、MQ ツールを使用して MQ ブローカ起動パラメータを指定する必要があります。「起動引数」属性は無視されます。

EMBEDDED Java Message Service

JMS の「タイプ」属性が EMBEDDED の場合、Communications Server と JMS ブローカが同じ VM 内に共存し、JMS サービスはインプロセスで起動され、Communications Server によって管理されます。このモードでは、JMS 操作はネットワークスタックを通して行われ、パフォーマンスの最適化につながります。

JMS ホストリスト

JMS ホストは MQ ブローカを表します。Java Message Service には *JMS* ホストリスト (`AddressList` とも呼ばれる) が含まれており、このリストには Communications Server が使用するすべての JMS ホストが含まれます。

JMS ホストリストには指定された MQ ブローカのホストとポートが取り込まれ、JMS ホスト設定が変更になるたびに更新されます。JMS リソースを作成するかまたは MDB を配備すると、JMS リソースや MDB は JMS ホストリストを継承します。

注 – Sun GlassFish Message Queue ソフトウェアでは、`AddressList` プロパティーは `imqAddressList` と呼ばれています。

デフォルト JMS ホスト

JMS ホストリスト内のホストの 1 つが、`Default_JMS_host` という名前のデフォルト JMS ホストに指定されます。Communications Server インスタンスは、Java Message Service のタイプが LOCAL に設定されている場合に、デフォルト JMS ホストを起動します。

Sun GlassFish Message Queue ソフトウェア内にマルチブローカクラスタを作成してある場合は、デフォルト JMS ホストを削除してから、その Message Queue クラスタのブローカを JMS ホストとして追加します。この場合、デフォルト JMS ホストがJMS ホストリスト内の最初のホストになります。

Communications Server が Message Queue クラスタを使用する場合には、デフォルト JMS ホスト上で Message Queue 固有のコマンドが実行されます。たとえば、3 つのブローカを持つ Message Queue クラスタ用に物理送信先を作成する場合、物理送信先を作成するコマンドはデフォルトの JMS ホスト上で実行されますが、クラスタ内の 3 つのブローカすべてがその物理送信先を使用します。

JMS ホストの作成

追加の JMS ホストを、以下の方法で作成できます。

- 管理コンソールを使用します。関係する設定の「Java メッセージサービス」コンポーネントを開き、「JMS ホスト」コンポーネントを選択してから、「新規」をクリックします。詳細については、管理コンソールのオンラインヘルプを参照してください。
- `asadmin create-jms-host` コマンドを使用します。詳細については、`create-jms-host(1)`を参照してください。

  JMS ホスト設定が変更されるたびに、JMS ホストリストは更新されます。

接続プールとフェイルオーバー

Communications Server は JMS 接続プールとフェイルオーバーをサポートします。Sun GlassFish Communications Server は JMS 接続を自動的にプールします。「アドレスリストの動作」属性が `random` (デフォルト) である場合、Communications Server は主ブローカを JMS ホストリストからランダムに選択します。フェイルオーバーが発生すると、MQ は負荷を別のブローカに透過的に転送し、JMS セマンティクスを保持します。JMS タイプが LOCAL タイプの場合、「アドレスリストの動作」属性のデフォルト値は `priority` です。

接続が失われたときに Communications Server が主ブローカへの再接続を試行するかどうかを指定するには、「再接続」チェックボックスを選択します。再接続を有効に設定した状態で、主ブローカが停止すると、Communications Server は JMS ホストリストにある別のブローカへの再接続を試みます。

「再接続」を有効にする場合には、以下の属性も指定します。

- アドレスリストの動作: 接続を、JMS ホストリスト内のアドレスの順序 (priority) とランダムな順序 (random) のどちらで行うかを指定します。Priority に設定すると、Java Message Service は JMS ホストリストの最初に指定された MQ ブローカに接続を試行し、そのブローカが利用できない場合にのみ別のブローカを使用します。Random に設定すると、Java Message Service は JMS ホストリストから MQ ブローカをランダムに選択します。多数のクライアントが同じ接続ファクトリを使用して接続を試行する場合は、すべてのクライアントが同じアドレスに接続しないようにこの設定を使用します。
- アドレスリストの繰り返し: 接続の確立または再確立のために、JMS ホストリストを介して Java Message Service が試行を繰り返す回数です。値 -1 は試行回数が無制限であることを示します。
- 再接続試行: クライアントランタイムがリストの次のアドレスを試行する前に、JMS ホストリストに指定した各アドレスへの接続 (または再接続) を試行する回数を指定します。値 -1 は、再試行回数が無制限であることを示します。クライアントランタイムは、接続が成功するまで最初のアドレスへの接続を試みます。

- 再接続間隔: 再接続を試行する間隔を秒数で指定します。これは、JMS ホストリストで指定した各アドレスおよびリストのそれ以降のアドレスへの試行に適用されます。間隔が短すぎると、ブローカにリカバリする時間が与えられません。間隔が長すぎると、再接続が許容できない遅延を示す場合があります。

これらの設定は、JMS 接続ファクトリ設定を使用してオーバーライドできます。詳細は、『Sun GlassFish Communications Server 1.5 管理ガイド』の「JMS 接続ファクトリ」を参照してください。

負荷分散されたメッセージのインフロー

メッセージ駆動型 Bean の `sun-ejb-jar.xml` ファイル内の `activation-config-property` 要素を使用して、`jmsra` リソースアダプタの `ActivationSpec` プロパティーを設定できます。メッセージ駆動型 Bean (`EndPointFactory`) が配備されるたびに、コネクタランタイムエンジンがこれらのプロパティーを検出し、それに従ってリソースアダプタ内でそれらのプロパティーを設定します。『Sun GlassFish Communications Server 1.5 Application Deployment Guide』の「activation-config-property」を参照してください。

Communications Server はメッセージが同じ `ClientID` を持つメッセージ駆動型 Bean にランダムに配信されることを透過的に有効にします。`ClientID` は永続的なサブスクライバには必須です。

`ClientID` が設定されない非永続サブスクライバに対しては、同じトピックをサブスクライブする特定のメッセージ駆動型 Bean のすべてのインスタンスは同等であると見なされます。メッセージ駆動型 Bean が Communications Server の複数のインスタンスに配備される場合、メッセージ駆動型 Bean のうちの 1 つだけがメッセージを受信します。複数の異なるメッセージ駆動型 Bean が同じトピックをサブスクライブすると、メッセージ駆動型 Bean ごとに 1 つのインスタンスがメッセージのコピーを受信します。

同じキューを使用する複数のコンシューマをサポートするには、物理送信先の `maxNumActiveConsumers` プロパティーを大きい値に設定します。このプロパティーを設定すると、Sun メッセージキュー ソフトウェアは、プロパティーに設定した数までのメッセージ駆動型 Bean が同じキューからメッセージを消費することを許可します。メッセージはそれらのメッセージ駆動型 Bean にランダムに配信されます。`maxNumActiveConsumers` を `-1` に設定した場合は、コンシューマの数に制限はありません。

ローカル配信が優先されることを保証するには、`addresslist-behavior` を priority に設定します。この設定は、`AddressList` 内の最初のブローカが最初に選択されることを指定します。この最初のブローカは、ローカルで共存する メッセージキュー インスタンスです。このブローカが利用できない場合、`AddressList` 内で列挙されている順序でブローカへの接続試行が行われます。この設定は、クラスタに属する Communications Server インスタンスに対するデフォルトです。

注 – クラスタ化機能は開発者プロファイルでは利用できません。プロファイルの詳細については、『Sun GlassFish Communications Server 1.5 管理ガイド』の「プロファイル」を参照してください。

MQ クラスタと Communications Server の併用

MQ Enterprise Edition は、ブローカクラスタと呼ばれる、相互に接続した複数のブローカインスタンスをサポートします。ブローカクラスタによって、クライアント接続はクラスタ内のすべてのブローカに分散されます。クラスタ化することで、水平方向のスケーラビリティーが提供され、可用性が向上します。

この節では、Sun Java System Message Queue クラスタを使用するために Communications Server を設定する方法を説明します。また、Message Queue クラスタを開始および設定する方法も解説します。

Communications Server および MQ 配備のトポロジの詳細については、『Sun GlassFish Enterprise Server 2.1 配備計画ガイド』の「Message Queue ブローカの配備の計画」を参照してください。

非 HA クラスタの自動クラスタ化

これまでは、この節のあとの手順で説明されているように、「非高可用性」MQ クラスタ (マスターブローカを持つ MQ クラスタ) を管理者が個別に設定する必要がありました。このリリースでは、(REMOTE タイプの) 手動でのプロセスによる MQ クラスタの設定に加えて、Communications Server は「自動クラスタ化」を提供します。これは、ユーザーが Application Server クラスタを作成するときに、(LOCAL タイプの) 共存する非 HA クラスタが自動的に作成されることを意味します。これは MQ クラスタの作成のデフォルトモードになります。たとえば、3 つの Application Server インスタンスを持つ Application Server クラスタを管理者が作成すると、各 Application Server インスタンスは共存するブローカと連動するように設定され、結果的に MQ クラスタが透過的に 3 つの MQ ブローカインスタンスを形成します。最初の Application Server インスタンスの MQ ブローカがマスターブローカに設定されます。ただし、自動クラスタ化には短所もあります。管理者がクラスタにインスタンスを追加する場合、自動的に作成される MQ ブローカインスタンスはクラスタに参加できません。この動作は、インスタンスがクラスタから削除される場合にも適用されます。

▼ MQ クラスタと Communications Server クラスタの併用を有効にする

始める前に　クラスタが REMOTE タイプの場合、次の手順を実行します。クラスタが LOCAL タイプの場合、手順 1 ～ 4 は該当しません。

1. クラスタがまだない場合、クラスタを作成します。
   クラスタの作成については、55 ページの「クラスタを作成する」を参照してください。

2. **MQ** ブローカクラスタを作成します。
   まず、ドメイン管理サーバーによって起動されるブローカを参照するデフォルト JMS ホストを削除してから、MQ ブローカクラスタに 3 つの外部ブローカ (JMS ホスト) を作成します。

   JMS ホストの作成は、管理コンソールまたは `asadmin` コマンド行ユーティリティーのいずれかを使用して行います。

   `asadmin` を使用する場合は、たとえば次のコマンドを実行します。

   ```
   asadmin delete-jms-host --target cluster1 default_JMS_host
   asadmin create-jms-host --target cluster1
        --mqhost myhost1 --mqport 6769
        --mquser admin --mqpassword admin broker1
   asadmin create-jms-host --target cluster1
        --mqhost myhost2 --mqport 6770
        --mquser admin --mqpassword admin broker2
   asadmin create-jms-host --target cluster1
        --mqhost myhost3 --mqport 6771
        --mquser admin --mqpassword admin broker3
   ```

   管理コンソールを使用してホストを作成するには、次のようにします。

   a. 「**JMS** ホスト」ノードに移動します (「設定」> *config-name* >「**Java** メッセージサービス」>「**JMS** ホスト」)。

   b. デフォルトのブローカ **(default_JMS_host)** を削除します。
      そのブローカの横にあるチェックボックスを選択して、「削除」をクリックします。

   c. 「新規」をクリックして、各 **JMS** ホストを作成し、それぞれにプロパティー値を入力します。
      ホスト名、DNS 名または IP アドレス、ポート番号、管理ユーザー名、パスワードの値を指定します。

3 マスター **MQ** ブローカとほかの **MQ** ブローカを起動します。

JMS ホストマシン上で起動する 3 つの外部ブローカに加えて、任意のマシン上で 1 つのマスターブローカを起動します。このマスターブローカは、ブローカクラスタの一部である必要はありません。次に例を示します。

```
/usr/bin/imqbrokerd -tty -name brokerm -port 6772
 -cluster myhost1:6769,myhost2:6770,myhost2:6772,myhost3:6771
 -D"imq.cluster.masterbroker=myhost2:6772"
```

4 クラスタ内のインスタンスを起動します。

5 クラスタ上に **JMS** リソースを作成します。

a. **JMS** 物理送信先を作成します。

たとえば、次の `asadmin` を使用します。

```
asadmin create-jmsdest --desttype queue --target cluster1 MyQueue
asadmin create-jmsdest --desttype queue --target cluster1 MyQueue1
```

管理コンソールを使用する場合は、次のようにします。

i. 「物理送信先」ページに移動します (「設定」> *config-name* >「**Java** メッセージサービス」>「物理送信先」)。

ii. 「新規」をクリックして、各 **JMS** 物理送信先を作成します。

iii. 各送信先に対して名前とタイプ (キュー) を入力します。

b. **JMS** 接続ファクトリを作成します。

たとえば、次の `asadmin` を使用します。

```
asadmin create-jms-resource --target cluster1
--restype javax.jms.QueueConnectionFactory jms/MyQcf
asadmin create-jms-resource --target cluster1
--restype javax.jms.QueueConnectionFactory jms/MyQcf1
```

管理コンソールを使用する場合は、次のようにします。

i. 「**JMS** 接続ファクトリ」ページに移動します (「リソース」>「**JMS** リソース」>「接続ファクトリ」)。

ii. それぞれの接続ファクトリを作成するために、「新規」をクリックします。
「新しい JMS 接続ファクトリ」ページが開きます。

iii. 各接続ファクトリについて、「**JNDI** 名」(`jms/MyQcf` など) を入力し、「リソースタイプ」に `javax.jms.QueueConnectionFactory` を指定します。

iv. ページ最下部にリストされた利用可能なターゲットからクラスタを選択して、「追加」をクリックします。

v. 「了解」をクリックして、接続ファクトリを作成します。

c. **JMS** 送信先リソースを作成します。

たとえば、次の `asadmin` を使用します。

```
asadmin create-jms-resource --target cluster1
     --restype javax.jms.Queue
     --property imqDestinationName=MyQueue jms/MyQueue
asadmin create-jms-resource --target cluster1
     --restype javax.jms.Queue
     --property imqDestinationName=MyQueue1 jms/MyQueue1
```

管理コンソールを使用する場合は、次のようにします。

i. 「**JMS** 送信先リソース」ページに移動します (「リソース」>「**JMS** リソース」>「送信先リソース」)。

ii. それぞれの送信先リソースを作成するために、「新規」をクリックします。

「新しい JMS 送信先リソース」ページが開きます。

iii. 各送信先リソースについて、「**JNDI** 名」(`jms/MyQueue` など) を入力し、「リソースタイプ」に `javax.jms.Queue` を指定します。

iv. ページ最下部にリストされた利用可能なターゲットからクラスタを選択して、「追加」をクリックします。

v. 「了解」をクリックして、送信先リソースを作成します。

6 – `retrieve` オプションを指定して、アプリケーションをアプリケーションクライアント用に配備します。次に例を示します。

```
asadmin deploy --target cluster1
--retrieve /opt/work/MQapp/mdb-simple3.ear
```

7 アプリケーションにアクセスして、期待どおりの動作をするかテストします。

8 **Communications Server** をデフォルトの **JMS** 設定に戻す場合は、作成した **JMS** ホストを削除して、デフォルトを作成し直します。次に例を示します。

```
asadmin delete-jms-host --target cluster1 broker1
asadmin delete-jms-host --target cluster1 broker2
asadmin delete-jms-host --target cluster1 broker3
asadmin create-jms-host --target cluster1
 --mqhost myhost1 --mqport 7676
```

```
--mquser admin --mqpassword admin
default_JMS_host
```

管理コンソールを使用して、これに相当する操作を実行することもできます。

注意事項 問題が起きた場合は、次の点を考慮してください。

- サーバーログファイルを表示します。このファイルの場所は、*as-install-dir*/nodeagents/*node-agent-name*/*instance-name*/logs/server.log です。MQ ブローカがメッセージに応答しないとログファイルに記録されている場合は、ブローカを停止してから再起動します。
- ブローカログを表示します。このログの場所は、*as-install-dir*/nodeagents/*node-agent-name*/*instance-name*/imq/*imq-instance-name*/log/log.tx です。
- リモートの JMS タイプについては、必ず、MQ ブローカを最初に起動してからインスタンスを起動するようにしてください。
- すべての MQ ブローカが停止した場合、Java Message Service のデフォルト値では、Communications Server の停止または起動までに 30 分かかります。Java Message Service の値を調整して、このタイムアウトを許容できる値にしてください。次に例を示します。

```
asadmin set --user admin --password administrator
cluster1.jms-service.reconnect-interval-in-seconds=5
```

◆ ◆ ◆ 第 8 章

RMI-IIOP 負荷分散とフェイルオーバー

この章では、RMI-IIOP 上のリモート EJB 参照と JNDI オブジェクトに高可用性 (HA) 機能を使用する方法について説明します。

概要

RMI-IIOP 負荷分散では、IIOP クライアント要求が別のサーバーインスタンスまたはネームサーバーに分散されます。目標は、負荷をクラスタ間に均等に拡散して、スケーラビリティーを実現することです。また、IIOP 負荷分散を EJB のクラスタリングおよび可用性と結合すれば、EJB フェイルオーバーも実現されます。

クライアントがオブジェクトに対して JNDI 検索を実行すると、ネームサービスは、特定のサーバーインスタンスに関連付けられた `InitialContext` (IC) オブジェクトを作成します。それ以降、その IC オブジェクトを使用して作成された検索要求はすべて、同じサーバーインスタンスに送信されます。その `InitialContext` を使用して検索された `EJBHome` オブジェクトはすべて、同じターゲットサーバーにホストされます。また、それ以降に取得された Bean 参照もすべて、同じターゲットホスト上に作成されます。`InitialContext` オブジェクトの作成時に、ライブターゲットサーバーのリストがすべてのクライアントによってランダムに選択されるため、これにより負荷分散が効果的に実現されます。ターゲットサーバーインスタンスが停止すると、検索または EJB メソッド呼び出しは、別のサーバーインスタンスに処理が引き継がれます。

RMI-IIOP 負荷分散とフェイルオーバーは、透過的に発生します。アプリケーションの配備中に、特別な手順は必要ありません。Communications Server の IIOP 負荷分散およびフェイルオーバーは、クラスタの動的再設定をサポートしています。アプリケーションクライアントが配備される Communications Server インスタンスがクラスタに参加する場合、Communications Server は、クラスタ内で現在アクティブなすべての IIOP 端点を自動的に検出します。したがって、新しいインスタンスがクラスタ

に追加された、またはインスタンスがクラスタから削除された場合に、端点のリストを手動で更新する必要はありません。ただし、端点の 1 つで障害が発生した場合に備えて、クライアントにはブートストラップ目的で少なくとも 2 つの端点を指定しておくことをお勧めします。

要件

Sun GlassFish Communications Server は、RMI-IIOP 上で、リモート EJB 参照と `NameService` オブジェクトの高可用性を提供します。それには、次のすべての要件を満たしている必要があります。

- 配備に、2 つ以上のインスタンスのクラスタが含まれていること。
- Java EE アプリケーションが、負荷分散にかかわるすべてのインスタンスとクラスタに対して配備されていること。
- RMI-IIOP クライアントアプリケーションで、負荷分散が有効であること。

Communications Server は、Application Client Container (ACC) で動作している Java アプリケーションに対する負荷分散をサポートしています。123 ページの「RMI-IIOP 負荷分散とフェイルオーバーの設定」を参照してください。

注 – Communications Server は、SSL (Secure Socket Layer) 上の RMI-IIOP 負荷分散とフェイルオーバーをサポートしていません。

アルゴリズム

Communications Server は、ランダム化とラウンドロビンのアルゴリズムを使用して、RMI-IIOP 負荷分散とフェイルオーバーを実現しています。

RMI-IIOP クライアントは最初に新しい `InitialContext` オブジェクトを作成すると、そのクライアントで利用可能な Communications Server IIOP 端点のリストが、ランダムに選ばれます。その `InitialContext` オブジェクトに対して、ロードバランサは、ランダムに選択されたリストの最初の端点に検索要求とほかの `InitialContext` 操作を命令します。最初の端点が利用できない場合、リストの 2 番目の端点が使用され、以下同様です。

クライアントが続けて新しい `InitialContext` オブジェクトを作成するたびに、端点リストがローテーションし、異なる IIOP 端点が `InitialContext` 操作で使われます。

`InitialContext` オブジェクトによって確保される参照から Beans を入手または作成する場合、それらの Beans は、`InitialContext` オブジェクトに割り当てられた IIOP 端点を処理する Communications Server インスタンスで作成されます。それらの Beans に対する参照には、クラスタ内のすべての Communications Server インスタンスの IIOP 端点アドレスが含まれます。

プライマリ端点は、Bean の検索または作成に使用される InitialContext 端点に対応する Bean 端点です。クラスタ内のほかの IIOP 端点は、代替端点として指定されています。Bean のプライマリ端点が利用できなくなると、その Bean での追加の要求は、代替端点の 1 つにフェイルオーバーされます。

RMI-IIOP 負荷分散とフェイルオーバーは、ACC で動作しているアプリケーションとともに動作するように設定できます。

RMI-IIOP 負荷分散とフェイルオーバーの設定

RMI-IIOP 負荷分散とフェイルオーバーは、アプリケーションクライアントコンテナ (ACC) で動作しているアプリケーション用に設定できます。重み付きラウンドロビンによる負荷分散もサポートされています。

▼ アプリケーションクライアントコンテナ用に RMI-IIOP 負荷分散を設定する

この手順は、アプリケーションクライアントコンテナ (ACC) とともに RMI-IIOP 負荷分散とフェイルオーバーを使用するために必要な手順の概要を示しています。ACC の詳細については、『Sun GlassFish Communications Server 1.5 Developer's Guide』の「Developing Clients Using the ACC」を参照してください。

1. *install_dir*/bin ディレクトリに移動します。
2. package-appclient を実行します。
   このユーティリティーによって、appclient.jar ファイルが生成されます。package-appclient の詳細については、package-appclient(1M) を参照してください。
3. appclient.jar ファイルを、クライアントを実行するマシンにコピーして展開します。
4. asenv.conf または asenv.bat パス変数を編集して、そのマシン上の正しいディレクトリ値を参照するようにします。
   このファイルは、*appclient-install-dir*/config/ に格納されています。
   更新するパス変数の一覧については、package-appclient(1M)を参照してください。
5. 必要に応じて、appclient スクリプト実行ファイルを作成します。
   たとえば、UNIX では chmod 700 を使用します。

6 クラスタ内の少なくとも 2 つのインスタンスの IIOP リスナーポート番号を調べます。手順 7 で IIOP リスナーを端点として指定します。

各インスタンスに対して、次のようにして IIOP リスナーポートを取得します。

a. 管理コンソールのツリーコンポーネントで、「クラスタ」ノードを展開します。

b. クラスタを展開します。

c. クラスタ内のインスタンスを選択します。

d. 右の区画で、「プロパティー」タブをクリックします。

e. インスタンスに対する IIOP リスナーポートを記録します。

7 `sun-acc.xml` ファイルに、少なくとも 2 つの `target-server` 要素を追加します。

注 – クラスタ化機能は開発者プロファイルでは利用できません。プロファイルの詳細については、『Sun GlassFish Communications Server 1.5 管理ガイド』の「プロファイル」を参照してください。

手順 6 で取得した端点を使用します。

アプリケーションクライアントが配備される Communications Server インスタンスがクラスタに参加する場合、ACC は、クラスタ内で現在アクティブなすべての IIOP 端点を自動的に検出します。ただし、端点の 1 つで障害が発生した場合に備えて、クライアントにはブートストラップ目的で少なくとも 2 つの端点を指定しておくことをお勧めします。

`target-server` 要素は、負荷分散に使用される 1 つまたは複数の IIOP 端点を指定します。`address` 属性は IPv4 アドレスまたはホスト名であり、`port` 属性はポート番号を指定します。『Sun GlassFish Communications Server 1.5 Application Deployment Guide』の「client-container」を参照してください。

`target-server` 要素を使用する代わりに、`endpoints` プロパティーを次のように使用できます。

```
jvmarg value = "-Dcom.sun.appserv.iiop.endpoints=host1:port1,host2:port2,..."
```

8 重み付きラウンドロビンによる負荷分散が必要な場合、次の手順を実行します。

a. 各サーバーインスタンスの負荷分散の重みを設定します。

```
asadmin set instance-name.lb-weight=weight
```

b. sun-acc.xml で、**ACC** の com.sun.appserv.iiop.loadbalancingpolicy プロパティーを ic-based-weighted に設定します。

```
...
<client-container send-password="true">
  <property name="com.sun.appserv.iiop.loadbalancingpolicy" value="ic-based-weighed"/>
...
```

9 --retrieve オプションを使用してクライアントアプリケーションを配備し、クライアントの **JAR** ファイルを取得します。

クライアントの JAR ファイルはクライアントマシンに置いたままにします。

次に例を示します。

```
asadmin deploy --user admin --passwordfile pw.txt --retrieve /my_dir myapp
```

10 アプリケーションクライアントを、次のように実行します。

appclient -client *clientjar* -name *appname*

例 8–1 RMI-IIOP 重み付きラウンドロビン負荷分散に使用する負荷分散の重みの設定

この例では、3 つのインスタンスを含むクラスタでの負荷分散の重みを、次の表に示すように設定します。

インスタンス名	負荷分散の重み
i1	100
i2	200
i3	300

これらの負荷分散の重みを設定するための一連のコマンドは、次のようになります。

```
asadmin set i1.lb-weight=100
asadmin set i2.lb-weight=200
asadmin set i3.lb-weight=300
```

次の手順 フェイルオーバーをテストするには、クラスタ内の 1 つのインスタンスを停止し、アプリケーションが正常に動作するかどうかを調べます。また、クライアントアプリケーション内にブレークポイント (またはスリープ) を設定することもできます。

負荷分散をテストするには、複数のクライアントを使用し、すべての端点にわたって負荷がどのように分散されるかを調べます。

索引

有