For Review Purposes Only

Sun Java System Directory Server Enterprise Edition 6.3 配備計画ガイド

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

Part No: 820–4820
2008 年 4 月

080407@19860

目次

図目次

はじめに

『Sun Java System Directory Server Enterprise Edition 6.3 配備計画ガイド』には、ディレクトリサービスの配備を計画するために必要な情報が含まれています。このガイドでは、データ型、アクセス制御、サイズ決定などの問題について行う必要がある初期決定について説明します。また、対象の企業や組織に特有の要件に合わせて使用できる、概略レベルのサンプルおよび方針も提示します。

対象読者

このガイドは主に、ディレクトリサービス配備の分析と設計を担当する配備アーキテクトやビジネス計画担当者を対象に記述されています。エンタープライズアプリケーションの設計と実装を担当するシステムインテグレータやその他の人々にとっても役立つ情報も含まれています。

お読みになる前に

このガイドでは、読者が LDAP ディレクトリサーバーの基本的な概念を理解しており、次のドキュメントにあらかじめ目を通していることを前提にしています。

- 『Sun Java System Directory Server Enterprise Edition 6.3 Release Notes 』
- 『Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide 』

マニュアルの構成

このガイドは、配備計画のさまざまなフェーズを記述するソリューションライフサイクルにベースにしています。

第 1 部では、Directory Server Enterprise Edition の概要を示し、配備の計画に関係する手順 (ソリューションライフサイクル) について説明します。

第 2 部では、論理配備アーキテクチャーの作成を開始する前に行う必要がある技術要件分析について説明します。技術要件分析を行うには、ビジネス領域、ビジネスの目的、および配下のシステム技術についての理解が必要です。

第 3 部では、Directory Server Enterprise Edition 配備の論理アーキテクチャーを作成する方法について説明します。また、Directory Server Enterprise Edition の一般的な配備シナリオに基づくサンプルの論理アーキテクチャーも示します。

第 4 部では、Solaris オペレーティングシステムでの LDAP ベースネームサービスの利用、Identity Synchronization for Windows、仮想ディレクトリの配備など、特殊な配備に関するトピックを扱います。

Directory Server Enterprise Edition マニュアルセット

この Directory Server Enterprise Edition マニュアルセットでは、Sun Java System Directory Server Enterprise Edition を使用してディレクトリサービスを評価、設計、配備、および管理する方法について説明します。Directory Server Enterprise Edition 用のクライアントアプリケーションを開発する方法も示します。Directory Server Enterprise Edition のマニュアルセットの英語版は http://docs.sun.com/coll/1224.4 から入手できます。また、一部のマニュアルは、翻訳されており、http://docs.sun.com/coll/1757.1 から入手できます。

Directory Server Enterprise Edition について理解を深めるには、次の表に示すドキュメントを順番に参照してください。

表 P–1　Directory Server Enterprise Edition のマニュアル

マニュアルタイトル	目次
『Sun Java System Directory Server Enterprise Edition 6.3 Release Notes 』	既知の問題を含め、Directory Server Enterprise Edition についての最新情報を提供しています。
『Sun Java System Directory Server Enterprise Edition 6.3 Documentation Center 』	マニュアルセットの重要な領域へのリンクを提供しています。
『Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide 』	このリリースの重要な機能を紹介します。これらの機能の仕組みや提供される利点を、単独システムに実装可能な架空の配備のコンテキストに沿って例示します。
『Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide 』	Directory Server Enterprise Edition をベースとする、可用性と拡張性に優れたディレクトリサービスを計画および設計する方法について説明します。配備の計画および設計の基本的な概念および原則を提示します。ソリューションのライフサイクルについて検討し、Directory Server Enterprise Edition ベースのソリューションを計画するために使用する概略レベルのサンプルおよび戦略を提供します。

表 P–1 Directory Server Enterprise Edition のマニュアル (続き)

マニュアルタイトル	目次
『Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide』	Directory Server Enterprise Edition ソフトウェアのインストール方法について説明します。インストールするコンポーネントを選択する方法、インストール後にそれらのコンポーネントを設定する方法、および設定したコンポーネントが正しく機能することを検証する方法を示します。 Directory Editor のインストール手順については、http://docs.sun.com/coll/DirEdit_05q1 を参照してください。 Directory Editor をインストールする前に、『Sun Java System Directory Server Enterprise Edition 6.3 Release Notes』の Directory Editor についての情報を必ずお読みください。
『Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide』	Directory Server、Directory Proxy Server、および Identity Synchronization for Windows の以前のバージョンから移行するための手順を示します。
『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』	Directory Server Enterprise Edition をコマンド行から管理するための手順を示します。 Directory Service Control Center (DSCC) を使用して Directory Server Enterprise Edition を管理する際のヒントおよび手順については、DSCC のオンラインヘルプを参照してください。 Directory Editor の管理手順については、http://docs.sun.com/coll/DirEdit_05q1 を参照してください。 Identity Synchronization for Windows のインストールと設定については、Part II,「Installing Identity Synchronization for Windows,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide』を参照してください。
『Sun Java System Directory Server Enterprise Edition 6.3 Developer's Guide』	Directory Server Enterprise Edition の一部として提供されているツールと API でディレクトリクライアントアプリケーションを開発する方法を説明しています。
『Sun Java System Directory Server Enterprise Edition 6.3 Reference』	Directory Server Enterprise Edition の技術および概念の基礎を紹介します。コンポーネント、アーキテクチャー、プロセス、および機能について説明しています。開発者 API への参照も示しています。
『Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference』	Directory Server Enterprise Edition を通じて利用可能なコマンド行ツール、スキーマオブジェクト、およびその他の公開インタフェースについて説明しています。このドキュメントの個別の節を、オンラインマニュアルページとしてインストールすることができます。
『Sun Java System Directory Server Enterprise Edition 6.3 Troubleshooting Guide』	さまざまなツールを使用して問題の範囲を定義し、データを収集し、問題領域をトラブルシューティングするための情報を提供しています。
『Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide』	Identity Synchronization for Windows の計画と配備に関する一般的なガイドラインやベストプラクティスを示しています。

関連資料

SLAMD 分散負荷生成エンジンとは、ネットワークベースのアプリケーションの負荷テストを実行し、パフォーマンスを分析するために設計された Java™ アプリケーションです。SLAMD は当初 Sun Microsystems, Inc. によって、LDAP ディレクトリサーバーのパフォーマンスをベンチマークおよび分析する目的で開発されました。SLAMD は、OSI が承認したオープンソースライセンスである Sun Public License のもとでオープンソースアプリケーションとして公開されています。SLAMD についての情報を入手するには、`http://www.slamd.com/` を参照してください。SLAMD は java.net プロジェクトとしても公開されています。`https://slamd.dev.java.net/` を参照してください。

Java Naming and Directory Interface (JNDI) 技術は、LDAP および DSML v2 を利用した、Java アプリケーションからのディレクトリサーバーへのアクセスをサポートします。JNDI の詳細については、`http://java.sun.com/products/jndi/` を参照してください。『JNDI チュートリアル』には、JNDI の使用方法についての詳しい説明およびサンプルを収録しています。このチュートリアルは `http://java.sun.com/products/jndi/tutorial/` から入手できます。

Directory Server Enterprise Edition のライセンス形態には、スタンドアロン製品、Sun Java Enterprise System のコンポーネント、Sun Java Identity Management Suite などの Sun 製品群の一部、または Sun からのほかのソフトウェア製品へのアドオンパッケージがあります。Java Enterprise System は、ネットワークまたはインターネット環境で分散配備されるエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーです。Directory Server Enterprise Edition が Java Enterprise System のコンポーネントとしてライセンスされる場合、`http://docs.sun.com/coll/1286.3` から入手可能なシステムマニュアルに目を通してください。

Identity Synchronization for Windows は、Message Queue を制限されたライセンスで使用します。Message Queue のマニュアルは `http://docs.sun.com/coll/1307.2` から入手できます。

Identity Synchronization for Windows は、Microsoft Windows のパスワードポリシーを管理するための製品です。

- Windows Server 2003 のパスワードポリシーについての情報は、Microsoft TechNet Web サイトで公開されています。
- Microsoft 証明書サービスのエンタープライズルート認証局に関する情報は、Microsoft サポートオンライン Web サイトで公開されています。
- Microsoft システムでの LDAP over SSL の設定に関する情報は、Microsoft サポートオンライン Web サイトで公開されています。

再配布可能なファイル

Directory Server Enterprise Edition では、お客様による再配布が可能なファイルは提供されません。

デフォルトのパスとコマンドの場所

この節では、マニュアルで使用するデフォルトのパスについて説明し、オペレーティングシステムや配備タイプによって異なるコマンドの場所を示します。

デフォルトのパス

次の表では、このドキュメントで使用するデフォルトのパスについて説明します。インストールされるファイルの詳細な説明については、次の製品マニュアルを参照してください。

- Chapter 14, 「Directory Server File Reference,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』
- Chapter 25, 「Directory Proxy Server File Reference,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』
- Appendix A, 「Directory Server Resource Kit File Reference,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』

表 P–2 デフォルトのパス

プレースホルダ	説明	デフォルト値
install-path	Directory Server Enterprise Edition ソフトウェアのベースインストールディレクトリを表します。 ソフトウェアは、このベース *install-path* の下位のディレクトリにインストールされます。たとえば、Directory Server ソフトウェアは *install-path*`/ds6/` にインストールされます。	`dsee_deploy`(1M) を使用して ZIP 形式の配布パッケージからインストールするとき、デフォルトの *install-path* は現在のディレクトリです。*install-path* は、`dsee_deploy` コマンドの `-i` オプションを使用して設定できます。 Java Enterprise System インストーラを使用する場合など、ネイティブパッケージ配布からインストールする場合、デフォルトの *install-path* は次のいずれかの場所になります。 ■ Solaris システム - `/opt/SUNWdsee/` ■ Red Hat システム - `/opt/sun/` ■ Windows システム - `C:\Program Files\Sun\JavaES5\DSEE`

表 P–2 デフォルトのパス (続き)

プレースホルダ	説明	デフォルト値
instance-path	Directory Server または Directory Proxy Server のインスタンスのフルパスを表します。 このマニュアルでは、Directory Server には `/local/ds/` を、Directory Proxy Server には `/local/dps/` を使用します。	デフォルトのパスはありません。ただしインスタンスのパスは、常にローカルファイルシステム上に存在します。 推奨されるディレクトリは以下のとおりです。 `/var` (Solaris システム) `/global` (Sun Cluster を使用する場合)
serverroot	Identity Synchronization for Windows のインストール先の親ディレクトリを表します	インストールごとに異なります。Directory Server では、*serverroot* の概念が存在しなくなったことに注意してください。
`isw-`*hostname*	Identity Synchronization for Windows インスタンスのディレクトリを表します	インストールごとに異なります
/path/to/`cert8.db`	Identity Synchronization for Windows におけるクライアントの証明書データベースのデフォルトのパスおよびファイル名を表します	*current-working-dir*`/cert8.db`
serverroot`/isw-`*hostname*`/logs/`	システムマネージャー、各コネクタ、および Central Logger のログを Identity Synchronization for Windows がローカルに保存する場所のデフォルトパスを表します	インストールごとに異なります
serverroot`/isw-`*hostname*`/logs/central/`	Identity Synchronization for Windows セントラルログのデフォルトパスを表します	インストールごとに異なります

コマンドの場所

次の表は、Directory Server Enterprise Edition のマニュアルで使用されるコマンドの場所の一覧です。これらの各コマンドの詳細については、それぞれのマニュアルページを参照してください。

表 P–3 コマンドの場所

コマンド	Java ES ネイティブパッケージ配布	ZIP 形式の配布パッケージ
cacaoadm	Solaris - /usr/sbin/cacaoadm	Solaris - *install-path*/dsee6/cacao_2/usr/sbin/cacaoadm
	Red Hat - /opt/sun/cacao/bin/cacaoadm	Red Hat、HP-UX - *install-path*/dsee6/cacao_2/cacao/bin/cacaoadm
	Windows - *install-path*\share\cacao_2\bin\cacaoadm.bat	Windows - *install-path*\dsee6\cacao_2\bin\cacaoadm.bat
certutil	Solaris - /usr/sfw/bin/certutil	*install-path*/dsee6/bin/certutil
	Red Hat - /opt/sun/private/bin/certutil	
dpadm(1M)	*install-path*/dps6/bin/dpadm	*install-path*/dps6/bin/dpadm
dpconf(1M)	*install-path*/dps6/bin/dpconf	*install-path*/dps6/bin/dpconf
dsadm(1M)	*install-path*/ds6/bin/dsadm	*install-path*/ds6/bin/dsadm
dsccmon(1M)	*install-path*/dscc6/bin/dsccmon	*install-path*/dscc6/bin/dsccmon
dsccreg(1M)	*install-path*/dscc6/bin/dsccreg	*install-path*/dscc6/bin/dsccreg
dsccsetup(1M)	*install-path*/dscc6/bin/dsccsetup	*install-path*/dscc6/bin/dsccsetup
dsconf(1M)	*install-path*/ds6/bin/dsconf	*install-path*/ds6/bin/dsconf
dsee_deploy(1M)	提供されていません	*install-path*/dsee6/bin/dsee_deploy
dsmig(1M)	*install-path*/ds6/bin/dsmig	*install-path*/ds6/bin/dsmig
entrycmp(1)	*install-path*/ds6/bin/entrycmp	*install-path*/ds6/bin/entrycmp
fildif(1)	*install-path*/ds6/bin/fildif	*install-path*/ds6/bin/fildif
idsktune(1M)	提供されていません	ZIP 形式の配布パッケージを解凍したディレクトリ
insync(1)	*install-path*/ds6/bin/insync	*install-path*/ds6/bin/insync
ns-accountstatus(1M)	*install-path*/ds6/bin/ns-accountstatus	*install-path*/ds6/bin/ns-accountstatus
ns-activate(1M)	*install-path*/ds6/bin/ns-activate	*install-path*/ds6/bin/ns-activate
ns-inactivate(1M)	*install-path*/ds6/bin/ns-inactivate	*install-path*/ds6/bin/ns-inactivate
repldisc(1)	*install-path*/ds6/bin/repldisc	*install-path*/ds6/bin/repldisc

表 P–3 コマンドの場所 (続き)

コマンド	Java ES ネイティブパッケージ配布	ZIP 形式の配布パッケージ
schema_push(1M)	*install-path*/ds6/bin/schema_push	*install-path*/ds6/bin/schema_push
smcwebserver	Solaris、Linux - /usr/sbin/smcwebserver Windows - *install-path*\share\webconsole\bin\smcwebserver	このコマンドは、ネイティブパッケージ配布を使用してインストールされる DSCC のみに関係します。
wcadmin	Solaris、Linux - /usr/sbin/wcadmin Windows - *install-path*\share\webconsole\bin\wcadmin	このコマンドは、ネイティブパッケージ配布を使用してインストールされる DSCC のみに関係します。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–4 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% you have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	machine_name% **su** Password:
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。

表 P–4 表記上の規則 (続き)

字体または記号	意味	例
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

コード例は次のように表示されます。

- C シェル

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

- C シェルのスーパーユーザー

  ```
  machine_name# command y|n [filename]
  ```

- Bourne シェルおよび Korn シェル

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

- Bourne シェルおよび Korn シェルのスーパーユーザー

  ```
  # command y|n [filename]
  ```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

コマンド例のシェルプロンプト

次の表は、デフォルトのシステムプロンプトおよびスーパーユーザープロンプトを示しています。

表 P–5　シェルプロンプトについて

シェル	プロンプト
UNIX および Linux システムの C シェル	`machine_name%`
UNIX および Linux システムの C シェルのスーパーユーザー	`machine_name#`
UNIX および Linux システムの Bourne シェルおよび Korn シェル	`$`
UNIX および Linux システムの Bourne シェルおよび Korn シェルのスーパーユーザー	`#`
Microsoft Windows のコマンド行	`C:\`

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–6　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–6 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 のサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

パート I

Directory Server Enterprise Edition の配備計画の概要

この第 1 部では、Directory Server Enterprise Edition の概要を示し、配備 (ソリューションライフサイクル) の計画に必要な手順を説明します。次の章で構成されています。

- 第 1 章では、配備計画プロセスについて説明します。
- 第 2 章では、ビジネス要件について説明します。

配備計画プロセスの詳細は、『Sun Java Enterprise System 配備計画ガイド』を参照してください。

◆ ◆ ◆ 第 1 章

1 Directory Server Enterprise Edition 配備計画の概要

この章では、Sun Java System Directory Server Enterprise Edition の概要を示し、配備計画プロセスについて概略レベルでの説明を行います。

この章の内容は次のとおりです。

Directory Server Enterprise Edition について

Directory Server Enterprise Edition は、アイデンティティーデータを格納および管理するための、セキュリティー、高可用性、拡張性を備えたディレクトリサービスを提供します。Directory Server Enterprise Edition は企業のアイデンティティーインフラストラクチャーの基礎を形成します。この製品により、ミッションクリティカルなエンタープライズアプリケーションや大規模エクストラネットアプリケーションから、整合性と信頼性のあるアイデンティティーデータにアクセスできるようになります。

Directory Server Enterprise Edition は、アイデンティティープロファイル、アクセス特権、アプリケーションとネットワークのリソース情報を格納および管理するための集中リポジトリを提供します。Directory Server Enterprise Edition は、マルチプラットフォーム環境へのスムーズな統合が可能です。また、Microsoft Active Directory との間で、パスワード、ユーザー、およびグループをオンデマンドで安全に同期できます。

Directory Server Enterprise Edition 以前は、Sun はこれらの機能を Directory Server、Directory Proxy Server、Directory Server Resource Kit、および Identity Synchronization for Windows の 4 つの独立した製品で提供していました。これらを含む各製品は、現在では、1 つの包括的な統合型ソリューションのコンポーネントとなっています。

堅牢なディレクトリサービスのためのサービス品質要件

企業内でユーザーやアプリケーションの数が増加するにつれ、堅牢なディレクトリサービスはますます必要不可欠なものとなります。Directory Server Enterprise Edition では、次に示すサービス品質要件を提供することにより、急速に変化および拡大する企業が直面する課題を解決します。

- 可用性: システムのリソースやサービスにエンドユーザーがアクセスできる時間の指標であり、しばしばシステムの「稼働時間」として表現されます。
- 拡張性: 配備済みのシステムに、容量やユーザーを段階的に追加できる能力のことです。拡張は通常、システムにリソースを追加することによって行いますが、その際に配備アーキテクチャーの変更を必要としないことが理想的です。
- セキュリティー: システムとそのユーザーの完全性を担保する各種要因の複合的な組み合わせです。セキュリティーを構成する要素には、ユーザーの認証と承認、データのセキュリティー、配備済みシステムへのセキュリティー保護されたアクセスなどがあります。
- 相互運用性: システムをほかのシステムと組み合わせて容易に運用できることを指します。
- 保守容易性: 配備済みシステムを容易に保守できることを指します。保守タスクには、システムの監視、発生した問題の修復、ハードウェアおよびソフトウェアコンポーネントのアップグレードなどが含まれます。

この章では、Directory Server Enterprise Edition の各種コンポーネントがどのようにしてサービス品質要件を満たすかについて簡単に説明します。個々の要件については、このマニュアルの次章以降で詳しく説明します。

Directory Server Enterprise Edition のコンポーネントとその機能

Directory Server Enterprise Edition には、次に示す独立したコンポーネントが含まれます。

- 29 ページの「Directory Server」
- 31 ページの「Directory Proxy Server」
- 32 ページの「Identity Synchronization for Windows」
- 33 ページの「Directory Editor」
- 33 ページの「Directory Server Resource Kit」

これらの各コンポーネントは、前出のサービス品質要件の 1 つまたは複数に関係します。この節では、これらのコンポーネントについて説明し、堅牢なディレクトリサービスを実現するためのコンポーネントの組み合わせ例を示します。

Directory Server

Directory Server は、アイデンティティー情報を格納するための、拡張性がありパフォーマンスの高いデータストアを提供します。Directory Server は、標準規格ベースのアクセスのために、Lightweight Directory Access Protocol (LDAP) v3 および Directory Service Markup Language (DSML) v2 をネイティブでサポートします。HTTP または SOAP (Simple Object Access Protocol) 経由で LDAP および DSML を利用することにより、ネットワーク上のあらゆる場所のクライアントが、ディレクトリデータオブジェクトを安全に検索および更新できます。クライアントは、ほかのアプリケーションによって行われた変更を受信したり、ファイアウォール経由でユーザーまたはアプリケーションを認証することもできます。

Directory Server とセキュリティー

Directory Server は、情報セキュリティーポリシーへの準拠を達成するためのさまざまなセキュリティー機能を提供します。これらの機能により、適切に承認されたユーザーのみが情報にアクセスできることが保証されます。

- マクロレベルの動的なアクセス制御命令 **(ACI):** LDAP 属性のレベルに至るまでの、きめ細かなアクセス制御を定義するための手段を提供します。アクセス制御ポリシーは一度定義すれば、以後はディレクトリツリー全体で再利用が可能です。マクロ ACI を使用してディレクトリ内の ACI の数を最適化することにより、セキュリティーフレームワークの複雑さを軽減できます。
- ロールベースのアクセス**:** ユーザーのエントリに含まれる情報に基づいて、アクセス権限を付与することができます。ロールはグループと同様に定義および管理されますが、アプリケーションに対してより効率的なグループ化機構を提供します。ACI 内でロールを使用して、データへのアクセスを制御できます。また、Directory Server の機能であるサービスクラス (CoS) では、ロールを使用することにより、多数のエントリに同時に適用できる仮想属性を作成できます。このような仮想属性は、エントリの記憶領域要件を軽減します。仮想属性を使用して、関連する多数のエントリを 1 回の変更で更新することもできます。
- 実行権限取得制御**:** 情報のセットに対してユーザーに付与するアクセス権限を決定するための手段を提供します。ディレクトリサービスのアクセスポリシーを保守する管理者は、ディレクトリのユーザーおよびアプリケーションのアクセス許可を監査することによってセキュリティーを強化できます。この機能を使用して、ユーザーの権限に応じて変化するインタフェースを持つアプリケーションを構築することもできます。
- 暗号化機構**:** ディスク上のデータや、通信チャネルを介して転送中のデータを保護します。Directory Server は、アクセス権限に基づいた分割レプリケーションやデータ非表示もサポートします。これらの機構は、EU およびその他の国際的なプライバシー規制に準拠する目的で使用できます。

- 複数のパスワードポリシー: ユーザー単位での定義や、特定グループのみを対象とした定義が可能です。これらのポリシーは、定期的なパスワード変更をユーザーに強制したり、アカウントへの未承認アクセスを確実にブロックするために役立ちます。

Directory Server と可用性

Directory Server は各種のアクセスプロトコルを標準でサポートし、分散環境での可用性の保証に役立つ、高い柔軟性と拡張性を備えたレプリケーション環境を提供します。

Directory Server のレプリケーションは、これらのプロトコルを使用してアイデンティティーデータにアクセスしているアプリケーションのシングルポイント障害を防ぎます。Directory Server は、ローカルエリアネットワークと広域ネットワークの両方をまたいでレプリケーションが行われる環境において、理論的には無制限の個数のマスターおよび読み取り専用コンシューマをサポートします。レプリケーションプロトコルの特殊な機能により、待ち時間の多いネットワーク経由でデータをレプリケートするときの最適化が可能になります。詳細については、207 ページの「高可用性を実現するためのレプリケーションと冗長性の使用」を参照してください。

Solaris プラットフォームでは、Directory Server はクラスタ化をサポートします。これは、あらかじめパッケージ化された高可用性ハードウェアとソフトウェアのソリューションです。詳細は、217 ページの「高可用性を実現するためのクラスタリングの使用」を参照してください。

Directory Server と拡張性

Directory Server では、配備の大幅な再設計を必要としない、垂直的な拡張と水平的な拡張の両方がサポートされています。このレベルの拡張性は、配備の規模が上がるにつれてますます重要になります。

Directory Server は、ハードウェア構成にもよりますが、1 台のマシン上での毎秒 20,000 エントリの持続的な検索パフォーマンスと、毎秒数千件の検索を実行できる水平的な拡張性を提供できます。読み取りの拡張容易性を高めるための Directory Server の配備方法については、第 10 章を参照してください。

情報を絶え間なく格納または更新するという要件は、組織全体でシステムの利用が拡大するにつれて顕著となります。Directory Server では、リレーショナルデータベースの書き込みパフォーマンスに近い更新パフォーマンスを実現しています。書き込みの拡張容易性を高めるための Directory Server の配備方法については、第 10 章を参照してください。

Directory Server は、「キャッシュからの読み取り」操作に関して、最大 28 CPU までの直線的な CPU 拡張性を提供します。メモリー容量の上限までのアクセスが可能であり、高いパフォーマンスを提供します。これにより、大規模なディレクトリを単一システム上で運用して、ハードウェアから最大限の利益を引き出すことができます。

Directory Server と保守容易性

Directory Server は、ディレクトリサービス全体だけでなく個別のサーバーを管理するための豊富な管理ツール群を提供します。

集中化された Web ベースの管理コンソールを使用して、複数の Directory Server を設定および管理できます。インタフェースには、設定から監視までの日常的なサーバー管理やサービスを効果的に実施するために必要な、すべてのツールが含まれています。加えて、サーバーの実行中に、コマンド行ユーティリティーの `dsadm` および `dsconf` を動的に使用できます。これらの管理機能により、ディレクトリをオンラインにしたままでほとんどの管理操作を実行できるため、可用性が最大化されます。

管理の柔軟性は、多くの異なる環境へのディレクトリサービスの配備を簡略化します。コマンド行ユーティリティーにより、サービスがローカルのデータセンター内にある場合と同様の簡単さでリモート管理を実行できます。

Directory Proxy Server

Directory Proxy Server は、LDAP アプリケーション層プロトコルのゲートウェイです。この製品は、強化されたディレクトリアクセス制御、スキーマ互換性、および高可用性を実現するために設計されています。

Directory Proxy Server と可用性

Directory Proxy Server は、設定可能な負荷分散、フェイルオーバー、フェイルバックなどの機能を通じて、システムが必要なデータにアクセスできることを保証します。

Directory Proxy Server は Directory Server と連携して、信頼性を保証し、サービス妨害攻撃を防御します。Directory Proxy Server は要求を適切に経路指定し、Directory Server に対して、セキュリティー保護されたファイアウォールと同様のサービスを提供します。

ミッションクリティカルアプリケーションのシングルポイント障害を防ぐために、Directory Proxy Server は停止部位を検出し、影響を受ける領域を避けてトラフィックを経路指定し、システム間で要求負荷を効率的に分散します。影響を受ける領域が復元されて稼働を再開すると、Directory Proxy Server は復元されたサーバーを自動的に検出します。

詳細は、209 ページの「冗長ソリューションの一部としての Directory Proxy Server の使用」を参照してください。

Directory Proxy Server とセキュリティー

Directory Proxy Server は、多数のユーザーによるディレクトリアクセスに対応し、このレベルのアクセスを提供することに伴うセキュリティーリスクを最小にします。管理者はセキュリティー機能を利用して、要求の発行元を特定し、要求を許可するかどうかを決定し、必要な認証のタイプを指定します。検索要求の場合、要求が最低限の要件を満たしているかどうかを Directory Proxy Server で検証することもできます。

Directory Proxy Server ではグループを使用して、LDAP クラスの識別方法や、特定のグループに属するクライアントに適用する制限を定義します。グループはさまざまな基準を使用して定義できます。

非公開ディレクトリの情報を未承認アクセスから保護するために、Directory Proxy Server では、LDAP ディレクトリに対するきめ細かなアクセス制御ポリシーを設定できます。このポリシーでは、ディレクトリのさまざまな箇所に対して、どのユーザーがどのようなタイプの操作を実行できるかを制御できます。Directory Proxy Server では、特定のタイプの操作、たとえば、Web トローラや Web ロボットによる情報探索目的の検索操作から、ディレクトリを保護するための設定が可能です。

Identity Synchronization for Windows

Identity Synchronization for Windows は、Directory Server Enterprise Edition と Microsoft Active Directory の間での、アイデンティティーデータの基本的な同期機能を提供します。

Identity Synchronization for Windows は相互運用性の要件を満たします。パスワードなどの重要なアイデンティティーデータの同期により、ユーザーは、異なるアプリケーション認証機構に対してパスワードを何度も変更する必要がなくなります。

システムの動作に影響を及ぼすことなく重要なアイデンティティーデータを同期できる機能により、Active Directory サーバーにクライアントコンポーネントをインストールする必要がなくなり、保守にかかる時間と手間を削減できます。

Identity Synchronization for Windows の導入により、ユーザーはパスワードやその他のアイデンティティーデータを、Windows 環境と Web ベースのアプリケーション環境のどちらでも変更できるようになります。このようにして、Identity Synchronization for Windows は、Active Directory と Directory Server の間で同期を維持します。Active Directory と Directory Server の間で、無効化されたアカウントを同期することもできます。この同期により、アプリケーションやデータに対するアクセスポリシーの適合性が、Windows デスクトップと Web ベースアプリケーションの間で保証されます。

Directory Editor

Directory Editor は、効率的で低コストなディレクトリデータ管理を実現する Java Web アプリケーションです。

Directory Editor は、ユーザーがディレクトリサービスの内部でアイデンティティーデータを管理できるようにすることで、保守容易性要件の充足に寄与します。管理者は、ユーザーが日常的なタスクを実行するために使用できるフォームベースの Web インタフェースを作成できます。Directory Editor は、インタフェースの強力なカスタマイズ、ブランド化、および埋め込みをサポートします。カスタマイズはコードを記述するのではなく、設定のためのフォームベースのインタフェースを使用して行います。データのセキュリティーとプライバシーを保証するために、組み込み型の承認コントロールで、メニューおよび操作の可視性を制限します。ユーザーには、アプリケーションの内部でそのユーザーへの表示が承認された要素のみが表示されます。

Directory Server Resource Kit

Directory Server Resource Kit は、Directory Server Enterprise Edition の配備、アクセス、チューニング、保守のためのツールおよびアプリケーションプログラミングインタフェース (API) を提供します。これらのユーティリティーは、より堅牢な LDAP ベースソリューションの実装および保守に役立ちます。

パフォーマンステストツールと容量計画ツールは、管理者が Directory Server Enterprise Edition のインストールに対してパフォーマンスを測定したり、容量計画を実施したりするために役立ちます。デバッグツールと保守ツールは、Directory Server Enterprise Edition の日常的な保守だけでなくトラブルシューティングにも役立ちます。配備ユーティリティーと配備ツールは、Directory Server Enterprise Edition の新規インストールの展開と新しいリリースへの移行を支援します。LDAP 生産性ツールには、Directory Server Enterprise Edition を使用して開発されたサンプル LDAP アプリケーションが含まれています。

また Sun は、テスト用の強力な負荷生成アプリケーションである SLAMD を開発しました。SLAMD には、Directory Server Enterprise Edition アプリケーションに対して徹底的なパフォーマンステストを実行するために必要なすべてのテストが含まれています。SLAMD は `http://www.slamd.com` から無償で入手できます。

配備内の Directory Server Enterprise Edition コンポーネント

配備する Directory Server Enterprise Edition コンポーネントの組み合わせは、組織の要件によって異なります。次の図は、以前に説明したコンポーネントを使用した一般的な配備シナリオを示しています。

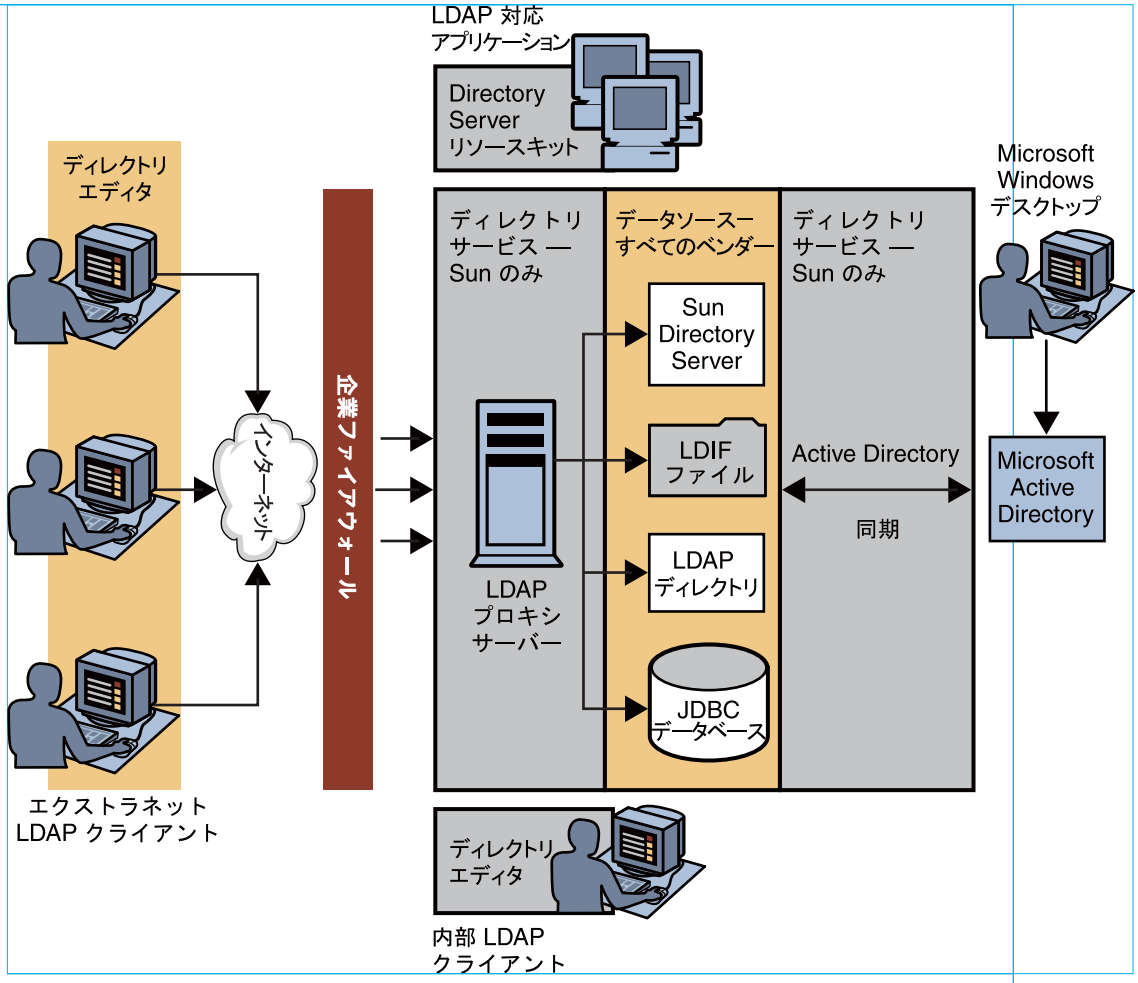


図 1–1　Directory Server Enterprise Edition コンポーネント

配備計画について

配備計画は、Directory Server Enterprise Edition ソリューションの実装を成功させる上で重要なステップです。それぞれの企業には、その企業に独自の一連の目標、要件、および考慮すべき優先事項が存在します。計画の成功は、企業の目標を分析し、その目標を達成するためのビジネス要件を特定することから始まります。次に、ビジネス要件を技術要件に翻訳する必要があります。技術要件は、企業の目標を達成するシステムを設計および実装するための基礎として使用できます。

成功する配備計画は、入念な準備、分析、および設計の成果として得られるものです。計画プロセスのどこかで発生した誤りや失敗により、多くの問題を内包したシ

ステムが生まれてしまう可能性があります。不十分な計画から生まれたシステムは、あとから重大な問題を引き起こす可能性があります。たとえば、システムに期待したパフォーマンスが出ない、保守が難しい、運用コストが高い、拡張性がない、などの問題が顕在化する可能性があります。

配備計画の原則については、『Sun Java Enterprise System 配備計画ガイド』で詳しく説明します。このガイドでは、明確に定義されたステップで配備計画を扱う「ソリューションライフサイクル」について言及します。

ソリューションライフサイクル

次の図に示すソリューションライフサイクルは、Java Enterprise System をベースとしたエンタープライズソフトウェアソリューションの計画、設計、および実装のステップを描いたものです。ライフサイクルは、配備プロジェクトを適切な状態に維持するための便利なツールです。ソリューションライフサイクルについては、『Sun Java Enterprise System 配備計画ガイド』で詳しく説明します。

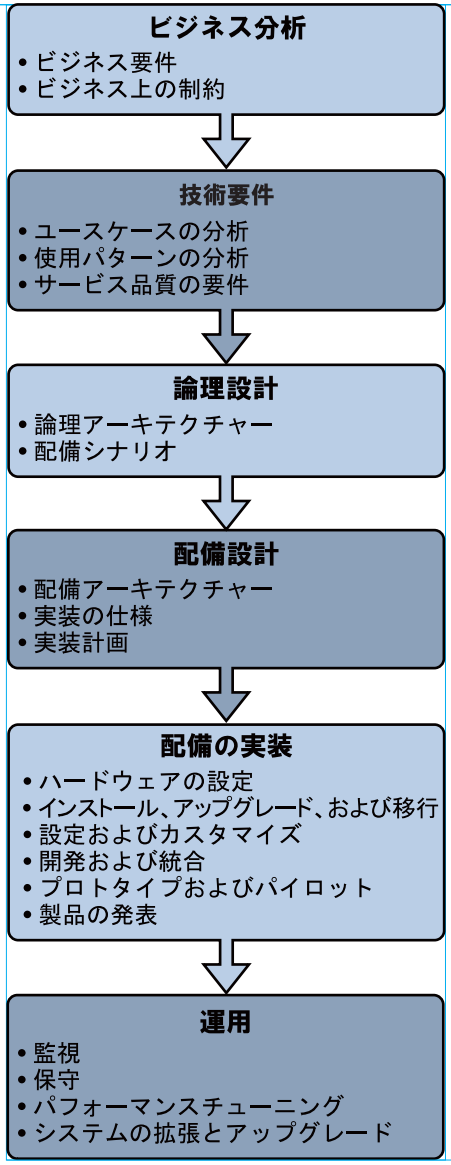


図1–2 ソリューションライフサイクル

◆ ◆ ◆ 第 2 章

Directory Server Enterprise Edition のためのビジネス分析

ソリューションライフサイクルのビジネス分析フェーズで、ビジネスの問題を分析することにより、ビジネス目標を定義します。次に、これらの目標を達成するためのビジネス要件およびビジネス制約を特定します。

この章で説明する内容は次のとおりです。

ビジネス分析について

ビジネス分析では、まずビジネス目標を記述します。次に、解決しなければならないビジネス問題を分析し、ビジネス目標を達成するために満たさなければならないビジネス要件を特定します。目標を達成する能力を制限するビジネス制約を検討します。ビジネス要件とビジネス制約の分析の成果物は、一連のビジネス要件ドキュメントです。

成果物として得られた一連のビジネス要件ドキュメントは、技術要件フェーズで技術要件を導出するための基礎として使用します。ソリューションライフサイクル全体を通して、計画およびソリューションの成功の度合いを、ビジネス分析フェーズで実施した分析に従って測定します。

Directory Server Enterprise Edition のビジネス要件の定義

ビジネス要件を識別するための単純な公式は存在しません。ビジネス要件を特定する上でベースとなるのは、アイデンティティー管理ソリューションを必要としている利害関係者との共同作業、対象のビジネス領域についての知識、そして創造的な応用思考です。『Sun Java Enterprise System 配備計画ガイド』では、ビジネス分析プロセスについて詳しく説明しています。このガイドでは、ビジネス要件およびビジ

ネス制約を定義するときに考慮すべき要因を示しています。この節では、堅牢なディレクトリサービスの必要性の動機となるビジネス要件について簡単に説明します。

ある企業では、次のような状況下で、堅牢なディレクトリサービスが必要とされています。

- 重要なビジネス情報およびビジネスアプリケーションをユーザーが利用できるようにする必要があります。ユーザー数は増加傾向にあり、入れ替わりも頻繁です。
- ここでいう「ユーザー」には、社内の従業員だけでなく、顧客、ベンダー、その他のビジネスパートナーなどの外部ユーザーも含まれます。

ディレクトリサービスは、効果的なアイデンティティー管理インフラストラクチャーのための、高可用性、拡張性、管理性、統合性、セキュリティーを備えた基礎を提供することによってこれらのニーズに応えます。ディレクトリサービスが提供する一連の機能により、ユーザーのアイデンティティーデータや、Web サービスアーキテクチャーのための補助データを格納する集中データストアを実現できます。

ディレクトリサービスは、効果的なアイデンティティー管理インフラストラクチャーを提供することによって、企業にとって重要な要件の達成に寄与します。その要件とは、ユーザーに対して、またユーザーが各自の業務を行うために利用するアプリケーションに対して安定したサービスを提供できることです。

そのような要件には、次のものが含まれます。

- 大規模で入れ替わりが頻繁なユーザーのグループに対し、開かれたアクセスを提供する
- セキュリティーを高め、情報の適正な利用と共有、および機密情報の保護を保証する
- ユーザーおよびアプリケーションに対し、アクセス信頼性と高いサービス品質を一貫して提供する
- ビジネスニーズの変化やユーザー要件の増加に左右されることなく、情報やサービスを効率的にユーザーに提供する

高可用性、信頼性、セキュリティー、パフォーマンスに優れたディレクトリサービスは、ビジネスの主たる原動力、すなわちセキュリティー、サービス品質、およびコスト効率の向上に寄与します。

パート II

技術要件

技術要件分析は、ソリューションライフサイクルのビジネス分析フェーズの間に作成されるビジネス要件ドキュメントから始まります。ビジネス分析を使用して、使用状況分析を実施します。この分析は、予測される負荷条件を特定したり、システムとユーザーの一般的な対話をモデル化するユースケースを作成したりするために役立ちます。この分析は、一連の品質サービス要件を作成するときにも役立ちます。これらの要件は、配備されるソリューションが、応答時間、可用性、セキュリティーなどの領域で満たさなければならない基準を定義します。

この第 2 部では、Directory Server Enterprise Edition の配備に対して定義する必要のある技術要件について説明します。次の章から構成されています。

- 第 3 章では、使用状況分析の要件について説明します。
- 第 4 章では、データ要件の定義方法について説明します。
- 第 5 章では、サービス品質要件について説明します。
- 第 6 章では、Directory Server Enterprise Edition のシステム要件について説明します。
- 第 7 章では、セキュリティー要件について説明します。
- 第 8 章では、設計時に行う必要がある管理面の決定について説明します。

◆ ◆ ◆ 第 3 章

3 Directory Server Enterprise Edition の使用分析

使用分析では、システムのユーザーを特定し、それらのユーザーの使用パターンを決定します。使用分析ではそうすることで、予期されるディレクトリサービスの負荷条件を決定できます。

使用分析の要因

サーバーをどのように配備するかは、なぜ、Sun Java System Directory Server Enterprise Edition をアイデンティティー管理ソリューションとして提供するのかという理由と密接に関連します。

使用分析時には、可能であれば常にユーザーへのインタビューを行います。既存データを調査して使用パターンを決定し、開発者や管理者に以前のシステムについてのインタビューを行います。使用分析では、第 5 章で説明するサービス要件の決定を可能にするようなデータを提供すべきです。

使用分析から得られるべき情報は、次のとおりです。

- クライアントアプリケーションの数と種類:配備がサポートする必要のあるクライアントアプリケーションの数を特定し、必要であればそれらのアプリケーションを分類します。
- 管理ユーザー:ディレクトリにアクセスしてその配備を監視、更新、サポートするユーザーを特定します。ファイアウォールの外側から配備を管理するなど、技術要件に影響する可能性のあるすべての特定の管理使用パターンを決定します。
- 使用パターン:各種アプリケーションのシステムへのアクセス方法を特定し、予期される使用法に対する目標値を定めます。

  たとえば、次の質問に答えてください。

  - 使用量が最高になる時間帯が存在するか。
  - 通常の業務時間帯。
  - クライアントアプリケーションがグローバルに分散配置されるか。

- アプリケーション接続の予期される持続時間。

- クライアントアプリケーションの増大:クライアントアプリケーションの数は固定されるか、あるいは増大することが予期されるかを決定します。アプリケーションが追加されることが予期される場合は、どれくらい増大するかを妥当な範囲で見積もってみてください。
- アプリケーショントランザクション:サポートする必要のあるトランザクションの種類を特定します。

  それらのトランザクションはユースケースに分類できます。次に例を示します。

  - アプリケーションによってどのようなタスクが実行されるか。
  - アプリケーションがディレクトリにバインドしたあと、そのバインドがそのまま維持されるか、あるいはいくつかのタスクを実行してから解除されるか。
- 調査書や統計データ:既存の調査書やその他のリソースに基づいてアプリケーションの動作パターンを決定します。企業や業界組織はしばしば研究調査書を所有していますが、そうした調査書からは、ユーザーやクライアントアプリケーションに関する有用な情報を抽出できます。既存のアプリケーションのログファイルには、システムの見積もりに役立つ統計データが含まれている可能性があります。

使用分析の詳細については、『Sun Java Enterprise System 配備計画ガイド』を参照してください。

◆ ◆ ◆ 第 4 章

4 データ特性の定義

ディレクトリに格納するデータの種類によって、ディレクトリの構造、データにアクセスできるユーザー、およびアクセス権限の付与方法が決定されます。特によく利用されるデータの種類には、ユーザー名、電子メールアドレス、電話番号、ユーザーが所属するグループについての情報などがあります。

この章では、データを検索、分類、構造化、および整理する方法について説明します。また、Directory Server のスキーマにデータをマップする方法についても説明します。この章の内容は次のとおりです。

データソースと所有者の設定

既存のデータを分類する最初のステップでは、データがもともとどこに存在していたものか、またそのデータの所有者がだれであるかを特定します。

データソースの特定

ディレクトリに含めるデータを特定するには、既存のデータソースの位置を特定して分析します。

- 情報を提供する組織を特定する。

  企業にとって重要な情報を管理している組織をすべて特定します。通常、これらの組織には、情報サービス部、人事部、給与計算部、および経理部が含まれます。

- 情報のソースであるツールとプロセスを特定する。

  情報の一般的なソースには、次のものがあります。

  - Windows、Novell Netware、UNIX® NIS などのネットワークオペレーティングシステム
  - 電子メールシステム
  - セキュリティーシステム
  - PBX (構内交換機) または電話交換システム
  - 人事アプリケーション

- データの集中化が各データに与える影響を判定する。

  集中データ管理では、新しいツールとプロセスが必要になることがあります。集中化によって、ある組織のスタッフを増員してほかの組織のスタッフを減らすことが必要になる場合は、問題が生じる可能性もあります。

データ所有者の決定

「データ所有者」とは、データを最新の状態に維持する責任のある個人または組織のことです。データの設計時に、ディレクトリにデータを書き込めるユーザーを決めておきます。データの所有者を決めるには、一般に次の規則に従います。

- ディレクトリの内容を管理する少人数のマネージャーグループを除くすべてのユーザーに対して、ディレクトリへの読み取り専用アクセスを許可する。
- 各ユーザーが自分自身に関する重要な情報を管理できるようにする。

  このような情報には、ユーザーのパスワード、ユーザー自身についての説明的情報、組織内でのユーザーのロールなどがあります。

- 人に関する重要な情報 (連絡先情報や役職など) を上司が書き込めるようにする。
- 組織の管理者がその組織に関するエントリを作成および管理できるようにする。

  これにより、実質的に組織の管理者が、ディレクトリ内容の管理者にもなります。

- グループ内のユーザーに読み取りアクセス権または書き込みアクセス権を与えるロールを作成する。

たとえば、人事、財務、経理などのロールを作成します。これらのロールごとに、そのグループが必要とするデータへの読み取りアクセス権、書き込みアクセス権、またはその両方を許可します。このようなデータには、給与情報、政府指定の ID 番号、自宅の電話番号と住所などがあります。

ロールとエントリのグループ化の詳細は、51 ページの「ディレクトリデータのグループ化と属性の管理」、Chapter 10, 「Directory Server Groups, Roles, and CoS,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』およびChapter 8, 「Directory Server Groups and Roles,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

データに書き込みを許可するユーザーを決定するときに、複数のユーザーに対して同じデータへの書き込み権限が必要になることがあります。たとえば、情報システムまたはディレクトリ管理グループには、従業員のパスワードへの書き込み権限を許可するのが一般的です。同時に、すべての従業員にも自分自身のパスワードへの書き込み権限を許可する場合があります。複数のユーザーに同じ情報への書き込み権限を与えなければならない場合がありますが、このような場合はこのグループを少人数に保ち、容易に特定できるようにします。グループを少人数にすることは、データの完全性の保証に役立ちます。

ディレクトリのアクセス制御の設定については、Chapter 7, 「Directory Server Access Control,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』および「How Directory Server Provides Access Control」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

ユーザーデータと設定データの区別

Directory Server やその他の Java Enterprise System サーバーを設定するために使用するデータと、ディレクトリに格納される実際のユーザーデータを区別するには、次の手順に従います。

- ユーザーデータと設定データのそれぞれに対して、異なるバックアップ方針を用意します。
- ユーザーデータと設定データのそれぞれに対して、異なる高可用性基準を用意します。
- 設定サーバーをすみやかにシャットダウンし、復元し、再起動します。
- ほかの Directory Server インスタンスの保守を実行している間、設定サーバーの稼働を維持します。

異なるデータソースからのデータの特定

データソースを特定するときは、従来のデータソースを含め、ほかのデータソースからのデータを必ず含めるようにしてください。このデータはディレクトリに格納されていない場合もあります。ただし、Directory Server がデータについて何らかの情報を持っている、またはデータを制御できることが必要な場合があります。

Directory Proxy Server には、複数のデータリポジトリからリアルタイムで情報を集約する「仮想ディレクトリ」機能が用意されています。このようなリポジトリには、LDAP ディレクトリ、JDBC 仕様に準拠するデータ、LDIF フラットファイルなどがあります。

仮想ディレクトリは、複数の異なるデータソースからの属性を処理する複合フィルタをサポートします。また、複数の異なるデータソースからの属性を結合する変更もサポートします。

データ分析フェーズの間、複数のアプリケーションが同じデータを異なる形式で必要とすることが判明する場合があります。このような情報 (データ) については、複製するのではなく、アプリケーションの側でそれぞれの要件に合わせて変換させるのが適切な方法です。

ディレクトリ情報ツリーを使用したデータの構造化

ディレクトリ情報ツリー (DIT) は、ディレクトリデータを構造化し、クライアントアプリケーションからデータを参照できるようにするための手段です。DIT は、ディレクトリデータの分散、レプリケート、アクセス制御の方法など、その他の設計判断と緊密に連携します。

DIT の用語

適切に設計された DIT は、次のような利点をもたらします。

- ディレクトリデータの管理を簡単にする
- レプリケーションポリシーとアクセス制御の作成における柔軟性
- ディレクトリを使用するアプリケーションをサポートする
- ユーザーが簡単にディレクトリを操作できるようにする

DIT の構造は、階層型の LDAP モデルに従います。DIT では、たとえば、グループ、ユーザー、あるいは場所ごとにデータを編成できます。また、ディレクトリツリーによって複数のサーバー間でどのようにデータを分散して配置するかが決まります。

DIT の設計は、レプリケーション設定や、Directory Proxy Server を使用してデータを分散させる方法に影響を及ぼします。DIT の特定部分をレプリケートまたは分散す

る場合は、レプリケーションおよびDirectory Proxy Serverの要件について設計時点で検討します。また、分岐点にアクセス制御を適用する必要があるかどうかについても設計時に検討します。

DITはサフィックス、サブサフィックス、および連鎖サフィックスによって定義されます。「サフィックス」とは分岐またはサブツリーであり、サフィックスの内容全体が管理タスクの単位として扱われます。インデックス生成はサフィックス全体に対して定義され、サフィックス全体を1回の操作で初期化できます。サフィックスは通常、レプリケーションの単位でもあります。同じ方法でアクセスおよび管理するデータは、同じサフィックスに格納する必要があります。サフィックスは、「ルートサフィックス」と呼ばれるディレクトリツリーのルートに配置することもできます。

データはサフィックスレベルのみで分割できるため、複数のサーバー間でデータを分散するには、適切なディレクトリツリー構造が必要です。

次の図は、2つのルートサフィックスを持つディレクトリを示しています。各サフィックスは独立した企業エンティティーを表します。

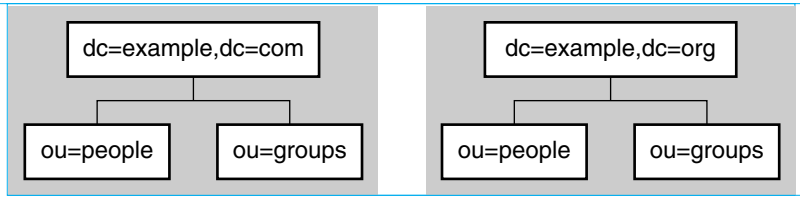


図4–1　単一Directory Server内の2種類のルートサフィックス

サフィックスは、他のサフィックスの分岐となることもできます。その場合は「サブサフィックス」と呼ばれます。親サフィックスには、管理操作のためのサブサフィックスの内容は含まれません。サブサフィックスはその親とは別個に管理されます。LDAP操作の結果にはサフィックスに関する情報は含まれないため、ディレクトリクライアントは、エントリがルートサフィックスの一部であるか、サブサフィックスの一部であるかを認識できません。

次の図に、大規模な企業エンティティーに対して、単一のルートサフィックスと複数のサブサフィックスを配置したディレクトリを示します。

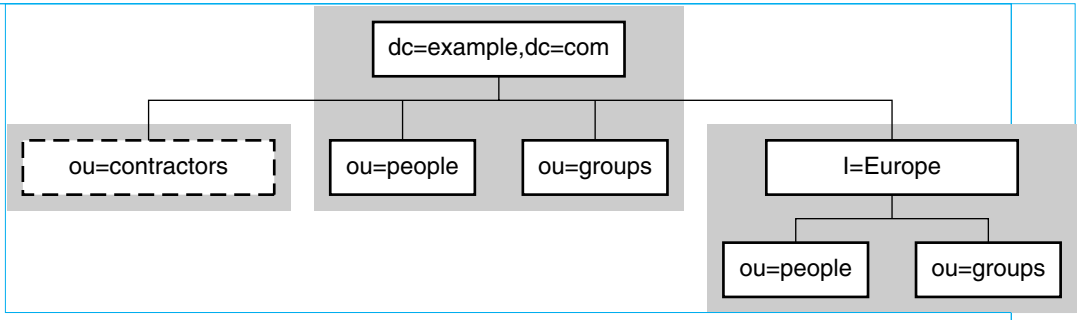


図 4–2 複数のサブサフィックスを持つ単一のルートサフィックス

サフィックスは、サーバー内の個々のデータベースに対応します。ただし、データベースとそのファイルはサーバーによって内部的に管理され、データベースの用語は使用されません。

連鎖サフィックスは、仮想 DIT を作成して、別のサーバー上にあるサフィックスを参照します。Directory Server は、連鎖サフィックスを使用してリモートサフィックスに対する操作を実行します。ディレクトリはその後、操作がローカルで実行された場合と同じように結果を返します。データの位置は透過的です。クライアントの側では、サフィックスが連鎖されている、またデータがリモートサーバーから取得されているという認識はありません。あるサーバー上のルートサフィックスが、別のサーバーに連鎖されているサブサフィックスを持つ場合があります。この場合、クライアントは単一のツリー構造を認識します。

カスケード型連鎖の場合、連鎖サフィックスはリモートサーバー上などの別の連鎖サフィックスを参照することがあります。各サーバーは操作を転送し、最終的にクライアントの要求を処理するサーバーへ結果を返します。

DIT の設計

DIT の設計では、データを格納するサフィックスの選択、データエントリ間の階層関係の決定、DIT 階層内のエントリのネーミングを行います。次に、設計の各段階について詳しく説明します。

サフィックスの選択

サフィックスは、DIT のルートにあるエントリの名前です。共通のルートを持たない DIT が 2 つ以上ある場合、複数のサフィックスを使用できます。デフォルトの Directory Server のインストールには、複数のサフィックスが含まれています。1 つのサフィックスはユーザーデータの格納に使用されます。その他のサフィックスは、設定情報やディレクトリのスキーマなど、内部的なディレクトリ操作に必要なデータの格納に使用されます。

ディレクトリ内のすべてのエントリは、共通のベースエントリ (サフィックス) の下に格納する必要があります。各サフィックス名は次のように指定する必要があります。

- グローバルに一意の名前にする
- 変更しない、または名前をまれにしか変更しないようにする
- そのサフィックスの下にあるエントリがオンラインで読みやすいように短い名前にする
- ユーザーが容易に入力および記憶できるものにする

一般的には、企業のドメイン名を識別名 (DN) にマップすることがベストプラクティスと考えられています。たとえば、`example.com` というドメイン名を持つ企業は「`dc=example,dc=com`」という DN を使用します。

DIT 構造の作成とエントリのネーミング

DIT の構造は、フラットまたは階層型から選択できます。フラットツリーのほうが管理が容易ですが、データのパーティション分割、レプリケーション管理、およびアクセス制御のために、ある程度の階層が必要な場合もあります。

分岐点とネーミングの考慮事項

「分岐点」とは、DIT の内部で新しいサブ区分を定義するポイントのことです。分岐点を決定するときは、問題が生じる可能性のある名前の変更は避けてください。名前を変更する確率は、名前が変更される可能性があるコンポーネントが、ネームスペース内に多く含まれているほど高くなります。DIT の階層が深いほどネームスペース内のコンポーネントは多くなり、名前を変更する確率が高くなります。

分岐点を定義および命名するときは、次のガイドラインを使用します。

- 企業組織内でもっとも大きい部門区分のみを表すようにツリーを分岐させます。

  分岐点を部門 (企業情報サービス、カスタマサポート、販売、プロフェッショナルサービスなど) に限定します。部門が安定していることを確認します。企業で組織変更が頻繁に行われる場合は、この種の分岐を実行しないでください。
- 組織の実際の名前ではなく、機能を表す名前または一般的な名前を使用します。

  組織名は変更される可能性があり、企業で部門の名前を変更するたびに DIT の変更が必要になるのは問題です。代わりに、組織の機能を表す一般的な名前を使用します。たとえば、「Widget 研究開発」ではなく「エンジニアリング」を使用します。
- 似たような機能を持つ組織が複数ある場合は、部門の構成に基づいて分岐点を作成するのではなく、その機能を表す分岐点を 1 つ作成します。

  たとえば、特定の製品ラインを担当する複数のマーケティング部門がある場合でも、1 つのマーケティングサブツリーのみを作成します。すべてのマーケティングエントリは、そのツリーに所属させます。

- 次の表に示した、従来からある分岐点属性のみを使用するようにします。

  従来からある属性を使用すると、サードパーティーの LDAP クライアントアプリケーションとの互換性が保たれる可能性が高くなります。加えて、従来からある属性はデフォルトのディレクトリスキーマにとって既知であるため、分岐識別名 (DN) のエントリの構築が簡略化されます。
- ディレクトリに格納されるデータの種類に応じた分岐を行います。

  たとえば、人、グループ、サービス、およびデバイスのそれぞれに対応する独立した分岐を作成します。

表 4–1 従来からある DN 分岐点の属性

属性名	定義
c	国名。
o	組織名。この属性は通常、大規模な部門の分岐を表すために使用します。そのような分岐の例には、企業の部門、教育機関の学部、子会社、企業内のその他の主要部門などがあります。また、ドメイン名を表す場合もこの属性を使用することをお勧めします。
ou	組織の構成単位。通常この属性は、組織よりも小さな組織内の部門を表すために使用します。組織単位は、一般にすぐ上の組織に属します。
st	州または県の名前。
l	地域 (都市、地方、オフィス、施設名など)。
dc	ドメインのコンポーネント。

分岐点の属性を選択するときは、整合性を保つようにしてください。DIT 全体で DN の形式が統一されていないと、一部の LDAP クライアントアプリケーションで問題が発生する可能性があります。DIT のある部分で `l (localityName)` が `o (organizationName)` の下位にある場合、必ず、ディレクトリのその他すべての部分でも `l` が `o` の下位にあるようにしてください。

レプリケーションに関する検討事項

DIT を設計するときは、どのエントリがほかのサーバーにレプリケートされるかを考慮します。エントリの特定グループを同じサーバー集合にレプリケートする場合、それらのエントリを特定のサブツリー下に配置することをお勧めします。レプリケートするエントリの集合を記述するには、サブツリーの頂点で DN を指定します。エントリのレプリケーションの詳細は、Chapter 4, 「Directory Server Replication,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』 を参照してください。

アクセス制御に関する検討事項

DIT 階層では、特定のタイプのアクセス制御を有効にできます。レプリケーションの場合と同様に、類似したエントリをグループ化したり、エントリを 1 つの分岐から管理したりすることがさらに容易になります。

また、DIT 階層で管理を分散させることもできます。たとえば、DIT を使用して、マーケティング部門の管理者にはマーケティング関連エントリへのアクセス権を付与し、セールス部門の管理者にはセールス関連のエントリへのアクセス権を付与することができます。

DIT ではなくディレクトリの内容に基づいてアクセス制御を設定することもできます。ACI でフィルタされたターゲット機構を使用して、単一のアクセス制御規則を定義します。この規則では、1 つのディレクトリエントリが、特定の属性値を含むすべてのエントリにアクセスできることを記述します。たとえば、`ou=Sales` という属性を含むすべてのエントリへのアクセス権を営業部の管理者に与える ACI フィルタを設定できます。

ただし、ACI フィルタは管理が簡単ではありません。対象のディレクトリにもっとも適したアクセス制御方法を決定する必要があります。DIT 階層で組織に対応した分岐を作成する、ACI フィルタを適用する、あるいはこの両者を組み合わせる、などの方法から選択します。

ディレクトリデータのグループ化と属性の管理

ディレクトリ情報ツリーは、エントリを階層構造で構成します。この階層は、グループ化メカニズムの 1 種です。この階層は、分散しているエントリの関連付け、頻繁に変更される組織、または多数のエントリで繰り返されるデータには適していません。Directory Server のグループとロールは、エントリ間のより柔軟な関連付けを提供します。サービスクラス (CoS) のメカニズムを使用すると、属性がエントリ間で共有されるように各属性を管理できます。この共有は、アプリケーションには見えない方法で実行されます。

これらのエントリのグループ化と属性管理のメカニズムは、Chapter 8, 「Directory Server Groups and Roles,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』およびChapter 9, 「Directory Server Class of Service,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』で詳細に説明されています。

ここでは、管理戦略の設計には十分なグループ化メカニズムの概要について説明します。ただし、このメカニズムの動作の仕組みや、それらの設定方法については説明していません。

この節は、次の項目で構成されています。

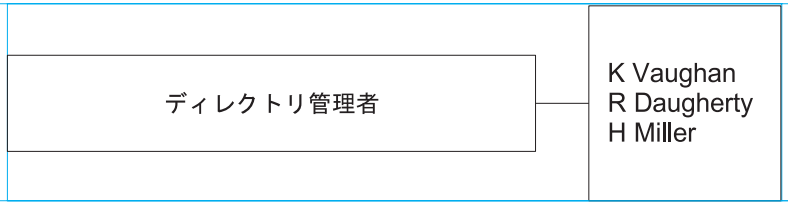
スタティックグループ、ダイナミックグループ、および入れ子のグループ

Directory Server は、スタティックグループ、ダイナミックグループ、および入れ子のグループを識別します。

グループが識別するメンバーはディレクトリ内のどこに配置してもかまいませんが、グループ定義自体は `ou=Groups` のように適切な名前が付けられたノードに配置するようにします。このようにすることで、バインド資格がグループのメンバーである場合にアクセス権を付与または制限するアクセス制御命令 (ACI) を定義するときなどに、グループ定義を簡単に見つけることができます。

スタティックグループ

スタティックグループは、メンバーエントリの名前を明示的に指定します。たとえば、次の図のように、ディレクトリ管理者のグループにはグループを構成する特定のユーザーの名前が指定されます。



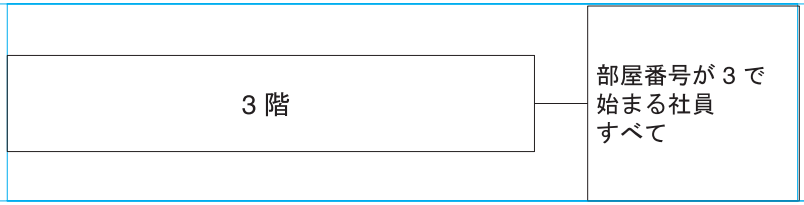
次の LDIF の抜粋は、このスタティックグループのメンバーが定義される方法を示しています。

```
dn: cn=Directory Administrators, ou=Groups, dc=example,dc=com
...
member: uid=kvaughan, ou=People, dc=example,dc=com
member: uid=rdaugherty, ou=People, dc=example,dc=com
member: uid=hmiller, ou=People, dc=example,dc=com
```

ダイナミックグループ

ダイナミックグループはフィルタを指定し、そのフィルタと一致するすべてのエントリがグループのメンバーとなります。このようなグループは、フィルタが評価されるたびにメンバーが定義されるため、ダイナミックグループと呼ばれます。

たとえば、すべての管理職とそのアシスタントがビルの 3 階におり、各社員の部屋番号がその階の数字で始まるとします。3 階にいる社員のみを含むグループを作成するのであれば、次の図のように、部屋番号を使用してそれらの社員だけを定義することができます。



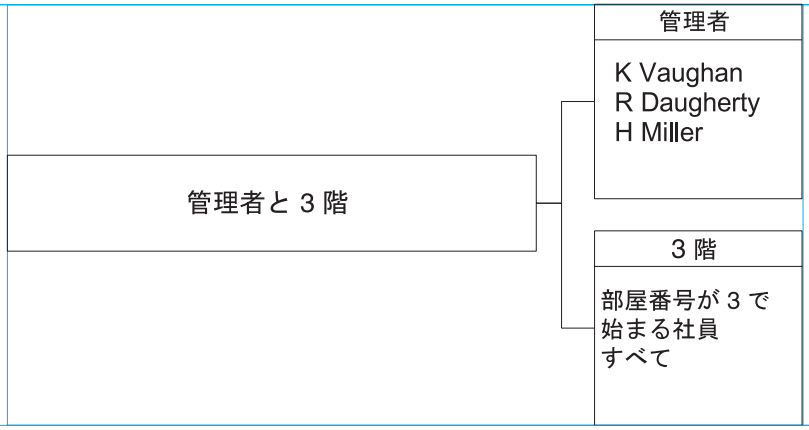
次の LDIF の抜粋は、このダイナミックグループのメンバーが定義される方法を示しています。

```
dn: cn=3rd Floor, ou=Groups, dc=example,dc=com
...
memberURL: ldap:///dc=example,dc=com??sub?(roomnumber=3*)
```

入れ子のグループ

入れ子のグループでは、別のグループの DN をスタティックまたはダイナミックグループの `uniqueMember` 属性として使用して、他のグループの内部にグループを配置します。Directory Server では、個々のエントリ、スタティックグループ、およびダイナミックグループを参照する混合グループもサポートしています。

たとえば、すべてのディレクトリ管理者、およびすべての管理職とそのアシスタントを含むグループを作成するとします。この場合、先ほど定義した 2 つのグループの組み合わせを使用して、次の図のように 1 つの入れ子のグループを作成することができます。



次の LDIF の抜粋は、この入れ子のグループのメンバーが定義される方法を示しています。

```
dn: cn=Admins and 3rd Floor, ou=Groups, dc=example,dc=com
...
member: cn=Directory Administrators, ou=Groups, dc=example,dc=com
member: cn=3rd Floor, ou=Groups, dc=example,dc=com
```

入れ子のグループはグループ化の手法として必ずしも効率的ではありません。ダイナミックな入れ子のグループには、非常に大きなパフォーマンスコストが必要となります。このようなパフォーマンス上の問題を避けるために、グループではなくロールの使用を考慮してください。

管理されているロール、フィルタを適用したロール、入れ子のロール

ロールは、エントリのグループ化メカニズムです。ロールを使用すると、エントリがディレクトリから取得されるとすぐに、ロールのメンバーシップを決定できます。各ロールはメンバー (そのロールを所有するエントリ) を持ちます。グループと同じようにロールのメンバーを明示的またはダイナミックに指定できます。

Directory Server は、次の 3 種類のロールをサポートしています。

- 管理されているロール: 明示的にメンバーエントリにロールを割り当てる。
- フィルタを適用したロール: エントリが指定された LDAP フィルタに一致した場合は、それらのエントリを自動的にメンバーにする。このため、ロールは各エントリに含まれている属性に依存する。
- 入れ子のロール: ほかのロールを含むロールを作成できる。

グループとロールのどちらを使用するかの決定

グループとロールのメカニズムは、一部の機能が重なっています。どちらのメカニズムにも利点と欠点があります。一般に、ロールメカニズムのほうが、頻繁に必要となる機能を効率的に提供できるように設計されています。グループ化メカニズムはサーバーの複雑さに影響を及ぼし、グループ化メカニズムを選択することで、クライアントがどのようにメンバー情報を処理するかが決まるため、グループ化メカニズムは慎重に計画する必要があります。どちらのメカニズムがより適しているかを判断するには、実行する標準的なメンバーシップクエリーと管理操作を理解してください。

グループメカニズムの利点

グループには、次の利点があります。

- スタティックグループは、標準に準拠した唯一のグループ化メカニズムである。そのため、スタティックグループは、ほとんどのクライアントアプリケーションおよびLDAPサーバーと相互運用できます。
- メンバーを列挙するには、ロールよりスタティックグループの方が適している。

  特定のセットのメンバーを列挙するだけでよい場合は、スタティックグループの方がコストが低くなります。`member`属性を取得してスタティックグループのメンバーを列挙することは、同じロールを共有するすべてのエントリを調べるより簡単です。Directory Serverでは、多くの値を持つ複数値属性に対して大幅なパフォーマンス強化が実現されています。特にスタティックグループとの関連では、これらの属性に対する等価照合や変更操作が大幅に機能強化されています。また、グループエントリに対するメンバーシップのテストも機能強化されています。これらの機能強化によって、スタティックグループに対して以前あった制限の一部(特にグループサイズに対する制限)が解消されています。

  また、Directory Serverでは`isMemberOf`オペレーショナル属性をユーザーエントリに直接指定することで、そのユーザーがどのグループのメンバーであるかを明示できます。この機能はスタティックグループのみに適用されますが、入れ子のグループも含まれます。詳細については、「Managing Groups」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。
- メンバーの割り当てや削除などの管理操作には、ロールよりスタティックグループが適している。

  スタティックグループは、ユーザーをセットに割り当てたり、ユーザーをセットから削除したりするためのもっとも単純なメカニズムです。ユーザーをグループに追加するために、特殊なアクセス権は必要ありません。

  グループエントリを作成する権限があると、そのグループにメンバーを割り当てる権限が自動的に与えられます。これは、管理されているロールやフィルタを適用したロールには当てはまりません。これらのロールでは、管理者が、ユーザーエントリに`nsroledn`属性を書き込むための権限も持っている必要があります。ア

クセス権に関する同じ制約が、入れ子のロールにも間接的に適用されます。入れ子のロールを作成するには、入れ子の対象となるすでに定義されているほかのロールに対する権限が必要となります。

- フィルタベースの ACI で使用するには、ロールよりダイナミックグループが適している。

  ACI でのバインドルールの指定など、フィルタに基づいてすべてのメンバーを検索するだけの場合は、ダイナミックグループを使用します。フィルタを適用したロールはダイナミックグループと似ていますが、ロールメカニズムを起動して `nsRole` 仮想属性を生成します。クライアントが `nsRole` 値を必要としない場合は、ダイナミックグループを使用してこの計算のオーバーヘッドを回避します。

- 既存のセットに対してセットの追加や削除を行うときは、ロールよりグループが適している。

  あるセットを既存のセットに追加したり、あるセットを既存のセットから削除する場合は、グループメカニズムがもっとも単純です。グループメカニズムには、入れ子の制限がありません。ロールメカニズムでは、他のロールを受け入れられるのは入れ子のロールに限られています。

- エントリのグループ化範囲の柔軟性が重要になる場合は、ロールよりグループが適している。

  グループのメンバー範囲は、グループ定義エントリの場所に関係なくディレクトリ全体であるため、範囲の面ではグループには柔軟性があります。ロールでも、特定のサブツリーを超えて範囲を拡張することができますが、入れ子のロールに範囲拡張属性 `nsRoleScopeDN` を追加することでしか実現できません。

ロールメカニズムの利点

ロールには、次の利点があります。

- セットのメンバーを列挙し、かつ特定のエントリがメンバーになっているすべてのセットを検索する場合は、ダイナミックグループよりロールの方が適している。スタティックグループでも、`isMemberOf` 属性によってこの機能を提供しています。

  ロールはメンバーシップ情報をユーザーエントリにプッシュします。そこでこの情報をキャッシュできるので、以後のメンバーシップのテストをより効率的に行えます。サーバーがすべての計算を実行し、クライアントは `nsRole` 属性の値を読み取るだけです。さらに、この属性にすべてのタイプのロールが含まれ、クライアントはすべてのロールを均等に処理できます。ダイナミックグループよりもロールの方が、両方の処理をより効率よく実行でき、クライアント側での処理が簡単になります。

- CoS、パスワードポリシー、アカウントの無効化、ACI など、Directory Server の既存の機能とグループ化メカニズムを統合する場合は、グループよりロールが適している。

  セットのメンバーシップをサーバーで「自然に」使用する場合は、ロールの方が優れたオプションです。ロールを利用するということは、サーバーが自動的に実行するメンバーシップの計算方法を使用することを意味します。ロールは、リソース指向の ACI で、CoS のベースとして、より複雑な検索フィルタの一部として、さらにパスワードポリシーやアカウントの無効化などに使用することができます。グループを使用してこのような統合を行うことはできません。

ロールに対するアクセス権の制限

ロールを使用する場合は、次の問題に注意してください。

- `nsRole` 属性を割り当てることができるのはロールメカニズムだけです。この属性は、どのディレクトリユーザーも割り当てたり、変更したりできませんが、どのディレクトリユーザーも「読み取れる」可能性があります。この属性が、不正なユーザーから読み取られないようにするには、アクセス制御を定義してください。
- `nsRoleDN` 属性は、管理されているロールのメンバーシップを定義します。ユーザーがロールに自分自身を追加したり削除したりできるかどうかを決定してください。自分のロールを変更できないようにするには、そのための ACI を定義してください。
- フィルタを適用したロールは、ユーザーエントリ内の属性の存在または値に基づくフィルタを介してメンバーシップを決定します。フィルタを適用したロール内のメンバーシップを定義できるユーザーを制御するには、これらの属性のユーザーアクセス権を慎重に割り当ててください。

サービスクラスによる属性の管理

サービスクラス (CoS) メカニズムを使用すると、エントリ間で属性を共有できます。ロールメカニズムと同様に、エントリが検出されると、CoS がそのエントリに対して仮想属性を生成します。CoS はメンバーを定義しませんが、一貫性の保持と容量の節約のために、関連するエントリがデータを共有できるようにします。CoS の値は、それらの値が要求されたときに動的に計算されます。CoS 機能や、CoS のさまざまな種類については、『Sun Java System Directory Server Enterprise Edition 6.3 Reference』で詳細に説明されています。

以降の節では、パフォーマンスの低下を回避しながら、CoS 機能を意図したとおりに使用する方法について詳しく説明します。

- 58 ページの「多数のエントリが同じ値を共有する場合の CoS の使用」
- 59 ページの「エントリが自然な関係を持っている場合の CoS の使用」
- 62 ページの「過剰な CoS 定義の回避」

注 – CoS の生成は、常にパフォーマンスに影響します。実際に必要とするより多くの属性を検索するクライアントアプリケーションによって、この問題がさらに悪化する可能性があります。

クライアントアプリケーションの記述方法を指示できる場合は、実際に必要な属性値のみを検索するようにクライアントアプリケーションを記述することで、パフォーマンスの大幅な向上が期待できることを開発者に伝えてください。

多数のエントリが同じ値を共有する場合の CoS の使用

CoS は、サブツリー内の多数のエントリに同じ属性値を表示する必要がある場合に、比較的低いコストで大きな利点を提供します。

たとえば、`ou=People` の下のすべてのユーザーエントリに `companyName` 属性が含まれている、MyCompany, Inc. のディレクトリを考えてみます。請負業者のエントリには `companyName` 属性に直接値がセットされていますが、すべての正社員の `companyName` には、CoS で生成された値 `MyCompany, Inc.` がセットされています。次の図は、この例でポインタ CoS を使用した場合を示しています。CoS によって、すべての正社員の `companyName` 値が生成されるが、請負業者の従業員用に直接設定されている実際の `companyName` 値は置き換えられないことに注意してください。会社名は、`companyName` が許可された属性になっているエントリに対してのみ生成されます。

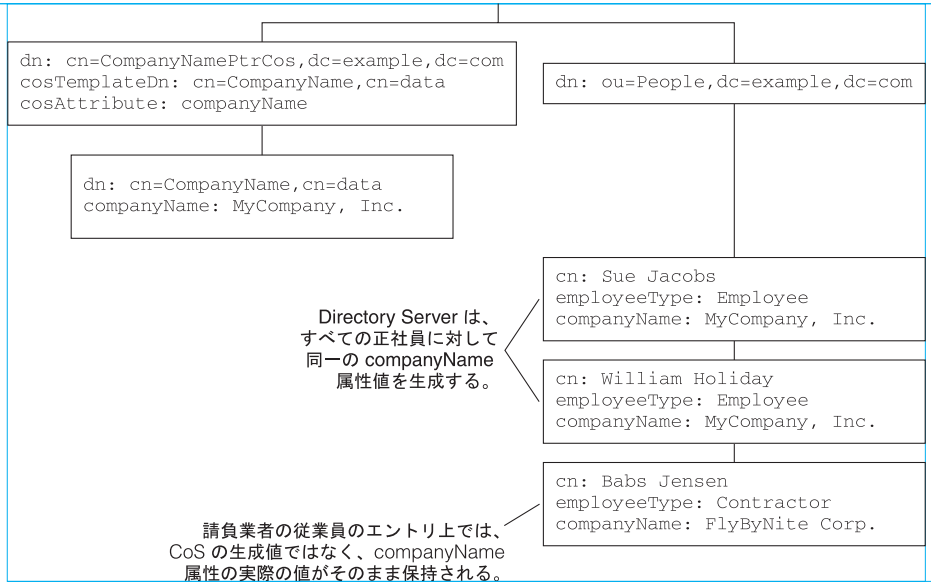


図 4–3　ポインタ CoS を使用した companyName の生成

多数のエントリが同じ値を共有する場合は、ポインタ CoS が特にうまく機能します。正社員に対する companyName の管理が容易になることで、属性値を生成するための追加の処理コストが相殺されます。深いディレクトリ情報ツリー (DIT) は、一般的な特性を共有するエントリをまとめる傾向があります。深い DIT でポインタ CoS を使用すると、ツリー内の適切な分岐に CoS 定義を配置することによって、一般的な属性値を生成できます。

エントリが自然な関係を持っている場合の CoS の使用

CoS はまた、ディレクトリデータが自然な関係を持っている場合も、データ管理の大きな利点を提供します。

すべての従業員にマネージャーがいる企業ディレクトリを考えてみます。すべての従業員がメールストップと Fax 番号を、もっとも近い管理アシスタントと共有しています。図 4–4 は、間接 CoS を使用して、マネージャーのエントリから部門番号を取得する様子を示しています。図 4–5 では、メールストップと Fax 番号が管理アシスタントのエントリから取得されています。

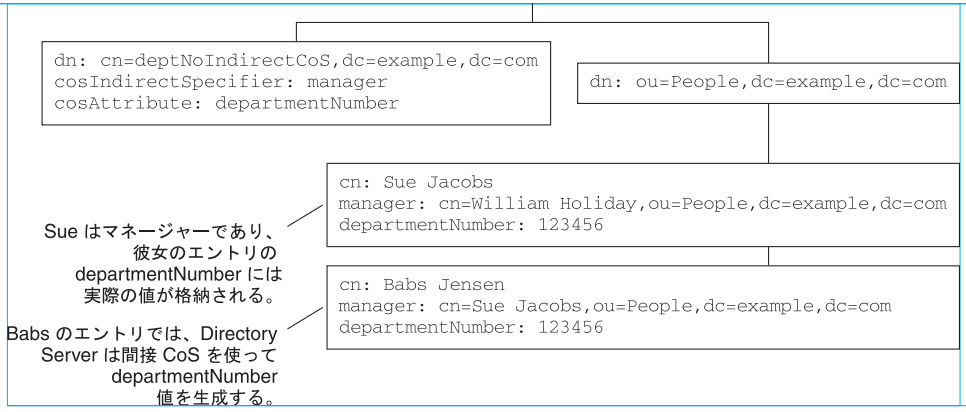


図 4–4　間接 CoS を使用した DepartmentNumber の生成

この実装では、マネージャーのエントリに departmentNumber の実際の値が含まれ、生成されているどの値よりもこの実際の値が優先されます。Directory Server は、CoS で生成された属性値からは属性値を生成しません。そのため、図 4–4 の例では、部門番号の属性値をマネージャーのエントリ上でのみ管理する必要があります。同様に、図 4–5 に示されている例では、メールストップと Fax 番号の属性を管理アシスタントのエントリ上でのみ管理する必要があります。

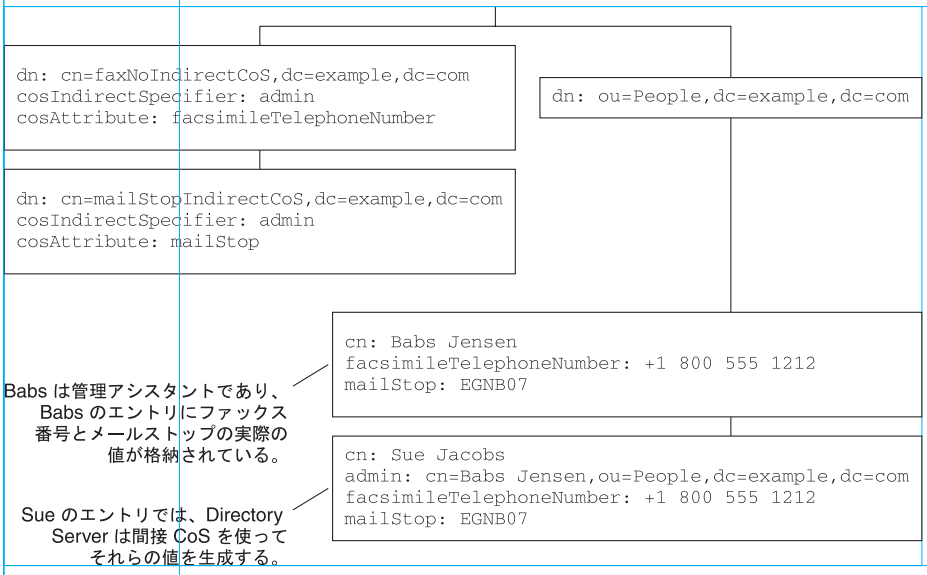


図 4–5　間接 CoS を使用したメールストップと Fax 番号の生成

単一の CoS 定義エントリを使用して、このような関係をディレクトリ内のさまざまなエントリに利用できます。

別の自然な関係に、サービスレベルがあります。顧客にスタンダード、シルバー、ゴールド、およびプラチナのパッケージを提供しているインターネットサービスプロバイダを考えてみます。顧客のディスク割り当て、メールボックスの数、前払いのサポートレベルに対する権限などは、購入したサービスレベルによって異なります。次の図は、クラシック CoS スキーマによってこの機能が可能になるようすを示しています。

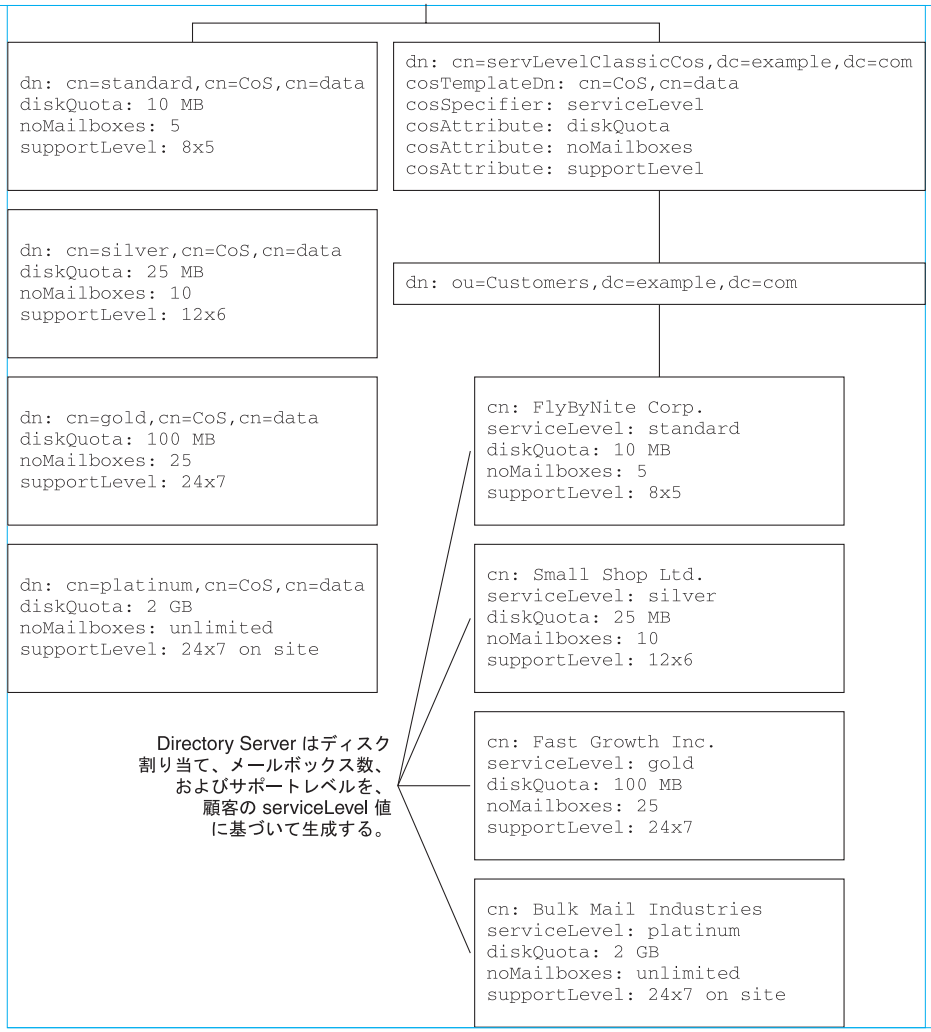


図 4–6 クラシック CoS を使用したサービスレベルデータの生成

1 つの CoS 定義が複数の CoS テンプレートエントリに関連付けられる場合があります。

過剰な CoS 定義の回避

Directory Server は、1 つのクラシック CoS 定義エントリが複数の CoS テンプレートエントリに関連付けられている場合は CoS を最適化します。Directory Server は、多数の CoS 定義が適用される可能性がある場合は CoS を最適化しません。代わりに、Directory Server は各 CoS 定義をチェックして、この定義が適用されるかどうかを判定します。この動作によって、数千の CoS 定義が存在する場合はパフォーマンスの問題が発生します。

この状況は、図 4–6 で示した例に変更がある場合に発生する可能性があります。顧客に、各顧客のサービスレベルの委任管理を提供しているインターネットサービスプロバイダを考えてみます。各顧客から、スタンダード、シルバー、ゴールド、およびプラチナのサービスレベルの定義エントリが提供されます。顧客数が 1000 に増えると、1000 個のクラシック CoS 定義が作成されることになります。どの定義が適用されるを判定するために 1000 個の CoS 定義のリストがチェックされるため、Directory Server のパフォーマンスが影響を受ける可能性があります。この種の状況で CoS を使用する必要がある場合は、間接 CoS を検討してください。間接 CoS では、顧客のエントリによって、その顧客のサービスクラス割り当てを定義するエントリが特定されます。

1 つまたは 2 つのターゲットエントリごとに別の CoS スキーマを選択することの限界に近づき始めたら、実際の値を更新することをお勧めします。CoS で生成されていない実際の値が読み取られるため、より高いパフォーマンスが実現します。

ディレクトリスキーマの設計

ディレクトリスキーマは、ディレクトリに格納できるデータの種類を記述します。スキーマ設計により、個々のデータ要素が LDAP 属性にマップされます。関連する要素は LDAP オブジェクトクラスにまとめられます。適切に設計されたディレクトリスキーマは、データ値のサイズ、範囲、および形式を制限することによって、データの整合性の維持に役立ちます。ディレクトリに含めるエントリの種類と、各エントリで使用できる属性を決定します。

Directory Server に付属の定義済みスキーマには、Internet Engineering Task Force (IETF) 標準の LDAP スキーマが含まれます。このスキーマには、サーバーの機能をサポートするための、アプリケーション固有の追加スキーマが含まれます。また、Directory Server 固有のスキーマ拡張も含まれています。定義済みのスキーマは大半のディレクトリの要件を満たしますが、対象のディレクトリに固有のニーズに対応するために、このスキーマを拡張して新しいオブジェクトクラスと属性を追加することが必要な場合もあります。

スキーマ設計のプロセス

スキーマ設計では、次のことを行う必要があります。

- デフォルトスキーマへのデータのマッピング

  既存のデータをデフォルトスキーマにマップするには、各データ要素が記述するオブジェクトの種類を識別してから、類似のオブジェクトクラスをデフォルトスキーマから選択します。group、people、organization などの共通オブジェクトクラスを使用します。データ要素にもっとも一致するオブジェクトクラスから、類似の属性を選択します。
- 一致しないデータを特定します。
- デフォルトスキーマを拡張して、残りの要件を満たす新しい要素を定義します。

  デフォルトのディレクトリスキーマで定義されているオブジェクトクラスと属性に一致しないデータ要素がある場合は、スキーマをカスタマイズできます。スキーマを拡張して、既存のスキーマに制約を追加することもできます。詳細は、「About Custom Schema」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。
- スキーマの保守を計画します。

可能であれば、デフォルトの Directory Server スキーマで定義されている既存のスキーマ要素を使用します。標準のスキーマ要素を使用することにより、ディレクトリ対応アプリケーションとの互換性を保ちやすくなります。このスキーマは LDAP 標準に基づいており、多数のディレクトリユーザーによる検討および合意を経ています。

データの整合性の維持

データの整合性を保つことにより、LDAP クライアントアプリケーションがディレクトリエントリを検索しやすくなります。ディレクトリに格納される情報の種類ごとに、その情報をサポートするために必要なオブジェクトクラスと属性を選択します。常に同じオブジェクトクラスと属性を使用します。整合性のないスキーマオブジェクトを使用すると、情報の検索が難しくなります。

次のようにすると、整合性のあるスキーマを維持できます。

- スキーマ検査を使用して、属性とオブジェクトクラスが必ずスキーマ規則にマッチしていることを確認する。

  スキーマ検査の詳細は、Chapter 12,「Directory Server Schema,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。
- 整合性のあるデータ形式を選択して適用する。

  LDAP スキーマを使用して、必要なデータを任意の属性値に格納できます。ただし、LDAP クライアントアプリケーションとディレクトリユーザーに適切な形式を選択して、DIT 内で整合性を維持するようにデータを格納することをお勧めします。Directory Server で LDAP プロトコルを使用する場合、RFC 4517 仕様で定められたデータ形式を使用してデータを表現する必要があります。

その他のディレクトリデータ関連資料

標準 LDAP スキーマおよび DIT 設計の詳細は、次のサイトを参照してください。

- RFC 4510: Lightweight Directory Access Protocol (LDAP): 技術仕様ロードマップ
  http://www.ietf.org/rfc/rfc4510.txt
- RFC 4511: Lightweight Directory Access Protocol (LDAP): プロトコル
  http://www.ietf.org/rfc/rfc4511.txt
- RFC 4512: Lightweight Directory Access Protocol (LDAP): ディレクトリ情報モデル
  http://www.ietf.org/rfc/rfc4512.txt
- RFC 4513: Lightweight Directory Access Protocol (LDAP): 認証方法とセキュリティー機構 http://www.ietf.org/rfc/rfc4513.txt
- RFC 4514: Lightweight Directory Access Protocol (LDAP): 識別名 (DN) の文字列表現
  http://www.ietf.org/rfc/rfc4514.txt
- RFC 4515: Lightweight Directory Access Protocol (LDAP): 検索フィルタの文字列表現
  http://www.ietf.org/rfc/rfc4515.txt
- RFC 4516: Lightweight Directory Access Protocol (LDAP): URL
  http://www.ietf.org/rfc/rfc4516.txt
- RFC 4517: Lightweight Directory Access Protocol (LDAP): 構文とマッチングルール
  http://www.ietf.org/rfc/rfc4517.txt
- RFC 4518: Lightweight Directory Access Protocol (LDAP): 国際化された文字列の準備
  http://www.ietf.org/rfc/rfc4518.txt
- RFC 4519: Lightweight Directory Access Protocol (LDAP): ユーザーアプリケーション用スキーマ http://www.ietf.org/rfc/rfc4519.txt
- 『Understanding and Deploying LDAP Directory Services』 T. Howes, M. Smith, G. Good. Macmillan Technical Publishing, 1999

Directory Server Enterprise Edition でサポートされているすべての RFC および標準規格の一覧は、Appendix A,「Standards and RFCs Supported by Directory Server Enterprise Edition,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide 』を参照してください。

◆ ◆ ◆ 第 5 章

5 サービスレベル契約の定義

サービスレベル契約とは、システムが特定の条件下でどのような動作をしなければならないかを決定する技術的仕様のことです。この章では、Directory Server Enterprise Edition に固有のサービス要件について説明します。また、配備がこれらの要件を満たすことを保証するために、計画フェーズの間に確認する必要があるチェックポイントを示します。

この章の内容は次のとおりです。

システム品質の識別

システム品質を識別するには、ディレクトリサービスが提供しなければならない最低限の要件を指定します。一般的に、サービス品質要件の基礎を形成するのは次のシステム品質です。

- パフォーマンス: ユーザーの負荷条件に対する、応答時間およびスループットの測定値。
- 可用性: システムのリソースやサービスにエンドユーザーがアクセスできる時間の指標であり、システムの稼働時間と表現されることもよくあります。
- 拡張性: 配備済みのシステムに容量やユーザーをあとから追加できる能力のこと。拡張性は一般的に、配備アーキテクチャーは変更せずに、システムにリソースを追加することを指します。

- セキュリティー: システムとそのユーザーの完全性を担保する各種要因の複合的な組み合わせです。セキュリティーを構成する要素には、ユーザーの認証と承認、データのセキュリティー、配備済みシステムへのセキュリティー保護されたアクセスなどがあります。
- 潜在処理能力: きわめて高いピーク負荷をリソースの追加なしで処理できるシステムの能力。潜在処理能力は可用性、パフォーマンス、および拡張性に関係する要因です。
- 保守容易性: 配備済みシステムを容易に保守できること。ここでいう保守には、システムの監視、発生する問題の修正、ハードウェアおよびソフトウェアコンポーネントのアップグレードなどが含まれます。

パフォーマンス要件の定義

パフォーマンス要件は、ディレクトリ使用状況の一般的なモデルに基づいて定義することをお勧めします。すべてのディレクトリ配備で、Directory Server は 1 つ以上のクライアントアプリケーションをサポートします。これらのアプリケーションの要件を評価する必要があります。ディレクトリに格納される情報の分量とアクセス頻度を見積もるには、これらのアプリケーションを識別し、アプリケーションが Directory Server をどのように利用するかを特定する必要があります。

クライアントアプリケーションの識別

ディレクトリにアクセスするアプリケーションと、データに対するこれらのアプリケーションのニーズは、パフォーマンス要件に大きな影響を及ぼします。クライアントアプリケーションを識別するときは、次のことを考慮します。

- Directory Server にアクセスするクライアントアプリケーションのタイプ
- これらの各アプリケーションにアクセスするユーザー数
- これらのアプリケーションが実行する操作のタイプ
- これらの操作の使用状況パターン

ディレクトリを使用する可能性がある一般的なアプリケーションには、次のものがあります。

- **White pages** などのブラウザアプリケーション: この種のアプリケーションは一般に、電子メールアドレス、電話番号、従業員名などの情報にアクセスします。
- メッセージングアプリケーション、特に電子メールサーバー: すべての電子メールサーバーは、電子メールアドレス、ユーザー名、およびルーティング情報を必要とします。サーバーの中には、ユーザーのメールボックスが格納されているディスク上の格納場所、休暇通知情報、およびプロトコル情報など、さらに高度な情報を必要とするものもあります。

- ディレクトリ対応の人事管理アプリケーション: これらのアプリケーションは、政府指定の ID 番号、自宅住所、自宅電話番号、給与詳細などのより個人的な情報を必要とします。
- セキュリティー、**Web** ポータル、個人化用アプリケーション: この種のアプリケーションは、プロファイル情報にアクセスします。

各アプリケーションが使用する情報を特定すると、いくつかのタイプのデータが複数のアプリケーションによって使用されていることが判明する場合があります。計画段階でこのような課題に取り組むことで、ディレクトリ内のデータが重複する問題を避けることができます。

ディレクトリエントリの数とサイズの決定

ディレクトリに格納されるエントリの数とサイズは、第 4 章で説明したようなデータ要件に大きく左右されます。

エントリの数とサイズを計算するときは、次のことを考慮します。

- 配備で一括インポート初期化を繰り返し実行する必要があるか。
- 必要な場合はインポートの実行頻度
- 一度にインポートされるエントリの数
- インポートされるエントリのタイプ
- サーバーが稼働したままの状態で、オンラインで初期化を実行する必要があるか

読み取り数の特定

読み取りトラフィックを見積もるときは、次のことを考慮します。

- 1 秒あたりの検索件数がどれくらいになるか
- どのようなタイプが検索されるか

  例: 一意 ID 検索、ワイルドカード検索、完全一致検索
- ピーク時の検索率はどの程度になるか
- 平均の検索率はどの程度になるか
- インデックスを使用しない検索がどの程度行われるか

  インデックスを使用しない検索とは、インデックスファイルではなくデータベースを直接検索する検索のことです。インデックスを使用しない検索が発生するのは、検索に使用されるインデックスファイルの内部で「すべての ID のしきい値」に達した場合、インデックスファイルが存在しない場合、または検索に必要な形式でインデックスファイルが構成されていない場合です。

  インデックスを使用しない検索は通常、インデックス検索よりも時間がかかります。

- 検索が特定のデータセンターまたは地理的な領域に集中しているか

  あるデータセンターが受信する検索トラフィックがほかのデータセンターに比べて多い場合、サーバーをレプリケートしてこのデータセンターに追加配置し、負荷分散を図ることを検討する余地があります。
- 検索が一日の特定の時間帯に集中しているか
- ファイアウォール内部での検索がどれくらい行われるか
- ファイアウォールの外からの検索がどれくらい行われるか

読み取りパフォーマンスがもっとも重視されるビジネス環境での配備については、第 10 章を参照してください。ここでは、読み取りトラフィックの増加に対応できる拡張性を確保しながらディレクトリサービスを配備するための提案事項を紹介しています。

書き込み数の特定

書き込みトラフィックを見積もるときは、次のことを考慮します。

- 1 秒あたりの更新件数がどれくらいになるか
- どのようなタイプが更新されるか
- ピーク時の更新率はどの程度になるか
- 平均の更新率はどの程度になるか
- 更新が特定のデータセンターまたは地理的な領域に集中しているか

  あるデータセンターが受信する更新トラフィックがほかのデータセンターと比べて多い場合、書き込み可能なサーバーをこのデータセンターに追加配置し、更新負荷を分散させることを検討する余地があります。
- 更新が一日の特定の時間帯に集中しているか

書き込みパフォーマンスがもっとも重視されるビジネス環境での配備については、第 10 章を参照してください。ここでは、書き込みトラフィックの増加に対応できる拡張性を確保しながらディレクトリサービスを配備するための提案事項を紹介しています。

許容可能な応答時間の見積もり

それぞれのクライアントアプリケーションについて、許容可能な最大の応答時間を特定します。許容可能な応答時間は、地理的条件や操作の種類によって異なる可能性があります。

許容可能なレプリケーション待ち時間の見積もり

マスターレプリカとコンシューマレプリカの間で必要な同期性のレベルを見積もります。Directory Server のレプリケーションモデルは疎整合です。これは、マスター上で更新が受理される際に、トポロジ内のほかのレプリカとの通信が不要なことを意味します。そのため、各レプリカの内容が一時的に異なっている可能性があります。これらのレプリカは時間の経過とともに収束し、最終的には各レプリカがデータの同じコピーを保持するようになります。パフォーマンス計画の一環として、レプリカが収束しなければならない最長の許容可能時間を決定します。

Directory Server 6.x には、新しい「優先順位付きレプリケーション」機能が含まれています。この機能では、特定の属性の変更をできるだけ早くレプリケートしなければならないことを指定できます。優先順位付きレプリケーションは、許容可能なレプリケーション待ち時間についての決定に影響を及ぼす場合があります。詳細については、「Prioritized Replication」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

可用性要件の定義

可用性は、ディレクトリサービスに関して合意された最低限の「稼働時間」およびパフォーマンスレベルを指します。ここでは、ディレクトリサービスがこの最小レベルのサービスを提供できなくなるあらゆる要因を障害と定義します。

可用性要件を評価するときは、次のことを考慮します。

- ディレクトリサービスは一日の特定の時間帯にのみアクセスされるか
- 読み取り操作と書き込み操作のそれぞれに対して異なる可用性要件が存在するか
- 地理的に離れた複数のサイトにまたがってサービスが提供されるか。その場合、アクセス時間に関する要件がこれらのサイト間で異なっているか
- システムの保守のためにシャットダウンできるか

  できる場合、許容可能な最長の停止時間はどのくらいか
- 移行中にシステムをシャットダウンできるか
- 停止時間が組織にもたらすコスト (損失) はどの程度か

高可用性ディレクトリサービスの配備に関する提案事項については、第 12 章を参照してください。

拡張性要件の定義

ディレクトリが拡大する過程で、サポートしなければならないサービスレベルが変化する可能性があります。システムの配備後にサービスのレベルを引き上げることは難しい場合があります。したがって、初期設計では将来の要件も考慮に入れる必要があります。

拡張性要件を定義するときは、次のことを考慮します。

- エントリ分量の増加が予測されるか
- 今後数年間で新規ユーザーの数はどれくらいになるか
- 今後数年間で、データ、ユーザー、およびクライアントアプリケーションの増加率はどの程度になるか
- 新しいビジネスプロセスの導入予定の有無

あまり早い時期に配備の拡張が必要にならないように、必要な CPU 数については多めに見積もっておきます。拡張の時期として想定しているマイルストーンと、今後の負荷の増加分を比較検討して、マイルストーンに到達するまでは十分な能力を維持できるようにします。

セキュリティー要件の定義

セキュリティー要件については、独立した解説が必要です。第 7 章で詳しく説明します。

潜在処理能力要件の定義

潜在処理能力要件を定義するときは、ディレクトリサービスのピーク時の負荷条件を見積もります。次の点を考慮してください。

- すべてのクライアントアプリケーションが動作中と仮定した場合に達する可能性がある、Directory Server への最大同時接続数
- 1 台以上のサーバーが停止したと仮定した場合に、配備内の残りのサーバーにかかると予測される負荷

保守容易性要件の定義

保守容易性要件については、第 8 章で詳しく説明します。

◆ ◆ ◆ 第 6 章

6 システム特性のチューニングとハードウェアサイジング

Directory Server Enterprise Edition を配備するには、まず特定のシステム特性が定義されている必要があります。この章では、配備の計画フェーズで解決する必要のあるシステム情報について説明します。

この章の内容は次のとおりです。

ホストシステムの特性

配備で使用するホストシステムを特定する際には、次の点を考慮してください。

- 1 つのサーバーの専用システムとするか。
- システム上でほかのアプリケーションを実行するか。実行する場合はどのようなアプリケーションを実行するか。
- それらのアプリケーションはシステムのリソースのどのくらいの割合を必要とするか。

ホストシステムの特定が完了したら、トポロジ内のホストごとにホスト名を選択します。必ず各ホストシステムが静的な IP アドレスを持つようにしてください。

ホストシステムへの物理アクセスを制限します。Directory Server Enterprise Edition に多数のセキュリティー機能が組み込まれているとしても、ホストシステムへの物理アクセスが制御されていない場合、ディレクトリのセキュリティーは危険にさらされます。

Directory Server インスタンスがネットワークのネームサービスを提供していない場合、または配備によりリモート管理を可能にするには、ネームサービスと、そのホストのドメイン名が適切に構成されている必要があります。

ポート番号

設計時に、各 Directory Server インスタンスおよび各 Directory Proxy Server インスタンスのポート番号を選択します。可能であれば、本稼働環境へのディレクトリサービスの配備後にポート番号を変更することは避けてください。

さまざまなサービスやコンポーネントに対してそれぞれ異なるポート番号を割り当てる必要があります。

- 76 ページの「Directory Server および Directory Proxy Server の LDAP および LDAPS ポート番号」
- 77 ページの「Directory Server の DSML ポート番号」
- 77 ページの「Directory Service Control Center および共通エージェントコンテナのポート番号」
- 78 ページの「Identity Synchronization for Windows のポート番号」

Directory Server および Directory Proxy Server の LDAP および LDAPS ポート番号

LDAP 接続を受け入れるためのポート番号を指定します。LDAP 通信の標準ポートは 389 ですが、ほかのポートを使用してもかまいません。たとえば、サーバーを通常のユーザーとして起動できるようにする必要がある場合には、特権のないポートを使用します。デフォルトは 1389 です。1024 より小さいポート番号では特権付きアクセスが必要になります。1024 より小さいポート番号を使用した場合、特定の LDAP コマンドを `root` として実行する必要があります。

SSL ベースの接続を受け入れるためのポート番号を指定します。SSL ベースの LDAP (LDAPS) 通信の標準ポートは 636 ですが、通常のユーザーとして実行する場合のデフォルトである 1636 など、ほかのポートを使用してもかまいません。たとえば、サーバーを通常のユーザーとして起動できるように特権のないポートが必要になる可能性があります。

特権のないポートを指定し、かつほかのユーザーがアクセスしているシステム上にサーバーインスタンスをインストールした場合、そのポートは別のアプリケーションに乗っ取られる危険にさらされる可能性があります。ほかのアプリケーションが同じアドレスとポートのペアにバインドできることになり、このため、悪意のあるアプリケーションがサーバー宛の要求を処理できる可能性があります。また、そうしたアプリケーションを使えば、認証処理時に使用されたパスワードを捕捉したり、クライアント要求やサーバー応答を変更したり、サービス拒否攻撃を仕掛けたりすることもできます。

Directory Server と Directory Proxy Server のどちらでも、サーバーが待機する IP アドレスのリストを制限できます。Directory Server には、設定属性 `nsslapd-listenhost` と `nsslapd-securelistenhost` があります。Directory Proxy Server では、`ldap-listener` および `ldaps-listener` 設定オブジェクト上に `listen-address` プロパティーがあります。待機するインタフェースのリストを指定すると、ほかのプログラムはサーバーと同じポート番号を使用できなくなります。

Directory Server の DSML ポート番号

Directory Server は、LDAP 要求を処理するだけでなく、Directory Service Markup Language v2 (DSML) で送信されてきた要求にも応答します。DSML は、クライアントがディレクトリ操作をエンコードするためのもう 1 つの方法です。Directory Server は、DSML をほかのすべての要求と同様に、同じアクセス制御とセキュリティー機能を使って処理します。

使用するトポロジに DSML アクセスが含まれている場合は、次の情報を特定します。

- DSML 要求を受信するための標準の HTTP ポート。デフォルトポートは 80 です。
- SSL が有効になっている場合は、暗号化された DSML 要求を受信するための暗号化 (HTTPS) ポート。デフォルトポートは 443 です。
- 相対 URL。これをホストとポートの末尾に追加すれば、クライアントが DSML 要求の送信時に使用する必要のある完全な URL が決まります。

DSML の設定方法については、「To Enable the DSML-over-HTTP Service」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

Directory Service Control Center および共通エージェントコンテナのポート番号

Directory Service Control Center (DSCC) は Sun Java Web コンソール向けの Web アプリケーションであり、Directory Server および Directory Proxy Server のインスタンスをユーザーが Web ブラウザ経由で管理できるようにします。あるサーバーが DSCC によって認識されるには、そのサーバーを DSCC に登録する必要があります。サーバーの登録を解除しても、それらのサーバーは依然としてコマンド行ユーティリティーを使って管理できます。

DSCC は、サーバーがインストールされたシステム上に存在している DSCC エージェントと通信します。DSCC エージェントは共通エージェントコンテナ内で実行されます。共通エージェントコンテナは、ネットワークトラフィックを各エージェントへ転送するとともに、エージェントの実行環境としてのフレームワークを提供します。

DSCC を使ってトポロジ内のサーバーを管理する予定である場合は、次の各ポート番号を特定します。

- DSCC がインストールされたシステム上の、Sun Java Web コンソール経由で DSCC にアクセスするための暗号化 HTTPS ポート。デフォルトポートは 6789 です。
- サーバーインスタンスがインストールされたシステム上の、DSCC がサーバーに対してローカルな場所にあるエージェントに共通エージェントコンテナ経由でアクセスするための管理トラフィックポート。デフォルトは 11162 です。
- 設定情報をレプリケートする予定である場合は、DSCC レジストリのインスタンスのポート番号。詳細については、`dsccsetup`(1M) のマニュアルページを参照してください。

同一システム上にすべてのコンポーネントがインストールされている場合でも、DSCC はそのエージェントとこれらのネットワークポート経由で通信します。

Identity Synchronization for Windows のポート番号

配備環境で、Microsoft Active Directory とのアイデンティティー同期を行う場合、Message Queue インスタンス用のポートが必要となります。このポートは、Active Directory との同期に関わる各 Directory Server インスタンス上で使用できる必要があります。Message Queue のセキュリティー保護されていないポートのデフォルトは 80、セキュリティー保護されたポートのデフォルトは 443 です。

また、配備計画時には、その他のインストール決定や設定決定も行う必要があります。Identity Synchronization for Windows のインストールや設定の詳細については、Part II, 「Installing Identity Synchronization for Windows,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide 』 を参照してください。

Directory Service Control Center のハードウェアサイジング

DSCC は、Web アプリケーションコンテナの内部で動作する Sun Java Web コンソールのさらに内部で、Web アプリケーションとして実行されます。また、DSCC は、設定データの格納用として独自のローカル Directory Server インスタンスも実行します。

DSCC を実行するための最小要件は、256M バイトのメモリーと 100M バイトの空きディスク容量です。ただし、最適なパフォーマンスを実現するには、少なくとも 1G バイトの DSCC 専用メモリーと数 G バイトの空きディスク容量を備えたシステム上で DSCC を実行します。

Directory Proxy Server のハードウェアサイジング

Directory Proxy Server はマルチスレッド Java プログラムとして実行され、複数のプロセッサ間で高いスケーラビリティーを実現できるように構築されています。一般に、利用可能な処理能力が大きいほどパフォーマンスも向上しますが、実際には、メモリーを追加したりディスクやネットワーク接続を高速化したりしたほうが、プロセッサを追加するよりもパフォーマンスが改善される可能性が高くなります。

仮想メモリーの設定

Directory Proxy Server は主に、処理中の情報を保持する目的でメモリーを使用します。複数のデータソースに対するいくつかの仮想ディレクトリ要求を処理することで、一時的に余分なメモリーを使用します。データソースの 1 つが LDIF ファイルであった場合、Directory Proxy Server はそのデータソースの内容をメモリー内に構築します。ただし、配備方法として推奨されていない大規模な LDIF データソースを使用するのでもないかぎり、数 G バイトのメモリーを Directory Proxy Server に割り当てれば十分です。十分なメモリーが利用可能である場合には、Directory Proxy Server 起動時の Java 仮想マシンのヒープサイズを増やすことをお勧めします。たとえば、Java 仮想マシンのヒープサイズを 1000M バイトに設定するには、次のコマンドを使用します。

```
$ dpadm set-flags instance-path jvm-args="-Xmx1000M -Xms1000M -XX:NewRatio=1"
```

このコマンドでは `-XX:NewRatio` オプションを使用していますが、これは、Sun の Java 仮想マシンに固有のオプションです。デフォルトのヒープサイズは 250M バイトです。

ワークスレッドとバックエンド接続の設定

Directory Proxy Server では、サーバーが要求を処理するためのスレッドを何個維持するかを設定できます。これを設定するにはサーバープロパティー `number-of-worker-threads` を使用します。これについては `number-of-worker-threads`(5dpconf) のマニュアルページを参照してください。経験則として、50 スレッドに、使用するデータソースごとに 20 スレッドを加算した値を設定してみてください。この数値が十分かどうかを判断するには、`cn=Work Queue,cn=System Resource,cn=`*instance-path*` ,cn=Application System,cn=DPS6.0,cn=Installed Product,cn=monitor` で Directory Proxy Server の作業キューの状態を監視します。作業キューの `operationalStatus` が `STRESSED` になっていた場合、それは、スレッドの不足している接続ハンドラが新しいクライアント要求を処理できないことを意味している可能性があります。Directory Proxy Server が追加のシステムリソースを利用できる場合には、`number-of-worker-threads` を増やすことで状況が改善される可能性があります。

また、ワークスレッドの数はバックエンド接続の数と比較しても適切な値であるべきです。バックエンド接続の数に比べて「ワークスレッドの数が多すぎる」場合、受信接続を受け入れても、それをバックエンド接続に送信することができません。そのような状況は一般に、クライアントアプリケーションにとって問題となります。

この状況が発生しているかどうかを判断するには、次のタイプのエラーメッセージがログファイルに含まれていないかチェックします。"Unable to get backend connections"。あるいは、負荷分散の cn=monitor エントリを確認します。そのエントリの totalBindConnectionsRefused 属性が null でなかった場合、バックエンド接続の不足が原因でプロキシが特定の操作を処理できなかったことになります。この問題を解決するには、バックエンド接続の最大数を増やします。各データソースのバックエンド接続の数を設定するには、そのデータソースの num-bind-limit、num-read-limit、および num-write-limit プロパティーを使用します。バックエンド接続の上限にすでに達した場合には、ワークスレッドの数を減らします。

バックエンド接続の数に比べて「ワークスレッドの数が不足している」場合には、サーバーのキュー内に多数の作業がたまり、新しい接続を処理できなくなる可能性があります。このため、クライアント接続が TCP/IP レベルで拒否されてしまい、何の LDAP エラーも返されない可能性があります。この状況が発生しているかどうかを判断するには、作業キューの cn=monitor エントリの統計を確認します。特に、readConnectionsRefused と writeConnectionsRefused は低いままであるべきです。また、maxNormalPriorityPeak 属性の値も低いままであるべきです。

Directory Proxy Server のディスク容量

Directory Proxy Server はデフォルトで、access ログ用として最大 1G バイトのローカルディスク容量を必要とし、errors ログ用としてさらに 1G バイトのローカルディスク容量を必要とします。Directory Proxy Server がクライアントアプリケーションの要求を処理する際に書き込む access ログメッセージの数を考えると、ロギングがパフォーマンス上の障害点になる可能性があります。ただし通常、本稼働環境ではロギングを有効なままにしておく必要があります。したがって、最高のパフォーマンスを実現するには、Directory Proxy Server のログを、高速な専用ディスクサブシステム上に配置します。ログ設定を調整する手順については、「Configuring Directory Proxy Server Logs」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

Directory Proxy Server のネットワーク接続

Directory Proxy Server はネットワーク集約型アプリケーションです。Directory Proxy Server は、クライアントアプリケーションから要求を受け取るたびに、複数の操作を異なるデータソースに送信する可能性があります。Directory Proxy Server とデータソース間のネットワーク接続が高速であり、十分な帯域幅があり、待ち時間も短い

ことを確認してください。また、Directory Proxy Server とクライアントアプリケーション間の接続が予期されるトラフィック量を処理できることも確認してください。

Directory Server のハードウェアサイジング

中規模から大規模の Directory Server 配備に適したハードウェアを決定するには、本番時に提供することが予期されるデータに類似したデータと、予期されるクライアントアプリケーションからのアクセスパターンに類似したアクセスパターンに基づいて、何らかのテストを行う必要があります。特定のシステム用に最適化する場合には、システムバス、周辺バス、入出力デバイス、およびサポートされているファイルシステムがどのように動作するのかを、必ず理解しておく必要があります。この知識があれば、Directory Server をサポートするように入出力サブシステムの機能をチューニングする際に、それらの機能を活用しやすくなります。Sun サービス (`http://www.sun.com/servicessolutions/`)は、要件に合ったハードウェアサイジングなど、正しい配備決定を行う際に役立つ可能性があります。

この節では、Directory Server のハードウェアサイジングへのアプローチ方法を説明します。ここでは、配備内の Directory Server 専用として使用するプロセッサの個数、メモリーの容量、ディスクの容量、およびネットワーク接続の種類を決定する際の考慮点を示します。

この節の内容は、次のとおりです。

注 – 特に明記されていないかぎり、次の各節で説明されている「サーバープロパティー」は `dsconf` コマンドで設定できます。`dsconf` の使用方法の詳細については、`dsconf`(1M) のマニュアルページを参照してください。

チューニングプロセス

パフォーマンスのチューニングは、デフォルト設定を変更して特定の配備要件が反映されるようにすることを意味します。次のプロセスフェーズのリストには、Directory Server のチューニング時に考慮すべき主要項目が記載されています。

目標の定義

配備要件に基づいて、特定の測定可能なチューニング目標を定義します。

次の各質問を検討してください。

- どのアプリケーションが Directory Server を使用しますか。
- システムの全体を Directory Server の専用にすることができますか。

  システム上でほかのアプリケーションも実行されますか。

  そうであれば、システム上で実行されるほかのアプリケーションはどれですか。
- この配備によって何件の「エントリ」が処理されますか。

  エントリ件数はどのくらいになりますか。
- Directory Server は 1 秒あたり何件の「検索」をサポートする必要がありますか。

  どのようなタイプが検索されるか
- Directory Server は 1 秒あたり何件の「更新」をサポートする必要がありますか。

  どのようなタイプが更新されるか
- どのような種類のピークの更新頻度および検索頻度が予期されますか。

  平均の更新頻度はどれくらいになると予期されますか。
- 配備がこのシステム上で一括インポートによる初期化を繰り返し必要としますか。

  そうであれば、データのインポート頻度はどのくらいであると予期されますか。何件のエントリがインポートされますか。

  エントリの種類はわかりますか。

  サーバーが稼働したままの状態で、オンラインで初期化を実行する必要があるか

このリストはすべての要素を網羅しているわけではありません。自身の目標リストがすべての要素を網羅していることを確認してください。

方法の選択
最適化をどのように実施する予定であるかを決定します。また、最適化結果をどのように測定および解析する予定であるかも決定します。

次の各質問を検討してください。

- システムのハードウェア構成を変更できるか。
- すでに存在しているハードウェアを使用したり、サーバーが稼働しているオペレーティングシステムや Directory Server のみをチューニングしたりすること以外に方法がないか。
- ほかのアプリケーションをどのような方法でシミュレートできるか。
- テスト用の典型的なデータサンプルをどのような方法で生成すべきか。
- 結果をどのような方法で測定すべきか。
- 結果をどのような方法で解析すべきか。

テストの実行
計画したテストを実行します。配備が大規模で複雑な場合、このフェーズでかなりの時間がかかる可能性があります。

結果の確認
テストした潜在的な最適化によって、プロセス開始時に定義した目標が達成されたかどうかをチェックします。

最適化によって目標が達成された場合には、その結果を文書化します。

最適化によって目標が達成されなかった場合には、Directory Server のプロファイリングと監視を行います。

プロファイリングおよび監視
潜在的な変更を適用したあとで、Directory Server の動作のプロファイリングと監視を行います。

関係するすべての動作の測定値を収集します。

グラフ化と解析
プロファイリング中や監視中に監視した動作をグラフ化および解析します。証拠を見つけたり、さらなるテストを示唆するパターンを発見したりする努力を行なってください。

「プロファイリングおよび監視」フェーズに戻って別のデータを収集する必要がある可能性もあります。

調整およびチューニング
測定値の解析結果が示唆する、さらなる潜在的な最適化を適用します。

「テストの実行」フェーズに戻ります。

結果の文書化
適用した最適化によってプロセス開始時に定義した目標が達成された場合には、その最適化を適切に文書化し、あとでその最適化を容易に再現できるようにします。

サンプルディレクトリデータの作成

Directory Server に割り当てるディスクやメモリーの容量は、ディレクトリデータに依存します。サンプルデータが LDIF 形式ですでに存在している場合には、配備のハードウェアをサイジングする際にそのデータを使用します。ここで、サンプルデータとは、配備時に使用することが予期されるデータに対応するサンプルデータを意味しており、「配備時の実際のデータ」を意味しているのではありません。実際のデータは、現実的なプライバシーに関する配慮を必要とし、サンプルデータの生成に必要となる仕様に比べて桁違いに大きくなる可能性があるため、テスト対象のすべてのケースを実施することが難しくなる可能性があります。サンプルデータに含まれるエントリの平均サイズは、配備時に予期されるサイズに近く、その属性は、配備時に予期される値に似た値を持ち、その数は、配備時に予期される比率に似た比率で存在しています。

サンプルデータに基づいて何らかの決定を下す際には、予期される増加分を考慮に入れるようにしてください。容量計画時には、現在のデータのオーバーヘッド分を含めることをお勧めします。

サンプルデータをまだ用意していない場合には、`makeldif`(1) コマンドを使ってサンプル LDIF を生成し、それを Directory Server にインポートします。第 4 章は、配備用のサンプルデータを決定する際に役立つ可能性があります。`makeldif` コマンドは Directory Server Resource Kit ツールの 1 つです。

本番時に数百万件のエントリを提供することが予期される配備では、数百万件のエントリをテスト用に読み込むのが理想的です。ただし、数百万件のエントリを読み込むことは、最初の評価目的としては現実的でない可能性もあります。まず、10,000 件のエントリ、100,000 件のエントリ、1,000,000 件のエントリなど、サンプルデータのセットをいくつか作成し、それらをインポートし、その監視結果に基づいて推定することで、さらなるテストに必要となるハードウェアを見積もります。ハードウェア要件を見積もる際には、複数のサーバーにレプリケートされるデータを準備してください。

LDIF から Directory Server 内にディレクトリデータをインポートすると、その結果として得られるデータベースファイル (インデックスを含む) は、LDIF 表現よりも大きくなります。データベースファイルはデフォルトで、*instance-path*`/db/` ディレクトリ内に格納されます。

設定すべき項目とその理由

Directory Server のデフォルト設定は典型的な小規模配備向けに定義されたものであり、その目的は、この製品を容易にインストールおよび評価できるようにすることです。この節では、中規模から大規模の配備で調整すべきいくつかの主要設定について調査します。中規模から大規模の配備では、設定をユーザーの特定の配備に適応させることでパフォーマンスを大幅に改善できることがよくあります。

Directory Server のデータベースページサイズ

Directory Server は、データの読み書きを行う際に、ページと呼ばれる固定長のデータブロックを操作します。ページサイズを増やすことにより、1 つのディスク操作で読み書きされるブロックのサイズを増やせます。

ページサイズはエントリのサイズと関係しており、パフォーマンスを考えるうえで不可欠の要素です。エントリの平均サイズが `db-page-size/4–24` (24 はページごとのバイナリツリー内部構造) より大きいことがわかっている場合は、データベースページサイズを増やす必要があります。さらに、データベースのページサイズはファイルシステムのディスクブロックサイズと一致すべきです。

Directory Server のキャッシュサイズ

Directory Server は、クライアントアプリケーションの要求にすばやく応答するように設計されています。Directory Server は、ディスクからディレクトリデータが読み取られるまで待たずにすむように、メモリー内にデータを書き込みます。データベースファイル用、ディレクトリエントリ用、および LDIF からのディレクトリデータのインポート用のキャッシュとして、どれだけのメモリーを割り当てるかを設定できます。

すべてのディレクトリデータをキャッシュするのに十分な容量の物理メモリーを割り当てることのできるハードウェア上で Directory Server を実行するのが理想的です。データが無理なく収まるようにすることで、システムがその正常動作に必要な物理メモリーを確保でき、かつファイルシステムも自身のキャッシュ処理や正常動作に必要な物理メモリーを確保できるようにすべきです。いったんデータがキャッシュに書き込まれると、ディレクトリエントリが変更されないかぎり、Directory Server はディスクに対してデータの読み書きを行う必要がなくなります。

Directory Server は、64 ビットメモリーアドレス指定をサポートしているため、64 ビットプロセッサがアドレス指定可能なサイズであれば、どのように大きな合計キャッシュサイズでも処理できます。小規模から中規模の配備では、すべてのディレクトリデータをキャッシュ内に保持できるだけのメモリーを確保できることがよくあります。これに対し、大規模配備では、すべてをキャッシュに保持することは、現実的でないか、あるいは費用効果的に問題がある可能性があります。

大規模配備では、すべてをメモリーに保持すると、副作用が生じる可能性があります。プロセスのメモリーマップをトラバースしてデータを収集する、`pmap` コマンドなどのツールを実行すると、ユーザーが気づく程度の期間、サーバープロセスが動

かなくなる可能性があります。コアファイルが非常に大きくなるため、クラッシュ時にそれらをディスクに書き込むのに数分かかる可能性があります。サーバーが突然停止されたあと再起動された場合、起動時間が長くなる可能性があります。さらに、Directory Server がチェックポイントに到達し、ダーティーキャッシュページをディスクにフラッシュしなければならない場合には、サーバーが一時的に停止し、応答しなくなる可能性もあります。キャッシュが非常に大きい場合はこの一時停止も非常に長くなり、Directory Server が停止しているものと監視ソフトウェアによって判断される可能性があります。

オペレーティングシステムレベルの入出力バッファーを使えば、パフォーマンスが向上する可能性があります。非常に大きなバッファーは、比較的小さなデータベースキャッシュの悪影響を打ち消すことができます。

キャッシュやキャッシュ設定の詳細については、Chapter 5, 「Directory Server Data Caching,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。キャッシュサイズの調整方法については、「The Basics of Directory Server Cache Sizing (`http://blogs.sun.com/DirectoryManager/resource/ds_cache_sizing.pdf`)」を参照してください。

Directory Server のインデックス

Directory Server は、ディレクトリエントリの属性値に対するインデックスを作成することで、それらの属性の検索を高速化します。属性は、さまざまな方法でインデックスが作成されるように設定できます。たとえば、インデックスを使えば、ある属性が値を持っているか、特定の値に等しい値を持っているか、あるいは特定の部分文字列を含む値を持っているかを、Directory Server がよりすばやく決定できるようになります。

インデックスを使えば、検索時のパフォーマンスが向上する可能性がありますが、書き込み時のパフォーマンスに悪影響が及ぶ可能性もあります。ある属性のインデックスが作成されると、Directory Server はその属性の値が変更されるたびにインデックスを更新しなければいけません。

Directory Server はインデックスデータをファイルに保存します。設定するインデックスが多いほど、必要なディスク容量も多くなります。Directory Server のインデックスとデータファイルはデフォルトで、*instance-path*`/db/` ディレクトリ内に格納されます。

インデックス作成やインデックス設定の詳細については、Chapter 6, 「Directory Server Indexing,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

Directory Server の管理ファイル

Directory Server のいくつかの管理ファイルは、サイズが非常に大きくなる可能性があります。そうしたファイルとしては、ディレクトリデータを含む LDIF ファイル、バックアップ、コアファイル、ログファイルなどが挙げられます。

配備によっては、Directory Server データのインポート、補助バックアップの両方の用途で LDIF を使用することができます。標準テキスト形式の LDIF を使えば、バイナリデータだけでなく文字列もエクスポートできます。大規模配備の場合、LDIF は大量のディスク容量を占有する可能性があります。たとえば、平均サイズが 2K バイトのエントリを 1 千万件含んでいるディレクトリは、LDIF 表現ではディスク上で 20G バイトを占有します。この LDIF 形式を補助バックアップとして使用する場合、このサイズの LDIF ファイルが複数個維持される可能性があります。

バイナリのバックアップファイルも、少なくとも安全を確保するためにどこかほかの場所に移動されるまでは、ディスクの容量を占有します。Directory Server ユーティリティーによって生成されるバックアップファイルは、ディレクトリデータベースファイルのバイナリコピーから構成されます。大規模配備の場合は別の方法として、Directory Server を凍結モードにしたあと、ファイルシステムのスナップショットを取ることもできます。いずれにしても、バックアップに利用可能なディスク容量が存在している必要があります。

Directory Server はデフォルトで、ログメッセージを *instance-path*`/logs/access` と *instance-path*`/logs/errors` に書き込みます。Directory Server はデフォルトで、`access` ログ用として 1G バイトのローカルディスク容量を必要とし、`errors` ログ用としてさらに 200M バイトのローカルディスク容量を必要とします。

Directory Server のロギングの詳細については、Chapter 7, 「Directory Server Logging,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

Directory Server のレプリケーション

Directory Server では、ディレクトリデータをレプリケートすることで、配備内のサーバーの可用性を高め、サーバー間の負荷分散を行えるようになっています。Directory Server では、複数の読み書き (マスター) レプリカを一緒に配備できます。

サーバーはこれを可能にするために、内部的にディレクトリデータへの変更を追跡します。同じデータが複数の読み書きレプリカ上で変更されても、Directory Server はその変更をすべてのレプリカ上で解決できます。これらの変更を追跡するためのデータは、レプリケーション時に必要とされなくなるまで、保持しておく必要があります。変更は、「パージ遅延」によって指定された期間だけ保持されます。デフォルト値は 7 日間です。ディレクトリデータに多数の変更が加えられると、このデータは非常に大きくなる可能性があります。大きな複数値属性に変更が加えられる場合は特にそうです。

データの増大するレベルはさまざまな要素によって決まるので、データの増大量を求めるための決定的な公式はありません。最善のアプローチ方法は、通常の変更をテストして増大量を測定してみることです。エントリの変更によるデータの増大に影響する要素には、次のようなものがあります。

- 変更対象のエントリのタイプと属性のタイプ。

  複数値属性だとデータの増大量が大きくなります。属性値が小さい場合は、増大量が見えやすくなります。
- エントリに適用される作業負荷。

  エントリの追加や削除を行うと、データの増大量が大きくなります。属性値を追加すると、属性値を置き換えた場合よりもデータの増大量が大きくなります。
- 変更対象のエントリ数と、各エントリ内の変更対象の属性の数。
- データベースページのサイズ。

  非常に多くの変更を行なった場合、特定のエントリがデータベースページサイズよりも大きくなる可能性があります。

パージ遅延の期間が過ぎて、エントリがもう一度変更されるまで、レプリケーションのメタデータはエントリ内に保持されたままになっています。

Directory Server のレプリケーションの詳細については、Chapter 4, 「Directory Server Replication,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

Directory Server のスレッドとファイル記述子

Directory Server はマルチスレッドプロセスとして実行され、マルチプロセッサシステム上では高いスケーラビリティーを実現できるように設計されています。Directory Server が起動時に作成する処理用スレッドの数を設定することができます。Directory Server はデフォルトで 30 個のスレッドを作成します。この値を設定するには、`dsconf`(1M) コマンドを使ってサーバープロパティー `thread-count` を調整します。

大切なのは、多数のスレッドを処理する必要性による undo 処理のオーバーヘッドを発生させることなく、スレッドをできるだけビジー状態に保つことです。すべてのディレクトリデータがキャッシュ内に収まっているかぎり、プロセッサ数の 2 倍に予期される同時更新処理数を加えた値を `thread-count` に設定すると、しばしば良好なパフォーマンスが得られます。大きなディレクトリデータセットの一部しかキャッシュ内に収まらない場合には、データがディスクから読み取られるのを Directory Server のスレッドが待たなければいけない状況がしばしば発生します。そうした場合には、利用可能なプロセッサ数の最大 16 倍という、格段に大きなスレッド数を使用すれば、パフォーマンスを改善できる可能性があります。

Directory Server はファイル記述子を使って、開いているクライアントアプリケーション接続に関係するデータを保持します。Directory Server はデフォルトで、最大 1024 個のファイル記述子を使用します。この値を設定するには、`dsconf` コマンドを

使ってサーバープロパティー file-descriptor-count を調整します。too many fds open というメッセージが errors ログに含まれている場合には、file-descriptor-count を増やすことでパフォーマンスが向上する可能性があります。ただし、Directory Server が追加のファイル記述子を開くことをシステムが許可するものと仮定しています。

file-descriptor-count プロパティーは、Windows には適用されません。

Directory Server の規模の増大

Directory Server がいったん配備されると、その使用量はおそらく増加していきます。規模増大への備えが、配備を成功させ、一貫して高レベルのサービスを提供し続けるための鍵となります。将来的に予期される規模の増大も考慮に入れた要件定義を行い、現時点で必要なシステムよりも大規模で強力なシステムを計画してください。

ディレクトリサービスは、急激にあるいは突然に規模を増大させる必要に迫られる場合があります。たとえば、ある組織用にサイジングされたディレクトリサービスが別の組織のディレクトリサービスとマージされる場合などが、そのケースに該当します。規模増大への備えを前もって行い、予期される内容を明示的に特定することによって、急激な規模増大や突然の規模増大により適切に対処できるようになります。なぜなら、予期された増大が計画された容量を超えるかどうかを前もって知ることができるからです。

最重要のチューニングヒント

基本的な推奨事項を次に示します。これらの推奨事項はほとんどの状況で適用可能です。ここに示した推奨事項は基本的に有効ですが、実際の配備への影響を理解することなしにこれらの推奨事項を適用することは避けてください。この節の目的はチェックリストを提供することであり、模範解答を提供することではありません。

1. キャッシュサイズを調整します。

   理想的なサーバーでは、利用可能な物理メモリーが十分に存在しており、Directory Server が使用するすべてのキャッシュを保持できます。さらに、ある適切な量の追加物理メモリーが、将来の規模増大を考慮して利用可能になっています。十分な物理メモリーが利用可能な場合には、エントリキャッシュサイズを十分に大きな値に設定し、ディレクトリ内のすべてのエントリを保持できるようにします。entry-cache-size サフィックスプロパティーを使用します。db-cache-size プロパティー経由でデータベースキャッシュサイズを十分に大きな値に設定し、すべてのインデックスを保持できるようにします。dn-cache-size または dn-cache-count プロパティーを使用し、DN キャッシュのサイズを制御します。

2. インデックス作成を最適化します。

   a. 不要なインデックスを削除します。予期される要求をサポートするための追加インデックスを追加します。

新しいアプリケーションからの要求をサポートする追加インデックスを追加することもあります。インデックスの追加、削除、または変更は、Directory Server の稼働中に行えます。たとえば、`dsconf create-index` や `dsconf delete-index` などのコマンドを使用します。

システムインデックスを削除しないように注意してください。システムインデックスの一覧については、「System Indexes and Default Indexes」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

ユーザーがインデックスに変更を加えると、Directory Server は徐々にデータのインデックスを作成します。`dsconf reindex` コマンドを使って Directory Server に強制的にインデックスを再生成させることもできます。

b. インデックスを使用する検索のみを許可します。

インデックスを使用しない検索は、サーバーのパフォーマンスに多大な悪影響を及ぼす可能性があります。インデックスを使用しない検索は、多くのサーバーリソースを消費する可能性もあります。

`require-index-enabled` サフィックスプロパティーを `on` に設定することで、インデックスを使用しない検索を拒否するようサーバーに強制することを検討してください。

c. `all-ids-threshold` プロパティーを使って、1 インデックスキーあたりの値の最大数を調整します。

3. `idsktune` コマンドからの推奨内容に従って、サーバーを稼働しているマシンのオペレーティングシステムをチューニングします。詳細については、`idsktune`(1M) のマニュアルページを参照してください。
4. 操作制限を調整します。

操作制限を調整することで、Directory Server がある単一の操作に過度のリソースを割り当ててしまうのを防ぐことができます。大きな容量を要求するクライアントアプリケーションに一意のバインド DN を割り当て、それらの一意のバインド DN に対して具体的なリソース制限を設けることを検討してください。

5. ディスクのアクティビティーを分散します。

大量の更新をサポートする配備では特に、Directory Server はきわめてディスク入出力集中型になる可能性があります。可能であれば、独立したコントローラを使ってこの負荷を複数のディスクに分散することを検討してください。

6. 不要なロギングを無効にします。

ディスクアクセスはメモリーアクセスよりも低速です。したがって、過度のロギングはパフォーマンスに悪影響を及ぼします。読み取り専用のサーバーインスタンス上など、監査ロギングが必要ない場合にそのロギングをオフにしておくことで、ディスクの負荷を減らします。エラーログを使って問題のトラブルシューティングを行わないときには、エラーロギングを最小レベルにしておきます。また、ログファイルを専用のディスク上や、レプリケーション変更ログ用のディスクなど、使用頻度の少ないディスク上に配置することによっても、ロギングの影響を減らせます。

7. 多数の更新をレプリケートする場合には、適切なレプリケーションアグリーメントプロパティーを調整することを検討してください。

   プロパティーは `transport-compression`、`transport-group-size`、および `transport-window-size` です。

8. Solaris システム上では、データベースホームディレクトリを `tmpfs` ファイルシステムに移動します。

   `db-env-path` プロパティーで指定されるデータベースホームディレクトリは、Directory Server がデータベースキャッシュバッキングファイルを格納する場所を示します。データファイルはデフォルトでは、*instance-path*`/db` 内に存在し続けます。

   データベースキャッシュバッキングファイルを `tmpfs` ファイルシステム上に格納すれば、システムがデータベースキャッシュバッキングファイルをディスクに繰り返しフラッシュしなくなります。したがって、更新時のパフォーマンス障害の発生を回避できます。場合によっては、検索時のパフォーマンス障害の発生も回避できます。データベースのキャッシュメモリーは Directory Server のプロセス空間にマップされます。システムは基本的に、`tmpfs` ファイルシステム内のキャッシュメモリーとバッキングファイルの保持に使用されるメモリーを共有します。したがって、必要メモリー容量の点で基本的に何のコストも払うことなしに、パフォーマンスの向上を図れます。

   この最適化に関連する主なコストは、ホストマシンの再起動後にデータベースキャッシュを再構築する必要がある、という点です。ソフトウェアまたはハードウェアでの障害発生時にのみ再起動が発生することが予期される場合、このコストはおそらく避けて通ることはできません。そのような障害が発生すれば、いずれにしてもデータベースキャッシュを再構築する必要があります。

9. ソフトウェアまたはハードウェアでの障害発生時に更新が失われてもかまわない場合には、トランザクションバッチを有効にします。

   トランザクションバッチを有効にするには、サーバープロパティー `db-batched-transaction-count` を設定します。

   トランザクションログが更新されるたびに、更新データが失われないように同期処理が続いて実行されます。トランザクションバッチを有効にすると、いくつかの更新が一緒にまとめられてから、トランザクションログに書き込まれます。同期処理が実行されるのは、バッチの全体がトランザクションログに書き込まれるときだけです。したがって、トランザクションバッチを使えば、更新時のパフォーマンスを大幅に改善できます。この改善にはトレードオフがあります。そのトレードオフとは、クラッシュした時点でトランザクションログにまだ書き込まれていない更新データは失われてしまう、ということです。

注 – トランザクションバッチを有効にすると、ソフトウェアまたはハードウェアでの障害発生時に最大 db-batched-transaction-count - 1 件の更新が失われます。この損失が発生するのは、Directory Server が、バッチが満たされるまでの間か 1 秒間、どちらか短いほうだけ待機したあとで、内容をトランザクションログに、したがってディスクに、フラッシュするからです。

更新が失われると困る場合には、この最適化を使用「しない」でください。

10. 参照の完全性プラグインを設定して完全性チェックを遅延させます。

    参照の完全性プラグインは、エントリが変更されたりディレクトリから削除された場合に、それらのエントリへのすべての参照が間違いなく更新されるようにします。この処理はデフォルトでは、削除操作に対する応答がクライアントに返される前に、同期的に実行されます。この更新が非同期で実行されるようにプラグインを設定できます。ref-integrity-check-delay サーバープロパティーを使用します。

クライアントアプリケーション負荷のシミュレーション

Directory Server のパフォーマンスを測定するには、サーバーを準備したあと、本番時に予期されるタイプのクライアントアプリケーショントラフィックをそのサーバーに対して与えます。本番時に発生するクライアントアプリケーションのアクセスパターンタイプの再現性が高まるほど、ハードウェアのサイジングと Directory Server の設定の精度も高くなります。

Directory Server Resource Kit には、基本的なテストを行う際に使用可能な authrate(1)、modrate(1)、および searchrate(1) コマンドが含まれています。これらのコマンドを使えば、使用するディレクトリサービスがサポート可能なバインド頻度、変更頻度、および検索頻度を測定できます。

SLAMD を使って複雑で現実的なクライアントアクセスのシミュレーション、測定、およびグラフ化を行うこともできます。SLAMD 分散負荷生成エンジン (SLAMD) は、ネットワークベースのアプリケーションの負荷テストを行なったりそのパフォーマンスを解析したりするために設計された、Java アプリケーションです。これは当初、LDAP Directory Server のベンチマーク測定およびパフォーマンス解析用として、Sun Microsystems, Inc. によって開発されました。SLAMD は、OSI が承認したオープンソースライセンスである Sun Public License のもとでオープンソースアプリケーションとして公開されています。SLAMD についての情報を入手するには、http://www.slamd.com/ を参照してください。SLAMD は java.net プロジェクトとしても公開されています。https://slamd.dev.java.net/ を参照してください。

Directory Server とプロセッサ

Directory Server は複数のプロセッサが搭載されたシステム上で動作するように開発されたマルチスレッドプロセスであり、そのパフォーマンスはほとんどの場合、割り当てるプロセッサの数を増やすにつれて線形的に増加していきます。多数のプロセッサが搭載されたシステム上で Directory Server を実行する場合、Directory Server がサーバー操作を処理するために起動するスレッドの数を表すサーバープロパティー `thread-count` を、`dsconf` コマンドを使って調整することを検討してください。

ただし、特定のディレクトリ配備では、プロセッサを追加してもパフォーマンスがあまり改善されない可能性があります。検索、インデックス、およびレプリケーションで要求の厳しいパフォーマンス要件を扱うときは、解決策の一部として、負荷分散やディレクトリプロキシなどの技術を検討します。

Directory Server とメモリー

必要メモリー量に大きな影響を与える要因は、次のとおりです。

- Directory Server のデータベースキャッシュ、エントリキャッシュ、およびインポートキャッシュの設定
- ピーク時のレプリケーション負荷
- ピーク時のクライアントアプリケーション負荷
- `file-descriptor-count` と `thread-count` のサーバー設定
- オペレーティングシステム、システム上で実行されるほかのアプリケーション、およびシステム管理アクティビティーのオーバーヘッド

Directory Server の実行に必要なメモリーのサイズを見積もるには、Directory Server Resource Kit コマンドや SLAMD などによって生成されたアプリケーション負荷など、本番時と同様の負荷のかかったシステム上の特定の Directory Server 設定で必要とされるメモリーを見積もります。

Directory Server プロセスのサイズを測定する前に、サーバーを起動してからしばらく放置し、通常処理時またはピーク処理時と同様にエントリキャッシュが満たされるようにします。すべてをキャッシュメモリーに格納するだけの容量が存在する場合、ディレクトリ内のすべてのエントリを読み取ってエントリキャッシュを満たすことで、この Directory Server のウォームアップ期間を短縮することができます。すべてをキャッシュメモリーに格納するだけの容量が存在しない場合には、実際の場合と同様にキャッシュが満たされるまで、通常処理またはピーク処理のパターンを使ってクライアントアクセスのシミュレーションをしばらく実行します。

サーバーが平衡状態に達したら、Solaris や Linux 上の `pmap`、Windows タスクマネージャーなどのユーティリティーを使って Directory Server プロセスが使用しているメモリーを測定できます。このプロセスは、UNIX システム上では `ns-slapd`、Windows

システム上では slapd.exe になります。詳細については、pmap(1) のマニュアルページを参照してください。通常処理時、ピーク処理時の両方のプロセスサイズを測定したあとで、使用すべきメモリーの量を決定してください。

システム管理に必要なメモリーとシステム自体に必要なメモリーの量を、必ず見積もりに追加してください。オペレーティングシステムのメモリー要件は、システムの構成によって大きく変わる可能性があります。したがって、使用するオペレーティングシステムの実行に必要なメモリーの見積もりは、実験的に行う必要があります。システムのチューニングが完了したら、メモリーの使用量を監視し、その量を見積もりに追加します。Solaris の vmstat および sar コマンド、Windows のタスクマネージャーなどのユーティリティーを使えば、メモリーの使用量を測定できます。

Directory Server の実行時に、パフォーマンスに悪影響を及ぼす継続的なページスワッピングが発生しないだけのメモリーを、最低限提供するようにしてください。サポート対象外ですが Solaris システム用として別途入手可能な MemTool などのユーティリティーは、メモリーが実行中のアプリケーションによってどのように使用され、それらのアプリケーションにどのように割り当てられているかを監視する際に役立つ可能性があります。

システムに追加のメモリーを搭載できない場合に、これまでどおり継続的なページスワップを監視し続けるには、データベースキャッシュやエントリキャッシュのサイズを減らします。heap-high-threshold-size および heap-low-threshold-size サーバー設定を使ってメモリーの使用量を調整することもできますが、ヒープしきい値メカニズムを使うのは最後の手段と考えてください。Directory Server がヒープメモリーを解放するためにほかの処理を遅延させる必要が生じると、パフォーマンスが低下します。

Red Hat Linux システムの場合、/proc/sys/vm/swappiness パラメータを調整して、カーネルがどの程度積極的にメモリーをスワップアウトするかをチューニングできます。swappiness の値が大きいとカーネルがスワップアウトするデータ量が大きいことを意味し、swappiness の値が小さいとカーネルがスワップスペースをできるかぎり使用しないようにすることを意味します。したがって、swappiness 設定値を少なくすることにより、カーネルがメモリーに保持するサーバープロセスが増えてスワップアウトするまでの時間が長くなるので、Directory パフォーマンスが改善される場合があります。システムが単一の Directory Server インスタンス専用である場合は、swappiness をゼロに設定します。システム上で複数の重い処理または複数の Directory Server の並列インスタンスを実行している場合は、swappiness の設定をいろいろ変えて Directory パフォーマンスをテストしてみてください。

Directory Server とローカルディスク容量

ディスクの使用と I/O 能力は、パフォーマンスに強く影響します。多数の変更をサポートする配備では特に、ディスクサブシステムが入出力の障害点になる可能性があります。この節では、Directory Server インスタンスの全体的なディスク容量を見積もるために推奨される方法を示します。

注 – Directory Server やそれがアクセスするデータをネットワークディスク上にインストール「しない」でください。

Directory Server ソフトウェアは、ネットワークに接続されたストレージを NFS、AFS、または SMB 経由で使用することをサポートしません。設定、データベース、およびインデックスファイルはすべて、インストールを終えてからも、常にローカルストレージ上に配置する必要があります。ログファイルはネットワークディスク上に格納できます。

必要なローカルディスク容量に大きな影響を与える要因は、次のとおりです。

- ディレクトリエントリの数
- エントリの平均サイズ
- サーバーデータベースのページサイズの設定 (ディレクトリデータをインポートする場合)

  データベースのページサイズを調整するには、`nsslapd-db-page-size` 属性を設定します。詳細については、85 ページの「Directory Server のデータベースページサイズ」を参照してください。
- ディレクトリデータ上で維持されるインデックスの数
- 格納される LDIF、バックアップ、ログ、およびコアファイルのサイズ

インデックスの設定、データベースのページサイズの調整、およびディレクトリデータのインポートが完了したら、*instance-path/* の内容のサイズを読み取り、それに予期される LDIF、バックアップ、ログ、およびコアファイルのサイズを追加することで、インスタンスに必要なディスク容量を見積もることができます。また、特にピーク処理時に、測定したサイズがどのくらい増加することが予期されるかを見積もります。ログレベルやログサイズをデバッグ目的で増やす必要が生じた場合のために、数 G バイトの追加容量を `errors` ログ用として必ず残すようにしてください。

ディレクトリデータに必要なディスクの見積もりは、推定によって行える場合があります。本番時に予期されるのと同量のデータを使って Directory Server に負荷をかけることが現実的ではない場合には、84 ページの「サンプルディレクトリデータの作成」で示唆したように、より小さなサンプルデータセットに基づいて推定します。使用するディレクトリデータの量が本番時より少ない場合、ほかの測定値についても推定する必要があります。

ローカルディスクがどのくらい高速でなければいけないかを決定する要因は、次のとおりです。

- レプリケーショントラフィックの量など、維持される更新のレベル
- ディレクトリデータが主にキャッシュ内、ディスク上のどちらに存在するか
- アクセスロギングとエラーロギングで使用するログレベル、および監査ログを有効にするかどうか
- ディレクトリデータ、ログ、およびトランザクションログ (更新用) をそれぞれ専用のディスクサブシステム上に配置できるかどうか
- Directory Server がオンライン、オフラインのいずれの状態のときにバックアップを実行するか

通常の運用環境で、使用されるディスクをいっぱいにしないでください。Solaris `iostat` コマンドなどのツールを使えば、潜在的な入出力障害を特定できます。

ディスクのスループットを増やすには、ディスクサブシステム間でファイルを分散させます。トランザクションログ (`dsconf set-server-prop db-log-path:`*/transaction/log/path*)、データベース (`dsconf create-suffix --db-path`*/suffix/database/path suffix-name*)、およびログファイル (`dsconf set-log-prop path:`*/log/file/path*) にそれぞれ専用のディスクサブシステムを割り当てることを検討してください。さらに、Solaris `tmpfs` ファイルシステムなどのメモリーベースのファイルシステム上に、データベースキャッシュファイルを配置することを検討してください。そのようなファイルシステムでは、利用可能なメモリーが使い果たされた場合にのみファイルがディスクにスワップされます (例: `dsconf set-server-prop db-env-path:/tmp`)。メモリーベースのファイルシステム上にデータベースキャッシュファイルを配置する場合には、システムの容量が不足してそのファイルシステムの全体をメモリー内に保持できなくなることのないよう、注意する必要があります。

スループットをさらに増やすには、複数のディスクを RAID 構成にして使用します。Sun StorEdge™ 製品で提供されているような大容量で不揮発性 I/O バッファーや高パフォーマンスのディスクサブシステムによって、Directory Server のパフォーマンスや稼動時間は大きく向上します。Solaris 10 システムの場合、ZFS を使用してもパフォーマンスが改善する可能性があります。

Directory Server とネットワーク接続

Directory Server はネットワーク集約型アプリケーションです。次の式を使えば、理論的な最大スループットを見積もることができます。この式ではレプリケーションのトラフィックが考慮されていない点に注意してください。

```
max. throughput = max. entries returned/second x average entry size
```

Directory Server はピーク時に毎秒 5000 件の検索に応答する必要があり、またサーバーは検索ごとに 1 つのエントリを返すものとします。エントリの平均サイズは 2000 バイトです。理論的な最大スループットは 10M バイトつまり 80M ビットになります。ただし、レプリケーションは考慮していません。しかし 80M ビットの方が、100M ビット Ethernet アダプタ 1 つで実現するスループットよりも大きくなる傾向があります。Directory Server インスタンスのネットワーク可用性を改善するには、システムにより高速な接続を装備するか、複数のネットワークインタフェースを装備してください。Directory Server は、同一プロセス内で複数のネットワークインタフェース上で待機できます。

注 – 先の例では、ディレクトリの読み取り時または検索時にクライアントアプリケーションが「すべての」属性を要求するものと仮定していました。一般に、「必要な」属性のみを要求するようにクライアントアプリケーションを設計すべきです。

負荷分散のために、同じネットワーク上で Directory Server をクラスタにする場合は、レプリケーションで新たに生じる負荷がネットワークインフラストラクチャーで対応できることを確認します。広域ネットワーク上でマルチマスターレプリケーションを計画する場合には、その構成のテストを行い、その接続が最小限の待ち時間とゼロに近いパケット損失で十分なスループットを提供できることを確認してください。長い待ち時間と大量のパケット損失はどちらも、レプリケーションの速度を低下させます。さらに、レプリケーショントラフィックがロードバランサを通過するようなトポロジは避けてください。

クライアントが使用可能な Directory Server リソースの制限

Directory Server のデフォルトの設定では、クライアントアプリケーションが必要以上の Directory Server リソースを使用できるようになっています。

次のようなリソースの使い方をすると、ディレクトリのパフォーマンスに悪影響が及ぶ可能性があります。

- 多数の接続を開いたあと、それらをアイドル状態や未使用状態のまま放置する
- インデックスを使用しない高コストの検索を不必要に起動する
- 予定外の巨大なバイナリ属性値を格納する

配備状況によっては、デフォルトの設定を変更すべきでないことがあります。Directory Server のチューニングが不可能な配備では、Directory Proxy Server を使用することで、リソースの制限とサービス拒否攻撃への防備を行なってください。

配備状況によっては、Directory Server の 1 つのインスタンスが、ユーザーメールアプリケーションなどのディレクトリクライアントやメッセージングサーバーといった、クライアントアプリケーションをサポートしなければならないことがあり

ます。そのような状況では、バインド DN ベースのリソース制限を使用することで、ディレクトリ集約型アプリケーションに対して個別の制限を設けることを検討してください。ある特定のアカウントの制限を調整するには、そのエントリ上の属性 `nsSizeLimit`、`nsTimeLimit`、`nsLookThroughLimit`、および `nsIdleTimeout` を設定します。個々のアカウントのリソース制限を制御する方法については、「Setting Resource Limits For Each Client Account」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

表 6–1 では、リソース制限のグローバル値を設定するパラメータについて説明します。 表 6–1 の制限は、Directory Manager ユーザーには適用されません。したがって、クライアントアプリケーションが決して Directory Manager ユーザーとして接続することのないようにしてください。

表 6–1 クライアントアプリケーション専用リソースに対する推奨のチューニング方法

チューニングパラメータ	説明
サーバープロパティー `idle-timeout`	Directory Server がアイドル状態のクライアント接続を閉じるまでの時間を、秒単位で設定します。ここで、「アイドル状態」とは、接続が開いたままであるにもかかわらず、何の処理も要求されていない状態を意味します。デフォルトでは、何の時間制限も設定されていません。 このサーバープロパティーを設定するには、`dsconf set-server-prop` コマンドを使用します。 メッセージングサーバーなどの一部のアプリケーションは、トラフィックが少ないときにはアイドル状態のままであるが閉じるべきではないような一連の接続を開く可能性があります。この場合、そのアプリケーションをサポートする専用のレプリカを用意するのが理想的です。それが不可能な場合には、バインド DN ベースの個別制限を検討してください。 いずれにしても、この値を十分大きな値に設定して、ほかのアプリケーションが開いていることを期待するような接続を閉じないようにするとともに、この値を十分小さな値に設定して、接続を不必要にアイドル状態のままにできないようにしてください。たとえば、これを 7200 秒つまり 2 時間に設定することを検討してください。

表 6–1 クライアントアプリケーション専用リソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ	説明
属性 `dn: cn=config` の属性 `nsslapd-ioblocktimeout`	Directory Server が停止されたクライアント接続を閉じるまでの時間を、ミリ秒単位で設定します。ここで、「停止された」とは、クライアントへの出力送信時、クライアントからの入力読み取り時のいずれかにサーバーがブロックされることを意味します。 この属性を設定するには、`ldapmodify` コマンドを使用します。 特にサービス拒否攻撃の標的になっているような Directory Server インスタンスの場合、この値をデフォルトの 1,800,000 ミリ秒つまり 30 分より小さい値に設定することを検討してください。
サーバープロパティー `look-through-limit`	検索時に一致するかどうかをチェックする候補エントリの最大数を設定します。 このサーバープロパティーを設定するには、`dsconf set-server-prop` コマンドを使用します。 メッセージングサーバーなどの一部のアプリケーションは、ディレクトリの全体を検索する必要がある可能性があります。この場合、そのアプリケーションをサポートする専用のレプリカを用意するのが理想的です。それが不可能な場合には、バインド DN ベースの個別制限を検討してください。 いずれにしても、この値をデフォルトの 5000 エントリより小さい値に設定することを検討してください。ただし、`search-size-limit` のしきい値を下回ってはいけません。
属性 `dn: cn=config` の属性 `nsslapd-maxbersize`	BER (Basic Encoding Rules) に従ってエンコードされた ASN.1 受信メッセージの最大サイズを、バイト単位で設定します。Directory Server は、この制限よりもサイズの大きいエントリを追加する要求を拒否します。 この属性を設定するには、`ldapmodify` コマンドを使用します。 ディレクトリデータの最大エントリサイズを正確に予測できる場合には、この値をデフォルトの 2097152 つまり 2M バイトから、予測したディレクトリエントリの最大値に変更することを検討してください。 更新に対する 2 番目に大きなサイズ制限は、トランザクションログファイルのサイズ `nsslapd-db-logfile-size` です。このサイズはデフォルトで 10M バイトです。

表 6–1 クライアントアプリケーション専用リソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ	説明
サーバープロパティー `max-threads-per-connection-count`	1 クライアント接続あたりの最大スレッド数を設定します。 このサーバープロパティーを設定するには、`dsconf set-server-prop` コマンドを使用します。 メッセージングサーバーなどの一部のアプリケーションは、一連の接続を開き、それらの各接続上で多数の要求を発行する可能性があります。この場合、そのアプリケーションをサポートする専用のレプリカを用意するのが理想的です。それが不可能な場合には、バインド DN ベースの個別制限を検討してください。 一部のアプリケーションが 1 つの接続で多数の要求を実行することが予期される場合には、この値をデフォルトの 5 から増やすことを検討してください。ただし、これを 10 を超える値にまで増やしてはいけません。通常は、1 スレッドあたり 10 個を超えるスレッドを指定しないでください。
サーバープロパティー `search-size-limit`	検索要求への応答として Directory Server が返すエントリの最大数を設定します。 このサーバープロパティーを設定するには、`dsconf set-server-prop` コマンドを使用します。 メッセージングサーバーなどの一部のアプリケーションは、ディレクトリの全体を検索する必要がある可能性があります。この場合、そのアプリケーションをサポートする専用のレプリカを用意するのが理想的です。それが不可能な場合には、バインド DN ベースの個別制限を検討してください。 いずれにしても、この値をデフォルトの 2000 エントリから下げることを検討してください。

表 6–1 クライアントアプリケーション専用リソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ	説明
サーバープロパティー `search-time-limit`	Directory Server が 1 つの検索要求を処理する時間の限度を秒単位で設定します。 このサーバープロパティーを設定するには、`dsconf set-server-prop` コマンドを使用します。 メッセージングサーバーなどの一部のアプリケーションは、非常に大規模な検索を実行する必要がある可能性があります。この場合、そのアプリケーションをサポートする専用のレプリカを用意するのが理想的です。それが不可能な場合には、バインド DN ベースの個別制限を検討してください。 いずれにしても、この値は、配備要件を満たす範囲内でできるだけ低く設定してください。デフォルト値の 3600 秒つまり 1 時間は、多くの配備では必要以上に大きな値です。600 秒つまり 10 分を最適化テストの出発点として使用することを検討してください。

Directory Server が使用するシステムリソースの制限

表 6–2 では、Directory Server インスタンスによるシステムリソースやネットワークリソースの使用方法をチューニングするために使用可能なパラメータについて説明します。

表 6–2 システムリソースに対する推奨のチューニング方法

チューニングパラメータ	説明
属性 dn: cn=config の属性 nsslapd-listenhost	Directory Server が待機する IP インタフェースのホスト名を設定します。この属性は複数の値を持つことができます。 この属性を設定するには、ldapmodify コマンドを使用します。 デフォルト動作は、すべてのインタフェース上で待機することです。このデフォルト動作は、冗長性のあるネットワークインタフェースを使って可用性やスループットを高めるような、高ボリュームの配備に適しています。 マルチホームシステム上に配備する場合や、IPv4、IPv6 の各プロトコルをそれぞれ個別のインタフェースを使ってサポートするようなシステム上で IPv4 または IPv6 トラフィックのみを待機する場合に、この値の設定を検討してください。SSL 使用時には、nsslapd-securelistenhost の設定を検討してください。
サーバープロパティー file-descriptor-count	Directory Server が使用できるファイル記述子の最大数を設定します。 このサーバープロパティーを設定するには、dsconf set-server-prop コマンドを使用します。 デフォルト値は、Directory Server インスタンスが作られる時に、システム上の 1 つのプロセスに許可されるファイル記述子の最大数です。その最大値は、システム上の 1 つのプロセスに許可されるファイル記述子の最大数に対応します。詳細については、オペレーティングシステムのマニュアルを参照してください。 Directory Server はファイル記述子を使用することで、クライアント接続を処理し、ファイルを内部的に維持します。利用可能なファイル記述子の数が十分でないために Directory Server が新しい接続の待機を停止する場合があることがエラーログから判明した場合、この属性の値を増やせば、Directory Server が同時に処理できるクライアント接続の数が増える可能性があります。 システム上で利用可能なファイル記述子の数を増やした場合には、この属性の値もそれに応じて設定してください。このプロパティーの値は、システム上で利用可能なファイル記述子の最大数と等しいかそれより小さくなるようにしてください。

表 6–2 システムリソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ		説明
属性 dn: cn=config の属性 nsslapd-nagle		TCP パケットの送信をソケットレベルで遅延させるかどうかを設定します。 この属性を設定するには、ldapmodify コマンドを使用します。 ネットワークのトラフィックを減らす必要がある場合には、これを on に設定することを検討してください。

表 6–2 システムリソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ	説明

属性
 	
dn: cn=config の属性 nsslapd-reservedescriptors
	Directory Server がインデックス作成やレプリケーションなどの内部処理を管理するために維持するファイル記述子の数を設定します。そのようなファイル記述子は、「クライアント接続の処理用として使用することができなくなります」。
	
	この属性を設定するには、ldapmodify コマンドを使用します。次のすべてに該当する場合には、この属性の値をデフォルトの 64 から増やすことを検討してください。
	■ Directory Server が 10 個を超えるコンシューマにレプリケートするか、Directory Server が 30 個を超えるインデックスファイルを維持する。
	■ Directory Server が多数のクライアント接続を処理する。
	■ Directory Server がクライアント接続に関係「しない」処理を行うためのファイル記述子が不足していることを、エラーログ内のメッセージが示唆している。
	
	確保済みファイル記述子の数が増えると、クライアント接続の処理に利用可能なファイル記述子の数が減ることに注意してください。この属性の値を増やす場合には、システム上で利用可能なファイル記述子の数を増やし、file-descriptor-count の数を増やすことを検討してください。
	
	確保すべきファイル記述子の数を初めて見積もる目的でこの属性を変更することにした場合、次の式に従って nsslapd-reservedescriptors の値を設定してみてください。
	
	`20 +`
	`4 * (number of databases) +`
	`(total number of indexes) +`
	`(value of nsoperationconnectionslimit) *`
	`(number of chaining backends) +`
	`ReplDescriptors +`
	`PTADescriptors +`
	`SSLDescriptors`
	
	ここで、*ReplDescriptors* は、レプリケーションが使用される場合は、サプライヤレプリカの数に 8 を加えたものになります。*PTADescriptors* は、PTA (Pass Through Authentication) プラグインが有効になっている場合は 3、それ以外の場合は 0 になります。*SSLDescriptors* は、SSL が使用される場合は 5、それ以外の場合は 0 になります。

表 6–2 システムリソースに対する推奨のチューニング方法 (続き)

チューニングパラメータ	説明
	インスタンスがサフィックスごとに 2 つ以上のデータベースを使用するように設定されているのでないかぎり、データベースの数はインスタンスのサフィックスの数と同じになります。実験的なテストを通じて見積もり内容を確認してください。
属性 dn: cn=config の属性 nsslapd-securelistenhost	Directory Server が SSL 接続を待機する IP インタフェースのホスト名を設定します。この属性は複数の値を持つことができます。 この属性を設定するには、ldapmodify コマンドを使用します。 デフォルト動作は、すべてのインタフェース上で待機することです。この属性は、nsslapd-listenhost と同様に考えてください。
サーバープロパティー max-thread-count	Directory Server が使用するスレッドの数を設定します。 このサーバープロパティーを設定するには、dsconf set-server-prop コマンドを使用します。次のいずれかに該当する場合には、このプロパティーの値を調整することを検討してください。 ■ クライアントアプリケーションが、更新や複雑な検索などの時間のかかる処理を、同時に数多く実行する。 ■ Directory Server が多数の同時クライアント接続をサポートする。 マルチプロセッサシステムは、シングルプロセッサシステムよりも大きなスレッドプールを維持できます。この属性の値を最適化する際の最初の見積もりとしては、プロセッサ数の 2 倍、20 + 同時更新数のいずれかを使用してください。 また、1 クライアント接続あたりの最大スレッド数 max-threads-per-connection-count を調整することも検討してください。クライアント接続を処理するこれらのスレッドの最大数が、システム上で利用可能なファイル記述子の最大数を超えることはできません。場合によっては、この属性の値を増やすよりも「減らす」ほうが有効である可能性もあります。 実験的なテストを通じて見積もり内容を確認してください。その結果は、特定の配備状況だけでなく、サーバーが稼働しているシステムの状況にも依存します。

Directory Server の基本的なサイジングの例: ディスクとメモリーの要件

この節では、配備用として Directory Server のディスクおよびメモリー要件のサイジングを行う際の初期手順を示す例を取り上げます。この例で使用するシステムは、何か特別な理由があって選択したわけではなく、サイジングタスクをすばやく完了できるだけの処理能力とメモリーを備えているという理由で選択しただけです。これは必ずしも、業務用の推奨システムを表しているとはかぎりません。ただし、これを参考にすれば、本番システムで必要となるメモリーやディスクの容量に関する洞察が得られます。

システムの特性

次のシステム情報は、Solaris Management Console (`smc`) を使って監視したものです。

- 2 個の AMD64 CPU (2.2 GHz)
- Solaris 10 オペレーティングシステム
- 4G バイトの物理メモリー
- 40G バイトのスワップ
- Directory Server インストール前の使用済み物理メモリー: 700M バイト
- Directory Server インストール前の空き物理メモリー: 3400M バイト
- CPU 使用率: 1 %
- ローカルディスク: ロギング機能付き UFS としてフォーマットされた 1 つのパーティション

この例では、システムは Directory Server 専用でした。その他のユーザーは一人もログインしておらず、デフォルトのシステムプロセスのみが実行されていました。

Directory Server インスタンスの準備

zip ディストリビューションを展開したあと、Directory Server ソフトウェアをローカルディスク上にインストールします。

```
$ ./dsee_deploy install -c DS -i /local
```

便宜上、環境変数を次のように設定します。

```
$ export PATH=/local/ds6/bin:/local/dsrk6/bin:/local/dsee6/bin:${PATH}
$ export DIRSERV_PORT=1389
$ export LDAP_ADMIN_PWF=~/.pwd
```

ソフトウェアのインストールと環境変数の設定が完了したら、LDAP、LDAPS のそれぞれのデフォルトポートを使って Directory Server インスタンスを作成します。

```
$ dsadm create -p 1389 -P 1636 /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$ du -hs /local/ds
610K   /local/ds
```

サフィックスを作成するまでは、Directory Server インスタンスの使用ディスク容量は 1M バイトを下回っています。

```
$ dsadm start /local/ds
Server started: pid=8046
$ dsconf create-suffix dc=example,dc=com
Certificate "CN=hostname, CN=1636, CN=Directory Server,
 O=Sun Microsystems" presented by the server is not trusted.
Type "Y" to accept, "y" to accept just once, "n" to refuse, "d" for more
 details: Y
$ du -hs /local/ds
53M   /local/ds
```

この例では、特に明記しないかぎり、Directory Server のデフォルト設定に対する追加変更は行いません。

サフィックスに 10,000 件のサンプルディレクトリエントリを設定する

Directory Server Resource Kit の一部として提供されている `makeldif` コマンドとサンプルファイルを使用すれば、1K バイトから 1M バイトまでのサイズのサンプル LDIF ファイルを作成できます。`makeldif` コマンドの使用方法を示す例については、「To Install Directory Server Enterprise Edition 6.3 From Zip Distribution」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide 』を参照してください。

次の各ファイル内のエントリは、実際の配備時に使用するエントリ数よりは少なめかもしれません。

```
$ du -h /var/tmp/*
 57M   /var/tmp/100k.ldif
 5.7M   /var/tmp/10k.ldif
 573M   /var/tmp/1M.ldif
```

これらのファイル内のエントリの例を、次の LDIF に示します。

```
dn: uid=Aartjan.Aalders,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
givenName: Aartjan
```

```
sn: Aalders
cn: Aartjan Aalders
initials: AA
uid: Aartjan.Aalders
mail: Aartjan.Aalders@example.com
userPassword: trj49xeq
telephoneNumber: 935-748-6699
homePhone: 347-586-0252
pager: 906-399-8417
mobile: 452-898-9034
employeeNumber: 1000004
street: 64197 Broadway Street
l: Lawton
st: IN
postalCode: 57924
postalAddress: Aartjan Aalders$64197 Broadway Street$Lawton, IN  57924
description: This is the description for Aartjan Aalders.
```

まず、ディスク上で 5.7M バイトを占有する `10k.ldif` の内容をインポートすることで、サイジングを始めます。

```
$ dsadm stop /local/ds
Server stopped
$ dsadm import -i /local/ds /var/tmp/10k.ldif dc=example,dc=com
...
```

デフォルトのインデックス作成では、`10k.ldif` により、インスタンスファイルのサイズが 72M バイト - 53M バイト、つまり 19M バイトだけ増加します。

```
$ du -hs /local/ds
 72M   /local/ds
```

さらに別の 5 つの属性にもインデックスを付けると、サイズが約 7M バイト増加します。

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsconf reindex dc=example,dc=com
...
## example: Finished indexing.

Task completed (slapd exit code: 0).
$ du -hs /local/ds
 79M   /local/ds
```

デフォルトのキャッシュ設定でメモリーサイズを監視すると、サフィックスからエントリキャッシュへまだ何も読み込まれていなければ、サーバープロセスは約 170M バイトのメモリーを占有し、そのヒープサイズは約 56M バイトになります。

```
$ dsadm start /local/ds
Server started: pid=8482
$ pmap -x 8482
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
0000000000437000      61348      55632      55380          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     178444     172604      76532          -
```

次に、キャッシュに情報を格納してから pmap コマンドの出力を再度確認すると、ヒープが約 10M バイトだけ増加し、プロセスの合計サイズもそれと同じだけ増加しています。

```
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8482
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      70564      65268      65024          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     187692     182272      86224          -
```

この数値を、デフォルトインデックス作成の場合と比較してみましょう。

```
$ dsconf delete-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsconf reindex dc=example,dc=com
...
## example: Finished indexing.

Task completed (slapd exit code: 0).
$ dsadm stop /local/ds
 Server stopped
$ dsadm start /local/ds
 Server started: pid=8541
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8541
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      70564      65248      65004          - rw---    [ heap ]
...
```

```
---------------- ---------- ---------- ---------- ----------
        total Kb     187680     182240      86192          -
```

エントリ数がわずか 10,000 件であれば、デフォルトのキャッシュサイズを変更しないでください。

```
$ dsconf get-server-prop | grep cache
db-cache-size                     :  32M
import-cache-size                 :  64M
$ dsconf get-suffix-prop dc=example,dc=com | grep entry-cache-size
entry-cache-size                  :  10M
```

デフォルトのエントリキャッシュはサイズが小さいため、エントリ数がわずか 10,000 件であっても、情報が格納されると間違いなく、キャッシュが完全にいっぱいになります。エントリが完全に収まるキャッシュのサイズを確認するには、エントリキャッシュサイズをある大きな値に設定し、データをインポートし直してから、キャッシュに情報を格納します。

```
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:2G
$ dsadm stop /local/ds
Server stopped
$ dsadm import -i /local/ds /var/tmp/10k.ldif dc=example,dc=com
...
$ dsadm start /local/ds
Server started: pid=8806
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8806
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     116644     109996     109780          - rw---    [ heap ]
```

ここでは、10,000 件のエントリが、約 55M バイト (110 - 55) のエントリキャッシュメモリーを占有しています。

サフィックスに 100,000 件のサンプルディレクトリエントリを設定する

100,000 件のエントリを追加する場合は、データベースやエントリキャッシュ内に収めるべきディレクトリデータの量が増えます。まず、100,000 件のエントリをインポートし、この分量のディレクトリデータで必要となるディスクのサイズを確認します。

```
$ dsadm import -i /local/ds /var/tmp/100k.ldif dc=example,dc=com
...
```

```
$ du -hs /local/ds
 196M   /local/ds
```

データベース内に格納されている、サフィックス dc=example,dc=com のディレクトリデータは、この時点で約 142M バイトを占有しています。

```
$ du -hs /local/ds/db/example/
 142M   /local/ds/db/example
```

データベースキャッシュのサイズを増やせば、この内容がキャッシュ内に収まるようにすることができます。ディレクトリデータの分量が時間の経過とともに増加することが予期される場合には、データベースキャッシュを現在必要とされるよりも大きな値に設定できます。エントリキャッシュサイズも、必要とされるより大きな値に設定できます。エントリキャッシュは、起動時に割り当てられるデータベースキャッシュとは異なり、サーバーがクライアントの要求に応答するたびに増加します。

```
$ dsconf set-server-prop db-cache-size:200M
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:2G

$ dsadm stop /local/ds
 Server stopped
$ dsadm start /local/ds
 Server started: pid=8640
$ pmap -x 8640
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      61348      55404      55148          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     491984     485736     174620          -
```

これから、起動時のサーバーインスタンスのヒープは比較的小さいが、データベースキャッシュのメモリーは割り当て済みであることがわかります。プロセスのサイズは、1G バイトの約半分になっています。

```
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8640
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     610212     604064     603840          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1040880    1034428     723360          -
```

ヒープサイズはこの時点で、情報が格納されたエントリキャッシュを反映したものとなっています。これは、100,000 件の小規模ディレクトリエントリの場合に約 550M バイト増えています。その LDIF はディスク上で 57M バイトを占有していました。

インデックスを 5 つ追加しても、プロセスのサイズはほぼ同じです。データベースキャッシュのサイズに変わりはありません。

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsadm stop /local/ds
 Server stopped
$ dsadm import -i /local/ds /var/tmp/100k.ldif dc=example,dc=com
...
$ dsadm start /local/ds
 Server started: pid=8762
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8762
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     610212     603832     603612          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1040876    1034192     723128          -
```

ただし、データベースのサイズは若干大きくなっています。インデックスを追加したことで、データベースのサイズが 142M バイトから 163M バイトへと増加しました。

```
$ du -hs /local/ds/db/example/
 163M   /local/ds/db/example
```

サフィックスに 1,000,000 件のサンプルディレクトリエントリを設定する

100,000 件のエントリから 1,000,000 件のエントリに移行すると、4G バイトの物理メモリーを備えたシステム上ではもはや十分な容量を確保できず、すべてのエントリをエントリキャッシュ内に収めることができなくなります。まず、データをインポートし、そのディスク上での占有サイズを確認します。

```
$ dsadm import -i /local/ds /var/tmp/1M.ldif dc=example,dc=com
...
$ du -hs /local/ds/db/example/
 1.3G   /local/ds/db/example
```

このインスタンスの存続期間中にディレクトリデータのサイズが約 25 % 増加することが予期されると仮定して、データベースキャッシュのサイズを 1700M バイトに設定します。

```
$ dsadm start /local/ds
Server started: pid=9060
$ dsconf set-server-prop db-cache-size:1700M
$ dsadm stop /local/ds
Server stopped
$ dsadm start /local/ds
Server started: pid=9118
$ pmap -x 9118
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      65508      55700      55452          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1882448    1034180      76616          -
```

データベースキャッシュがこの大きさで、かつ物理メモリーが 4G バイトしかなければ、このサフィックスのエントリキャッシュに格納できるのは、ほんの一部のエントリだけになります。ここでは、エントリキャッシュサイズを 1G バイトに設定してからキャッシュに情報を格納し、プロセスのヒープサイズがどのように変化するかを確認します。

```
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:1G
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 9118
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000    1016868    1009852    1009612          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    2883268    2477064    1080076          -
```

プロセスの合計サイズは 2.8G バイトを超えています。

```
$ prstat -p 9118
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  9118 myuser   2816M 2374M sleep   59    0   0:03:26 0.5% ns-slapd/42
```

これまでのエントリキャッシュサイズに基づいて推定すると、十分な物理メモリーが存在していれば、エントリキャッシュだけで 5.5G バイトまたは 6G バイトが使用されることが予期できます。

インデックスを 5 つ追加してディレクトリデータベースのサイズを確認すると、そのインデックス追加によって、データベースのサイズが約 200M バイト増加したことがわかります。

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsadm stop /local/ds
Server stopped
$ dsadm import -i /local/ds /var/tmp/1M.ldif dc=example,dc=com
...
$ du -hs /local/ds/db/example
 1.5G   /local/ds/db/example
```

測定結果

表 6–3 は、この例での測定内容を記録したものです。ここには、サーバープロセスサイズ、デフォルトデータベースキャッシュファイルサイズのいずれも含まれていません。

注 – 実際の配備環境で同様の実験を行なった場合の結果はおそらく、ここで示したものと大幅に異なります。

表 6–3　サイジングの測定結果

エントリ数	LDIF ファイルのサイズ	デフォルトのインデックスを含むディスク	5 つのインデックスが追加されたディスク	データベースキャッシュ	エントリキャッシュ
0[1]	測定値なし	測定値なし	測定値なし	測定値なし	測定値なし
10,000	5.7M バイト	19M バイト	26M バイト	32M バイト	55M バイト
100,000	57M バイト	142M バイト	163M バイト	200M バイト	550M バイト
1,000,000	573M バイト	1300M バイト	1500M バイト	1700M バイト (デフォルトのインデックス作成)	測定値なし

[1] サフィックスは作成済みですが空の状態です。

実際の配備時には、エントリ数やインデックスを作成する属性の数が大幅に増える可能性があります。ハードウェアを注文する前に各自で実験的なテストを行い、調整を施すようにしてください。

Directory Server 向けのオペレーティングシステムチューニング

デフォルトのシステム設定でディレクトリサービスのパフォーマンスが最高になるとはかぎりません。この節では、最高のパフォーマンスを実現するにはオペレーティングシステムをどのようにチューニングすればよいかについて説明します。

オペレーティングシステムのバージョンとパッチのサポート

サポートされるオペレーティングシステムの一覧の最新情報は、『Sun Java System Directory Server Enterprise Edition 6.3 Release Notes』を参照してください。

システムの全体的なセキュリティーを維持する必要があります。また、Directory Server の正常な動作を保証する必要もあります。したがって、推奨される最新のシステムパッチ、サービスパック、またはバグフィックスをインストールします。使用するプラットフォームで適用すべき最新パッチの一覧の最新情報については、『Sun Java System Directory Server Enterprise Edition 6.3 Release Notes』を参照してください。

基本的なセキュリティーチェック

この節の推奨事項を実施しても、すべてのリスクを回避できるわけではありません。これらの推奨事項の目的は、典型的なセキュリティー上の危険に歯止めをかけるための簡易チェックリストを提供することです。

- ファイアウォールを使ってシステムを隔離する:可能であれば、Directory Server が稼働するシステムを、ネットワークファイアウォールを使って公的なインターネットから隔離してください。
- デュアルブートを許可しない:本番用の Directory Server が稼働するシステム上でほかのオペレーティングシステムを実行しないでください。アクセスを許可すべきでないファイルへのアクセスを、ほかのシステムが許可する可能性があります。
- 強いパスワードを使用する:少なくとも 8 文字の長さの root パスワードを使用します。パスワードには、コンマ、ピリオドなど、アルファベット以外の文字も含めるようにしてください。

「パスワード強度チェック」サーバープラグインを使えば、弱いパスワードを拒否できます。dsconf でサーバープロパティー pwd-strong-check-enabled を使えば、このプラグインを有効にできます。

より長いオペレーティングシステムパスワードを使用することにした場合、システムによるパスワードの処理方法を設定しなければいけない可能性があります。その手順については、オペレーティングシステムのマニュアルを参照してください。

- 安全なユーザーおよびグループ ID をサーバーに使用する: セキュリティー上の理由により、スーパーユーザー特権を使って Directory Server を実行しないでください。

  たとえば、UNIX コマンド groupadd と useradd を使えば、ログイン特権を持たないユーザーとグループを作成できます。次に、このユーザーとグループとしてサーバーを実行できます。

  たとえば、servers という名前のグループを追加するには、次のようにします。

  ```
  # groupadd servers
  ```

  server1 という名前のユーザーをグループ servers のメンバーとして追加するには、次のコマンドを使用します。

  ```
  # useradd -g servers -s /bin/false -c "server1"
  ```

  ある特定の配備で、メッセージングサーバーなどのほかのサーバーと Directory Server ファイルを共有する必要が生じる可能性があります。そのような配備では、同じユーザーおよびグループ ID を使ってそれらのサーバーを実行してください。
- コア機能を使用する:デバッグを支援するために、このユーザーおよびグループ ID で実行されているプロセスにコアダンプを許可できます。Solaris コマンド coreadm などのユーティリティーを使用します。たとえば、Directory Server がコアファイルを生成できるようにするには、setuid プロセスにその処理を許可したあと、coreadm の設定を更新します。

  ```
  # coreadm -e proc-setid
  # coreadm -u
  ```

  サーバーを起動するスクリプトを作成するとき、その起動スクリプトに次の行を追加できます。この行を追加すると、core.ns-slapd.*pid* という形式のコアファイルを Directory Server が生成できるようになります。ここで、*pid* はプロセス ID です。

  ```
  coreadm -p core.%f.%p $$
  ```
- 不要なサービスを無効にする:より少ないリスクで最高のパフォーマンスを実現するには、システムを Directory Server 専用にします。このガイドの別のところで説明したように、同じマシン上で Directory Service Control Center を実行しないでく

ださい。別のサービス、特にネットワークサービスを実行すると、サーバーのパフォーマンスとスケーラビリティーに悪影響が及びます。また、それにより、セキュリティー上の危険も増大します。

できるだけ多くのネットワークサービスを無効にしてください。Directory Server はファイル共有やその他のサービスを必要としません。IP Routing、Mail、NetBIOS、NFS、RAS、Web Publishing、Windows Network Client などのサービスを無効にしてください。`telnet` と `ftp`を無効にすることを検討してください。

`telnet` と `ftp` はほかの多くのネットワークサービスと同じく、セキュリティー上の危険を増大させます。これら 2 つのサービスは特に危険です。なぜなら、これらのコマンドはネットワーク経由でユーザーのパスワードを平文として送信するからです。`telnet` や `ftp` を使用する必要に迫られた場合には、セキュリティー保護されたシェルである `ssh` やセキュリティー保護された FTP である `sftp` などのクライアントを代わりに使用してください。ネットワークサービスを無効にする方法の詳細については、オペレーティングシステムのマニュアルを参照してください。

Directory Server インスタンスがネットワークに対してネームサービスを提供しない場合、システムのネームサービスを有効にすることを検討してください。その場合、Directory Server は、ACI を解決する場合などにそのネームサービスを使用します。

正確なシステムクロック

システムクロックがほかのシステムとほぼ同期が取れていることを確認してください。クロックの同期がとれていれば、レプリケーションも正常に行えます。また、ログファイル内の日付やタイムスタンプのシステム間での対応関係も把握しやすくなります。時間情報プロトコル (NTP、Network Time Protocol) クライアントを使って正しいシステム時刻を設定することを検討してください。

システムリブート時の再起動

`dsadm` コマンドを使えば、システムブート時にサーバーインスタンスを再起動させることができます。たとえば、Solaris 10 および Windows システムでは `dsadm enable-service` コマンドを使用します。ほかのシステムでは `dsadm autostart` コマンドを使用します。ネイティブパッケージからインストールしなかった場合には、オペレーティングシステムのマニュアルを参照して、システムブート時にサーバーを起動させる方法を確認してください。

可能であれば、`dsadm` コマンド、DSCC のいずれかを使って Directory Server を停止します。システムの停止中に Directory Server が突然停止されると、すべてのデータがディスクに正しく書き込まれたことが保証されなくなります。したがって、Directory Server は再起動時に、データベースの完全性を確認する必要があります。このプロセスにはある程度の時間がかかる可能性があります。

さらに、ファイルシステムのロギングオプションの使用を検討してください。ファイルシステムのロギングは一般に、書き込み時のパフォーマンスを改善すると同時に、ファイルシステムチェックの実行に必要な時間を短縮します。クラッシュ時にファイルシステムが正常にマウント解除されなかった場合には、システムはファイルシステムをチェックする必要があります。また、RAID をストレージとして使用することも検討してください。

idsktune コマンドによるシステム固有のチューニング

製品に付属する idsktune(1M) ユーティリティーを使えば、基本的なシステム設定の問題を診断するのに役立つかもしれません。このユーティリティーは、高いパフォーマンスを示すディレクトリサービスをサポートするようにシステムをチューニングするための推奨事項を提示します。このユーティリティーは、推奨事項を示すだけで、実際には何の操作も行いません。適切な権限を持ったシステム管理者が提示された推奨事項に基づいて必要な操作を行なってください。

ユーティリティーを root として実行すると、idsktune はシステムに関する情報を収集します。ユーティリティーは、注意、警告、およびエラーを推奨の対処法とともに表示します。idsktune コマンドがチェックする内容は、次のとおりです。

- オペレーティングシステムとカーネルのバージョンが、このリリースでサポートされている。
- 利用可能なメモリーと利用可能なディスク容量が、通常用途の最小要件を満たしている。
- システムリソースの限度が通常用途の最小要件を満たしている。
- 必須パッチがインストールされている。

ヒント – Directory Server ソフトウェアを本番用のシステムにインストールする前に、少なくともすべての ERROR 状態を解決してください。

個々の配備要件が最小要件を上回る可能性があります。idsktune ユーティリティーによって最小システム要件として指摘されたリソースよりも多くのリソースを提供することができます。

特定の推奨事項を実施する前に、ローカルネットワークの状態やその他のアプリケーションを考慮してください。ネットワーク設定のチューニングに関する追加のヒントについては、オペレーティングシステムのマニュアルを参照してください。

ファイル記述子の設定

Directory Server は、同時クライアント接続を処理する際にファイル記述子を使用します。したがって、サーバープロセスから利用可能なファイル記述子の最大数が小さければ、同時接続数が制限される可能性があります。したがって、ファイル記述子の数に関する推奨事項は、Directory Server が処理可能な同時接続数に関係します。

Solaris システムの場合、利用可能なファイル記述子の数を設定するには `rlim_fd_max` パラメータを使用します。利用可能なファイル記述子の数を変更するための詳細な手順については、オペレーティングシステムのマニュアルを参照してください。

伝送制御プロトコル (TCP) の設定

具体的なネットワーク設定はプラットフォームに依存します。一部のシステムでは、TCP 設定を変更することで Directory Server のパフォーマンスを向上させることができます。

注 – まずディレクトリサービスを配備し、次にそれらのパラメータのチューニングを必要に応じて検討します。

この節では、`idsktune` による TCP 設定関連の推奨事項の背景にある根拠について説明したあと、Solaris 10 システム上でそれらの設定をチューニングする方法を示します。

アクティブでない接続

一部のシステムでは、`keepalive` パケットの送信間隔を設定できます。この設定によって、アクティブでない、場合によっては切断されている TCP 接続を、どのくらいの期間維持するかを決めることができます。`keepalive` 間隔の設定値が大きすぎると、すでに切断されたクライアント接続を維持するためにシステムが不要なリソースを使用してしまう可能性があります。大部分の配備では、このパラメータの値を 600 秒に設定します。この値を 600,000 ミリ秒すなわち 10 分に設定することで、Directory Server への同時接続数を増やすことができます。

一方、`keepalive` 間隔の設定値が小さすぎると、ネットワークの一時的な停止時にもシステムが接続を切断してしまう可能性があります。

Solaris システムでこの時間間隔を設定するには、`tcp_keepalive_interval` パラメータを使用します。

送信接続

一部のシステムでは、送信接続が確立されるのをシステムがどのくらいの期間待つかを設定できます。設定値が大きすぎると、すぐに応答しないレプリカなどの送信先サーバーとの送信接続を確立する際に、大きな遅延が生じる可能性があります。

高速で信頼性の高いネットワーク上のイントラネット配備の場合、このパラメータの値を 10 秒に設定すれば、パフォーマンスの改善が期待できます。一方、低速、低信頼、または WAN 型の接続を含むネットワーク上では、そうした小さな値を使用しないでください。

Solaris システムでこの時間間隔を設定するには、`tcp_ip_abort_cinterval` パラメータを使用します。

再送信タイムアウト

一部のシステムでは、パケットの再送信間の初期時間間隔を設定できます。この設定は、確認応答のなかったパケットが再送信されるまでの待機時間に影響を与えます。設定値が大きすぎると、失われたパケットのためにクライアントが待たされる可能性があります。高速で信頼性の高いネットワーク上のイントラネット配備の場合、このパラメータの値を 500 ミリ秒に設定すれば、パフォーマンスの改善が期待できます。一方、往復時間が 250 ミリ秒を超えるようなネットワークでは、そうした小さな値を使用しないでください。

Solaris システムでこの時間間隔を設定するには、`tcp_rexmit_interval_initial` パラメータを使用します。

シーケンス番号

一部のシステムでは、システムが初期シーケンス番号の生成を処理する方法を設定できます。エクストラネットおよびインターネット配備では、シーケンス番号攻撃を避けるため、初期シーケンス番号の生成が RFC 1948 に基づいて行われるようにこのパラメータを設定してください。そうした環境では、ここで説明したほかの TCP チューニング設定は役に立ちません。

Solaris システムでこの動作を設定するには、`tcp_strong_iss` パラメータを使用します。

Solaris 10 システムでの TCP 設定のチューニング

Solaris 10 システム上で TCP 設定をチューニングするもっとも単純な方法は、単純な SMF サービスを次のようにして作成することです。

- Directory Server チューニング用の SMF プロファイルを作成します。
- 次の xml ファイルを環境に従って編集し、そのファイルを `/var/svc/manifest/site/ndd-nettune.xml` として保存します。

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/ service_bundle.dtd.1">
<!--
    ident    "@(#)ndd-nettune.xml    1.0    04/09/21 SMI"
-->

<service_bundle type='manifest' name='SUNWcsr:ndd'>

<service
    name='network/ndd-nettune'
    type='service'
    version='1'>

            <create_default_instance enabled='true' />

            <single_instance />

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

        <dependency
        name='loopback-network'
        grouping='require_any'
        restart_on='none'
        type='service'>
            <service_fmri value='svc:/network/loopback' />
        </dependency>

        <dependency
        name='physical-network'
        grouping='optional_all'
        restart_on='none'
        type='service'>
```

```
                <service_fmri value='svc:/network/physical' />
            </dependency>

            <exec_method
            type='method'
            name='start'
            exec='/lib/svc/method/ndd-nettune'
            timeout_seconds='3' />
            </exec_method>

            <exec_method
                   type='method'
                   name='stop'
                   exec=':true'
                   timeout_seconds='3' >
            </exec_method>

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

            <stability value='Unstable' />

            <template>
                <common_name>
                    <loctext xml:lang='C'>
                        ndd network tuning
                    </loctext>
                </common_name>
                <documentation>
                    <manpage title='ndd' section='1M'
                    manpath='/usr/share/man' />
                </documentation>
            </template>

    </service>

    </service_bundle>
```

- ndd-nettune.xml の設定内容をインポートする前に、その構文が正しいことを確認します。それには次のコマンドを実行します。

```
$ svccfg validate /var/svc/manifest/site/ndd-nettune.xml
```

- 次のコマンドを実行して設定をインポートします。

```
$ svccfg import /var/svc/manifest/site/ndd-nettune.xml
```

詳細については、svccfg(1M) のマニュアルページを参照してください。

- 次のシェルスクリプトを /lib/svc/method/ndd-nettune 内にコピーします。

```
#!/sbin/sh
#
# ident   "@(#)ndd-nettune.xml    1.0     01/08/06 SMI"

. /lib/svc/share/smf_include.sh
. /lib/svc/share/net_include.sh

# Make sure that the libraries essential to this stage of booting  can be found.
LD_LIBRARY_PATH=/lib; export LD_LIBRARY_PATH
echo "Performing Directory Server Tuning..." >> /tmp/smf.out
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 1024
/usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval 600000
/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_cinterval 10000
/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_interval 60000


# Reset the library path now that we are past the critical stage
unset LD_LIBRARY_PATH
```

- svcadm を実行して nettune (詳細は svcadm(1M) のマニュアルページを参照) を有効にします。
- svcs -x (詳細は svcs(1) のマニュアルページを参照) を実行します。

Directory Server の物理的な機能

次に、Directory Server のスケーラビリティーの決定要因となる物理的な機能を示します。

- プロセスサイズ。オペレーティングシステムによって異なりますが、32 ビット Directory Server は 2G バイト～ 4G バイトのプロセスサイズをサポートします。64 ビット Directory Server のプロセスサイズは、マシン上で使用可能な物理メモリーの容量によって決まります。128G バイトのプロセスサイズによるテストが完了しています。
- LDAP エントリの数。1 つのサーバーインスタンス上で作成できる LDAP エントリの総数は 2^32 -1 で、4G のエントリということになります。
- 各エントリのサイズ。LDAP サーバー内の 1 つのレコードのサイズは 4G バイトです。これはデータベースそのものによって決まります。エントリのサイズも、LDAP 要求の最大サイズ (maxbersize) によって決まります。この最大サイズは 2G バイトです。
- LDAP 接続の数。LDAP 接続の数は、プロセスが開くことのできるファイル記述子の数によって決まります。開いた接続の数が多すぎるとパフォーマンスが低下するので注意が必要です。

- LDAP サーバー (Berkery DB) のサイズ。LDAP サーバーのサイズは、使用するファイルシステムのサイズによって決まります。

◆ ◆ ◆ 第 7 章

7 セキュリティー要件の特定

Directory Server Enterprise Edition のデータを保護する方法は、ほかのすべての設計領域に影響します。この章では、セキュリティー要件の分析方法と、その要件を満たすディレクトリサービスの設計方法について説明します。

この章の内容は次のとおりです。

セキュリティーに対する脅威

ディレクトリのセキュリティーを脅かすもっとも典型的な脅威は、次のとおりです。

- 盗聴:情報は元の状態のままですが、その機密性が損なわれます。たとえば、だれかが、ユーザーのクレジットカード番号を知ったり、機密性の高い会話を記録したり、極秘情報を傍受したりする可能性があります。
- 不正なアクセス:この脅威には、データ取得操作による、データへの不正なアクセスが含まれます。不正なユーザーが、他のユーザーのアクセスを監視することにより、再利用可能なクライアント認証情報にアクセスする可能性があります。Directory Server Enterprise Edition の認証方法、パスワードポリシー、およびアクセス制御のメカニズムは、不正アクセスの防止に効果があります。
- 改ざん:転送中の情報が、変更または置換されてから宛先に送信されます。たとえば、だれかが商品の注文を変更したり、他人の履歴書を変更したりできます。

  この脅威には、データまたは設定情報の不正な変更が含まれます。ディレクトリが改ざんを検出できない場合、攻撃者がサーバーへのクライアントの要求を変更する可能性があります。また、攻撃者が要求を取り消したり、クライアントへのサーバーの応答を変更したりする可能性もあります。SSL (Secure Socket Layer) プロトコルや、それと同様の技術を利用して、接続の両端で情報に署名することで、この問題は解決できます。
- 偽装:情報が、意図する受信者のふりをした人に渡されます。

  偽装には、なりすましと不当表示の 2 つの形態があります。
  - なりすまし:人またはコンピュータが、ほかのだれかのふりをします。たとえば、ある人がメールアドレス `jdoe@example.com` を持っているふりをしたり、あるコンピュータが自身を偽って `www.example.com` という名前のサイトのふりをしたりする可能性があります。
  - 不当表示:人または組織が自身を不当表示します。たとえば、サイト `www.example.com` が、あたかも家具販売店であるふりをしているが、実際にはクレジットカードによる支払いを受けても決して商品を送らないサイトだったりします。
- サービス拒否:攻撃者が、システムリソースを使用することで、それらのリソースが正規ユーザーによって使用されるのを妨害します。

  サービス拒否攻撃とは、侵入者がディレクトリによるクライアントへのサービスの提供を妨害することです。Directory Server Enterprise Edition では、特定のバインド DN に割り当てるリソースに制限を設定することで、サービスの拒否攻撃を防ぎます。

セキュリティー手法の概要

セキュリティーポリシーは、不正なユーザーによって機密情報が変更または取得されるのを防げる必要がありますが、同時に十分に管理しやすい必要もあります。

Directory Server Enterprise Edition が提供するセキュリティー手法は、次のとおりです。

- 認証:一方が他方の識別情報を検証する方法を提供します。たとえば、LDAP のバインド操作時に、クライアントは Directory Server にパスワードを提示します。「パスワードポリシー」は認証プロセスの一部として、有効期限、長さ、構文など、パスワードが有効とみなされるために満たす必要のある条件を定義します。「アカウントの無効化」は、ユーザーアカウント、アカウントのグループ、またはドメイン全体を無効にして、すべての認証の試行に対して、自動的に拒否するようにします。
- 暗号化:情報の機密性を保護します。データを暗号化すると、データは受信者だけが復号化できるような方法で符号化されます。「Secure Sockets Layer」(SSL) は、転送中の情報を暗号化することでデータの完全性を維持します。送信する情報に暗号化とメッセージダイジェストを適用した場合、受信者は、その情報が転送中に改ざんされていないことを確認できます。「属性暗号化」は、格納された情報を暗号化することでデータの完全性を維持します。
- アクセス制御:さまざまなディレクトリユーザーに与えるアクセス権限を調整し、必要な証明情報またはバインド属性を指定する方法を提供します。
- 監査:ディレクトリのセキュリティーが危険にさらされていないかを確認できます。たとえば、ディレクトリで保持されるログファイルを監査できます。

これらのセキュリティーツールは、ユーザーのセキュリティー設計で組み合わせて使用できます。セキュリティーの設計をサポートするために、レプリケーションやデータの分散など、ほかのディレクトリの機能を使用することもできます。

認証方法の決定

Directory Server Enterprise Edition は、次の認証方法をサポートしています。

- 128 ページの「匿名アクセス」
- 128 ページの「単純パスワード認証」
- 129 ページの「セキュリティー保護された接続での単純パスワード認証」
- 129 ページの「証明書に基づくクライアント認証」
- 130 ページの「SASL ベースのクライアント認証」

ユーザーが人間か LDAP 対応アプリケーションかにかかわらず、すべてのユーザーに対して同じ認証メカニズムが適用されます。

この節には、上記の認証メカニズムのほかに、認証に関する次の情報も含まれています。

- 130 ページの「アカウントの無効化による認証の防止」
- 131 ページの「グローバルアカウントロックアウトを使用した認証の防止」
- 132 ページの「外部認証のマッピングおよびサービス」

匿名アクセス

匿名アクセスは、ディレクトリアクセスのもっとも単純な形態です。匿名アクセスは、ユーザーが認証済みかどうかにかかわらず、すべてのディレクトリユーザーに対してデータを利用可能にします。

匿名アクセスでは、だれが検索を実行しているかや、どのような種類の検索が実行されているかを追跡することができません。追跡できるのは、だれかが検索を実行している、ということだけです。匿名アクセスを許可すると、ディレクトリに接続するユーザーはだれでもデータにアクセスできます。データへの匿名アクセスを許可したまま、あるユーザーやグループをそのデータからブロックしようとしても、そのユーザーは、ディレクトリに匿名でバインドすることでそのデータにアクセスできます。

匿名アクセスの特権は制限できます。通常、ディレクトリ管理者は、匿名アクセスに対して読み取り、検索、および比較の特権だけを許可します。また、ユーザー名、電話番号、電子メールアドレスなど、一般的な情報を含む属性のサブセットにアクセスを限定することもできます。政府識別番号、自宅の電話番号や住所、給与情報など、機密データへの匿名アクセスを許可しないでください。

ルート DSE (ベース DN "") への匿名アクセスは必要です。アプリケーションはルート DSE への匿名アクセスを使用することで、サーバーの機能、サポートされているセキュリティーメカニズム、およびサポートされているサフィックスを確認できます。

単純パスワード認証

匿名アクセスが設定されていない場合、クライアントは、Directory Server への認証を行わないとディレクトリのコンテンツにアクセスできません。単純パスワード認証では、クライアントは再使用可能な簡単なパスワードを提供して、サーバーへの認証を行います。

クライアントはバインド操作を使って Directory Server への認証を行いますが、そのバインド操作でクライアントは識別名と資格情報を提供します。サーバーは、そのクライアント DN に対応するエントリを特定したあと、そのクライアントのパスワードがエントリに格納された値に一致するかどうかをチェックします。パスワードが一致すれば、サーバーはそのクライアントを認証します。パスワードが一致しない場合、認証操作は失敗し、クライアントにエラーメッセージが返されます。

注 – 単純パスワード認証の欠点は、パスワードが平文として送信されることです。その場合、セキュリティーが損なわれる可能性があります。悪意のあるユーザーが盗聴している場合、そのユーザーは承認されたユーザーを偽装できます。

単純パスワード認証を使えば、ユーザーの認証を容易に行えます。ただし、単純パスワード認証を使用するのは、組織のイントラネットに限定する必要があります。この種類の認証では、エクストラネットを介した取引先との転送やインターネット上での顧客との転送に求められるレベルのセキュリティーは提供されません。

セキュリティー保護された接続での単純パスワード認証

セキュリティー保護された接続では、暗号化によって第三者がデータを読めないようにした上で、Directory Server とクライアントの間でデータを送信します。クライアントは、次のいずれかの方法でセキュリティー保護された接続を確立できます。

- SSL (Secure Socket Layer) を使用してセキュリティー保護されたポートにバインドする
- 匿名アクセスでセキュリティー保護されていないポートにバインドし、Start TLS 制御を送信して TLS (Transport Layer Security) を使用する

どちらの場合も、サーバーにはセキュリティー証明書が必要で、この証明書を信頼するようにクライアントを設定する必要があります。サーバーは、証明書をクライアントに送信することで、公開鍵暗号化方式を使用して「サーバー認証」を行います。その結果、クライアントは目的のサーバーに接続していること、およびサーバーが改ざんされていないことを認識します。

次に、機密保護のため、クライアントとサーバーは、接続を介して送信されるすべてのデータを暗号化します。クライアントは、暗号化された接続でバインド DN とパスワードを送信してユーザー認証を受けます。それ以降の操作はすべて、そのユーザーの ID を使って実行されます。バインド DN がほかのユーザー ID のプロキシ権限を持っている場合には、プロキシ ID を使って操作が実行される可能性もあります。操作の結果がクライアントに返されるときは、すべてが暗号化されます。

証明書に基づくクライアント認証

SSL または TLS 上で暗号化接続を確立する場合、「クライアント認証」を要求するようにサーバーを設定することもできます。クライアントは、ユーザー ID の確認のためにサーバーに証明書を送信する必要があります。バインド DN の決定には、ユーザーの DN ではなく、証明書が使用されます。クライアント認証は、ユーザーの偽装を防ぐ保護で、もっとも安全な接続です。

証明書に基づくクライアント認証は、SSL 層と TLS 層のみで動作します。証明書に基づく認証 ID を LDAP で使用するには、SSL 接続の確立後に SASL EXTERNAL 認証を使用する必要があります。

証明書に基づくクライアント認証を設定するには、`dsconf get-server-prop` コマンドを使用します。詳細については、`dsconf`(1M) のマニュアルページを参照してください。

SASL ベースのクライアント認証

SSL または TLS 接続時のクライアント認証では、汎用セキュリティーインタフェースの Simple Authentication and Security Layer (SASL) を使ってクライアントの ID を確立することもできます。Directory Server Enterprise Edition が SASL を通じてサポートするメカニズムは、次のとおりです。

- **DIGEST-MD5:** このメカニズムは、クライアントから送信されたハッシュ値とユーザーパスワードのハッシュを比較することでクライアントを認証します。ただし、このメカニズムはユーザーパスワードを読み取る必要があり、DIGEST-MD5 による認証を希望するすべてのユーザーは、ディレクトリ内に `{CLEAR}` パスワード (平文形式のパスワード) を持つ必要があります。
- **GSSAPI:** General Security Services API (GSSAPI) は、Solaris オペレーティングシステム上でしか利用できません。これを使えば、Directory Server は Kerberos V5 セキュリティーシステムと対話してユーザーを特定できます。クライアントアプリケーションは Kerberos システムに証明情報を提示し、このシステムがユーザーの ID を Directory Server に返します。
- **EXTERNAL:** このメカニズムは、SSL や TLS などの外部セキュリティー層によって指定された資格情報に基づいて、LDAP のユーザーを認証します。

詳細については、「Using SASL DIGEST-MD5 in Clients」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』および「Using Kerberos SASL GSSAPI in Clients」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

アカウントの無効化による認証の防止

あるユーザーアカウントまたは一連のアカウントを無効にすることで、認証を一時的に防止することができます。アカウントを無効にすると、ユーザーは Directory Server にバインドできず、認証処理が失敗します。詳細については、「Manually Locking Accounts」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

グローバルアカウントロックアウトを使用した認証の防止

このバージョンの Directory Server では、パスワードによる認証失敗が監視およびレプリケートされます。これにより、無効なパスワードによる認証の試みがある指定された回数だけ行われたときに、グローバルアカウントロックアウトを速やかに行えます。グローバルアカウントロックアウトは、次のどの場合でもサポートされています。

- クライアントアプリケーションがトポロジ内の単一のサーバーだけにバインドする
- 読み取り専用のコンシューマがトポロジ内に 1 つも含まれていない
- Directory Proxy Server を使ってすべてのバインドトラフィックが制御される

すべてのバインド試行が同一のサーバーに向けられず、ロックアウトデータがレプリケートされるよりも早く、クライアントアプリケーションが複数のサーバー上でバインド試行を実行するような状況を想像してください。最悪の場合、クライアントがバインドを試みたサーバーごとに、指定された回数の試みをクライアントに許可することになります。クライアントアプリケーションを駆動する入力を与えるユーザーが人間である場合には、こうした状況はあまり起こりそうにありません。しかし、トポロジを攻撃するために構築された自動化クライアントであれば、この配備選択を悪用することができます。

優先順位付きレプリケーションを使えば、侵入者検出の際に、非同期レプリケーションの遅れによる影響を最小限に抑えることができます。ただし、バインド試行が指定された回数だけ失敗した直後にアカウントロックアウトを行いたい場合もあります。その状況では、ある特定のエントリ上でのすべてのバインドの試みを、Directory Proxy Server を使って同一のマスターレプリカに配信する必要があります。そのように配信するための Directory Proxy Server の設定方法については、「Operational Affinity Algorithm for Global Account Lockout」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

レプリケーショントポロジ内で厳密にローカルなロックアウトポリシーを維持するには、version 5.2 のパスワードポリシーとの互換性を維持する必要があります。その状況では、有効なポリシーがデフォルトのパスワードポリシーであってはいけません。また、読み取り専用のコンシューマにバインドを制限することによっても、ローカルロックアウトを実現することができます。

グローバルアカウントロックアウトの動作方法の詳細については、「Global Account Lockout」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

外部認証のマッピングおよびサービス

Directory Server は、ネットワークユーザーアカウントを Directory Server ユーザーアカウントに関連付けるユーザーアカウントホストマッピングを提供します。この機能を使えば、両方のユーザーアカウントの管理が容易になります。ホストマッピングは、外部で認証されたユーザーの場合に必要となります。

プロキシ承認

プロキシ承認は、特殊な形態のアクセス制御です。プロキシ承認またはプロキシ認証では、アプリケーションが、特定のユーザー名/パスワードの組み合わせを使ってサーバーへのアクセスを取得することを強制されます。

プロキシ承認では、管理者は、ある通常ユーザーの ID を引き受けることで Directory Server へのアクセスを要求できます。管理者は、自身の資格情報を使ってディレクトリにバインドし、その通常ユーザーの権利を許可されます。この引き受ける ID は、「プロキシユーザー」と呼ばれます。そのユーザーの DN は、「プロキシ DN」と呼ばれます。プロキシユーザーは通常のユーザーとして評価されます。プロキシユーザーのエントリがロックまたは無効化されているか、そのパスワードの有効期限が切れていると、アクセスが拒否されます。

プロキシメカニズムの利点は、Directory Server にアクセスしようとする複数のユーザーに対して、LDAP アプリケーションが単一のバインドを使ってサービスを提供できるという点にあります。各ユーザーがわざわざバインドおよび認証する代わりに、クライアントアプリケーションが Directory Server にバインドし、プロキシの権限を使用します。

詳細については、Chapter 7, 「Directory Server Access Control,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

パスワードポリシーの設計

パスワードポリシーは、システム内でパスワードがどのように管理されるかを規定した規則の集合です。Directory Server は、デフォルトのパスワードポリシーはもちろん、複数のパスワードポリシーもサポートします。

パスワードポリシーのいくつかの要素は設定可能であるため、組織のセキュリティー要件に合ったポリシーを設計できます。パスワードポリシーの設定については、Chapter 8, 「Directory Server Password Policy,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。パスワードポリシーの設定時に利用可能な個々の属性については、`pwpolicy`(5dssd) のマニュアルページを参照してください。

この節は、次の項目で構成されています。

- パスワードポリシーのオプション
- レプリケーション環境でのパスワードポリシー
- パスワードポリシーの移行

パスワードポリシーのオプション

次のようなパスワードポリシーのオプションがあります。

- デフォルトパスワードポリシーが適用されます。このデフォルトポリシーのパラメータは、変更可能です。
- 別の特殊なパスワードポリシーを特定のユーザーに適用できます。
- CoS およびロール機能を使用することで、別の特殊なパスワードポリシーを一連のユーザーに適用できます。パスワードポリシーは、スタティックグループには適用できません。

レプリケーション環境でのパスワードポリシー

デフォルトパスワードポリシーの設定情報はレプリケートされません。代わりに、それはサーバーインスタンスの設定の一部になっています。デフォルトパスワードポリシーを変更すると、トポロジ内のすべてのサーバー上でそれと同じ変更が施されます。レプリケート「される」パスワードポリシーが必要な場合は、レプリケートされるディレクトリツリー内に、特殊なパスワードポリシーを定義する必要があります。

ユーザーエントリに格納されているパスワード情報のすべてがレプリケートされます。この情報には、現在のパスワード、パスワード履歴、パスワードの有効期限などが含まれます。

レプリケートされた環境では、パスワードポリシーによる次の影響を考慮してください。

- パスワードの期限切れが近づいたユーザーは、パスワードを変更するまで、バインドするすべてのレプリカから警告を受信します。
- あるユーザーがパスワードを変更しても、その新しいパスワードがすべてのレプリカ上で更新されるまでに、しばらく時間がかかる可能性があります。あるユーザーがパスワードを変更してすぐに、新しいパスワードを使ってコンシューマレプリカの 1 つにバインドし直した、といった状況が発生する可能性があります。この場合、レプリカが更新されたパスワードを受信するまで、バインドがおそらく失敗します。この状況を緩和するには、優先順位付きレプリケーションを使ってパスワード変更が最初にレプリケートされるように強制します。

パスワードポリシーの移行

Directory Server Enterprise Edition のパスワードポリシーの設定は、5.x バージョンの Directory Server で提供されていたパスワードポリシーの設定とは異なります。異なるバージョンの Directory Server が稼働するサーバーがトポロジ内に含まれている場合には、「New Password Policy」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide 』を参照し、パスワードポリシーの設定の移行方法について確認してください。

Windows とのパスワードの同期

Identity Synchronization for Windows は、Directory Server と Windows との間で、パスワードを含むユーザーアカウント情報を同期させます。Windows Active Directory と Windows NT の両方がサポートされています。Identity Synchronization for Windows は、スケーラビリティーの高いセキュリティー機能の豊富なパスワード同期ソリューションを、小規模、中規模、および大規模企業向けに構築する際に役立ちます。

このソリューションが提供する機能は次のとおりです。

- Active Directory、Windows NT、および Directory Server 間における同期アカウントの作成、変更、無効化、および削除
- ネイティブパスワード変更を同期させるための、プロプライエタリな異種ディレクトリソースとの統合

配備内での Identity Synchronization for Windows の使用方法の詳細については、『Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide 』を参照してください。

暗号化手法の決定

暗号化は、転送中のデータや格納されたデータを保護する際に役立ちます。この節では、次のデータ暗号化手法について説明します。

- 134 ページの「SSL による接続のセキュリティー保護」
- 135 ページの「格納された属性の暗号化」

SSL による接続のセキュリティー保護

セキュリティー設計には、ユーザーを特定するための認証スキーマ方式や、情報を保護するためのアクセス制御方式以外の要素も含まれます。サーバーとクライアントアプリケーションとの間でネットワーク経由で送信される情報の完全性も保護する必要があります。

ネットワーク上で安全な通信を可能にするために、SSL (Secure Sockets Layer) 上で LDAP プロトコルと DSML プロトコルの両方を使用することができます。SSL が設定および有効化されると、クライアントはある専用のセキュアポートに接続します。そして、SSL 接続の確立後にすべての通信が暗号化されます。Directory Server と Directory Proxy Server は、Start Transport Layer Security (Start TLS) 制御もサポートします。Start TLS を使えば、クライアントは標準の LDAP ポート上で暗号化接続を開始できます。

Directory Server の SSL と TLS の概要については、Chapter 2, 「Directory Server Security,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

格納された属性の暗号化

属性の暗号化は、格納されたデータの保護に関係します。この節では、属性の暗号化機能について説明し、次のトピックを取り上げます。

- 135 ページの「属性暗号化とは」
- 136 ページの「属性暗号化の実装」
- 137 ページの「属性の暗号化とパフォーマンス」

属性暗号化とは

Directory Server Enterprise Edition は、パスワード認証、証明書に基づく認証、SSL、プロキシ承認など、アクセスレベルでデータを保護するためのさまざまな機能を提供しています。ただし、データベースファイル、バックアップファイル、および LDIF ファイルに格納されたデータも保護する必要があります。属性を暗号化する機能を使用することで、格納されている機密情報にユーザーがアクセスできないように保護できます。

属性の暗号化を使えば、特定の属性を暗号化された形式で格納できます。属性の暗号化はデータベースレベルで設定されます。したがって、ある属性を暗号化すると、その属性がデータベース内のすべてのエントリで暗号化されます。属性の暗号化はエントリレベルではなく属性レベルで行われるため、エントリ全体を暗号化するには、すべての属性を暗号化する必要があります。

属性の暗号化を使えば、データを暗号化された形式で別のデータベースにエクスポートすることもできます。属性の暗号化の目的は、格納またはエクスポートされる機密データを保護することです。したがって、この暗号化は常に可逆です。検索要求の結果として返された場合は、暗号化された属性は復号化されます。

次の図は、データベースに追加されるユーザーエントリを示しています。ここで設定されている属性暗号化は、`salary` 属性を暗号化します。

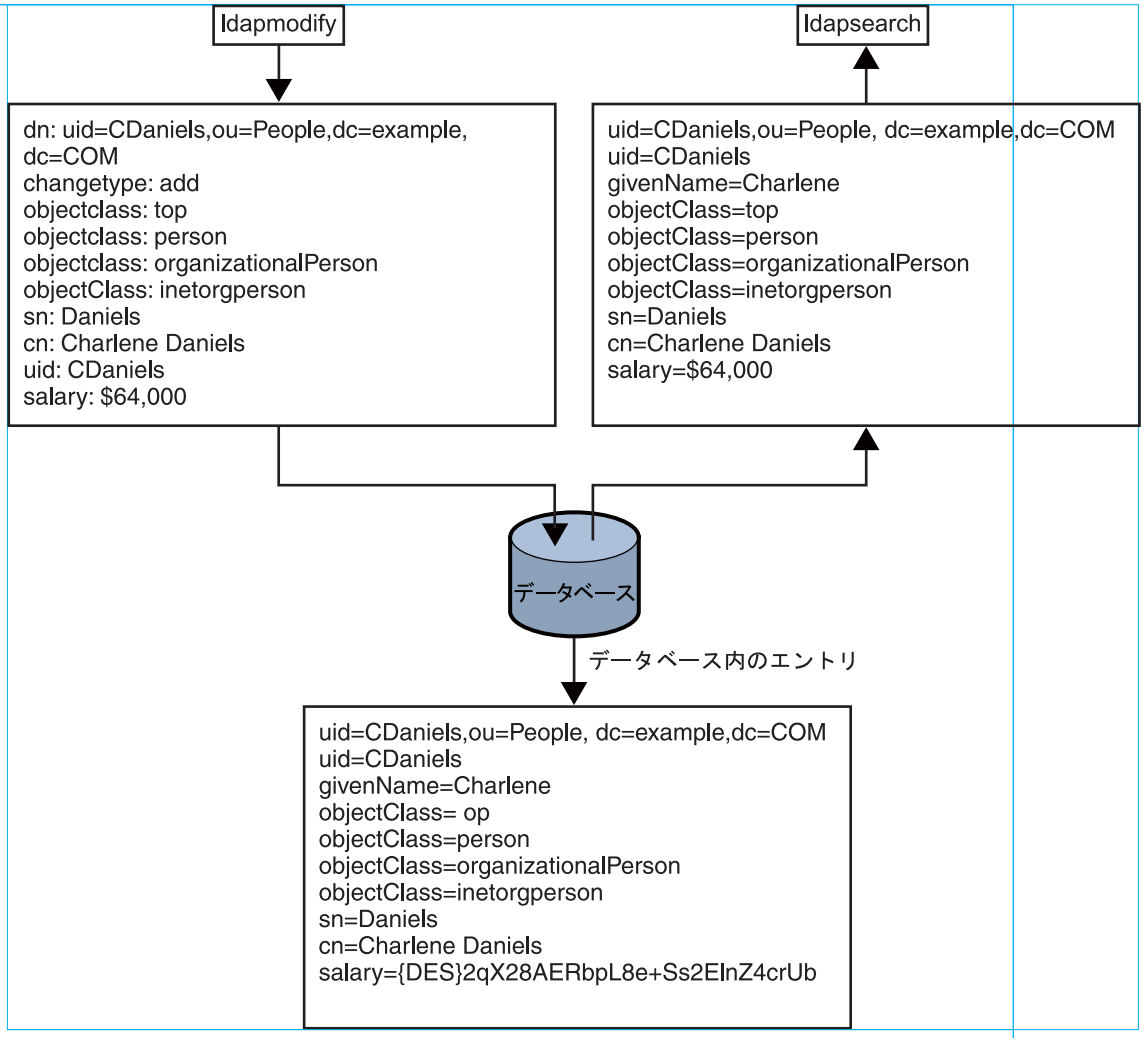


図 7–1　属性暗号化のロジック

属性暗号化の実装

属性の暗号化機能は、広範な暗号化アルゴリズムをサポートしています。異なるプラットフォーム間での移植性が保証されています。属性の暗号化は追加のセキュリティー手段として、サーバーの SSL 証明書の非公開鍵を使って自身の鍵を生成します。その後、この鍵は、暗号化および復号化処理を実行するために使用されます。属性を暗号化できるためには、サーバーが SSL 上で稼働している必要があります。SSL 証明書とその非公開鍵はデータベース内に安全に格納され、パスワードで保護されます。この鍵データベースのパスワードはサーバーへの認証時に必要になりま

す。サーバーは、この鍵データベースのパスワードにアクセスできるユーザーであれば、だれもが復号化されたデータのエクスポートを承認されたものとみなします。

属性の暗号化が保護するのは格納された属性だけである点に注意してください。これらの属性をレプリケートする場合には、転送中の属性を保護できるよう、SSL 上でレプリケーションを設定する必要があります。

属性の暗号化を使用する場合、バイナリコピー機能を使ってあるサーバーから別のサーバーを初期化することはできません。

属性の暗号化とパフォーマンス

属性の暗号化を使えばデータのセキュリティーを高められますが、この機能はパフォーマンスに影響を与えます。属性の暗号化は、機密性の特に高い属性を暗号化するためだけに使用してください。

機密データには、インデックスファイル経由で直接アクセスできます。したがって、暗号化する属性に対応するインデックスキーを暗号化することで、それらの属性が完全に保護されるようにする必要があります。インデックスキーを暗号化することによる追加コストがないとしても、インデックスを作成すること自体がすでにパフォーマンスに影響を及ぼします。したがって、データベースにはじめてデータをインポートまたは追加する「前」に、属性の暗号化を設定してください。こうすることで、暗号化された属性には最初からインデックスが付けられます。

ACI によるアクセス制御の設計

アクセス制御では、特定の情報へのアクセス権を一部のクライアントに与え、その他のクライアントには与えないように指定することができます。アクセス制御は、1 つまたは複数のアクセス制御リスト (ACL) を使用して実装します。ACL は、指定されたエントリとその属性へのアクセス権を許可または拒否する一連のアクセス制御命令 (ACI) から構成されます。アクセス権には、読み取り、書き込み、検索、プロキシ、追加、削除、比較、インポート、およびエクスポートを行う機能が含まれます。

ACL を使用すると、次の項目に対する権限を設定できます。

- ディレクトリ全体
- ディレクトリの特定のサブツリー
- ディレクトリ内の特定のエントリ
- 特定のエントリの属性セット
- 特定の LDAP 検索フィルタにマッチするすべてのエントリ

また、特定のユーザー、グループに属するすべてのユーザー、およびディレクトリのすべてのユーザーに対する権限を設定できます。さらに、IP アドレスや DNS 名など、ネットワークの場所に対してアクセス権を定義することもできます。

この節では、Directory Server のアクセス制御メカニズムの概要を説明します。アクセス制御や ACI の設定方法の詳細については、Chapter 7, 「Directory Server Access Control,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。アクセス制御メカニズムのアーキテクチャーについては、「How Directory Server Provides Access Control」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

デフォルト ACI

Directory Server のデフォルト動作は、アクセスを許可する特定の ACI が存在していないかぎりアクセスを拒否する、というものです。したがって、ACI が 1 つも定義されていなければ、サーバーへのすべてのアクセスが拒否されます。

Directory Server をインストールしたり新しいサフィックスを追加したりすると、ルート DSE 内にいくつかのデフォルト ACI が自動的に定義されます。この ACI は、セキュリティー要件に合うように変更できます。

デフォルト ACI やそれらの変更方法の詳細については、「How Directory Server Provides Access Control」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

ACI の適用範囲

Directory Server 6.x には、ACI の適用範囲に対する主な変更が 2 つ含まれています。

- **ACI** の適用範囲を指定する機能**:**Directory Server 5.x では、ACI の適用範囲を指定できませんでした。ACI は自動的に、その ACI の格納先となるエントリとそのすべてのサブツリーに対して適用されていました。このため、場合によっては「拒否」ACI を使用する必要がありました。拒否 ACI と許可 ACI が単一のエントリに適用される場合は特に、拒否 ACI の管理が困難になる可能性があります。

  Directory Server 6.x では ACI の適用範囲を指定できます。つまり、「許可」ACI を使ってアクセス制御を行えます。拒否ベースのアクセス制御の使用が避けられなかったり、そうしたアクセス制御を使用したほうが設定が容易になったりする場合もありますが、拒否 ACI の使用は基本的にはお勧めできません。ACI の適用範囲の指定方法については、Chapter 7,「Directory Server Access Control,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

- ルート **ACI** がルートエントリとそのサブツリー全体に適用されるようになった**:**Directory Server 5.x では、ルート DSE に格納された ACI はルートエントリのみに適用され、その子には適用されませんでした。ほかの任意のエントリ内の ACI は、その ACI の格納先エントリとそのすべてのサブツリーに対して適用されていました。Directory Server Enterprise Edition では、ルートエントリ内に配置された ACI が、ほかの任意の場所に配置された ACI と同様に扱われます。

  新しいルート ACI には、明確なセキュリティー上の利点が 1 つあります。管理者は、特定の操作を実行する際に Directory Manager としてバインドする必要がなくなりました。SSL などの強い認証によるバインドを管理者に強制できるようになりました。ルートエントリ「だけに」適用することを目的とした ACI を設定するには、その ACI の適用範囲を具体的に `base` に設定する必要があります。

ACI の適用範囲の変更は移行に影響を与えます。5.x バージョンの Directory Server から Directory Server 6.x に移行する場合には、「Changes to ACIs」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide 』を参照してください。

実効権限に関する情報の取得

Directory Server 6.x が提供するアクセス制御モデルでは、異なる多くのメカニズムを通じてユーザーにアクセスを許可できます。しかしながら、この柔軟性のために、セキュリティーポリシーがかなり複雑になる可能性があります。IP アドレス、マシン名、時刻、認証方法など、いくつかのパラメータを使用すれば、あるユーザーのセキュリティーコンテキストを定義できます。

セキュリティーポリシーを単純化するために、Directory Server では、指定されたディレクトリエントリおよび属性に対する、ある特定のユーザーの実行アクセス権

限を要求および一覧表示できます。詳細については、「Viewing Effective Rights」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

ACIの使用に関するヒント

次の各ヒントを参考にすれば、ディレクトリのセキュリティーモデルの単純化とパフォーマンスの改善を図れる可能性があります。

- ディレクトリ内のACIの数を最小化し、可能であればマクロACIを使用する。

  Directory Serverは50,000件を超えるACIを評価可能ですが、多数のACI文を管理するのは容易ではありません。また、過剰なACIはメモリー消費にも悪影響を及ぼす可能性があります。

- 許可権限と拒否権限のバランスを図る。

  デフォルトの規則は、アクセスを具体的に許可されていないすべてのユーザーのアクセスを拒否することです。ただし、ツリーのルートの近くでアクセスを許可する1つのACIを使用し、リーフエントリの近くで少数の拒否ACIを使用すれば、ACIの数を減らすことができます。このようにすると、最下位のエントリの近くでアクセスを許可するACIが必要以上に多くなることがなくなります。

- ACIでは最小の属性セットを指定する。

  オブジェクトの属性の一部でアクセスを許可または拒否する場合、許可または拒否する属性の最小リストを決めます。次に、最小の属性リストを管理するようにACIを表します。

  たとえば、ユーザーオブジェクトクラスに多くの属性が含まれている場合を考えます。いくつかの属性だけをユーザーが更新できるようにするには、その少数について書き込み権限を許可するACIを作成します。1つか2つの属性以外のすべてをユーザーが更新できるようにする場合は、それらの1つか2つの属性について書き込み権限を拒否するACIを作成します。

- LDAP検索フィルタは慎重に使用する。

  検索フィルタは、アクセス管理対象のオブジェクトを直接指定しません。したがって、特にディレクトリが複雑になるにつれて、検索フィルタによって予期しない結果が得られる可能性があります。ACI内で検索フィルタを使用する場合、その同じフィルタを使って`ldapsearch`操作を実行してください。このアクションにより、その変更の結果がユーザーのディレクトリにとって何を意味するのかを確認できます。

- ディレクトリツリーの別の部分でACIを重複させない。

  オーバーラップしているACIを探します。グループ書き込みアクセス権を`commonName`および`givenName`属性に対して許可するACIが、ディレクトリのルート位置に存在するとします。さらに、それと同じグループ書き込みアクセス権を`commonName`属性に対してだけ許可する別のACIも存在するとします。この場合、

そのグループに対して 1 つの属性のみの書き込みアクセス権を与えるように、ACI を修正することを検討してください。

ディレクトリが複雑になるほど、ACI のオーバーラップが間違って発生する確率も高くなります。ACI のオーバーラップを回避すれば、セキュリティーの管理が容易になり、ディレクトリ内の ACI の合計数も減少します。

- ACI に名前を付ける。

  ACI に名前を付けることは省略可能ですが、各 ACI に意味のある短い名前を付ければ、セキュリティーモデルの管理が容易になります。

- ユーザーエントリの標準属性を使用して、アクセス権限を決める。

  可能なかぎり、すでに標準ユーザーエントリの一部となっている情報を使用して、アクセス権限を決めてください。特別な属性を作成する必要がある場合は、それらの属性をロールまたはサービスクラス (CoS) の定義の一部として作成することを検討します。ロールと CoS の詳細については、Chapter 8, 「Directory Server Groups and Roles,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』およびChapter 9, 「Directory Server Class of Service,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

- ACI を互いにできるだけ近い位置にまとめる。

  ACI の配置をディレクトリのルートポイントとディレクトリの主な分岐点に限定します。ACI を編成してグループにまとめると、ACI リストの全体を管理しやすくなり、ACI の合計数を最小に保つことができます。

- 二重否定は使用しないようにする (バインド DN が `cn=Joe` と等しくない場合の書き込みの拒否など)。

  サーバーはこの構文を理解できますが、管理者にとっては混乱の元となります。

接続規則によるアクセス制御の設計

接続規則を使えば、選択されたクライアントが Directory Server への接続を確立するのを防ぐことができます。接続規則の目的は、悪意のあるクライアントや不適切に設計されたクライアントによって引き起こされるサービス拒否攻撃を防ぐことです。そうしたクライアントは Directory Server に接続し、そのサーバーを要求で溢れさせます。

接続規則は TCP レベルで確立されますが、それは「TCP ラッパー」を定義することで行われます。TCP ラッパーの詳細については、「Client-Host Access Control Through TCP Wrapping」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

Directory Proxy Server によるアクセス制御の設計

Directory Proxy Server の接続ハンドラは、サーバーに接続を試みるクライアントの分類を可能にするアクセス制御の方法を提供します。この方法では、接続がどのように分類されたかに基づいて、実行可能な操作を制限できます。

この機能を使えば、ある指定された IP アドレスから接続するクライアントだけにアクセスを制限したりすることができます。次の図は、Directory Proxy Server の接続ハンドラを使って特定の IP アドレスからの書き込み操作を拒否する方法を示したものです。

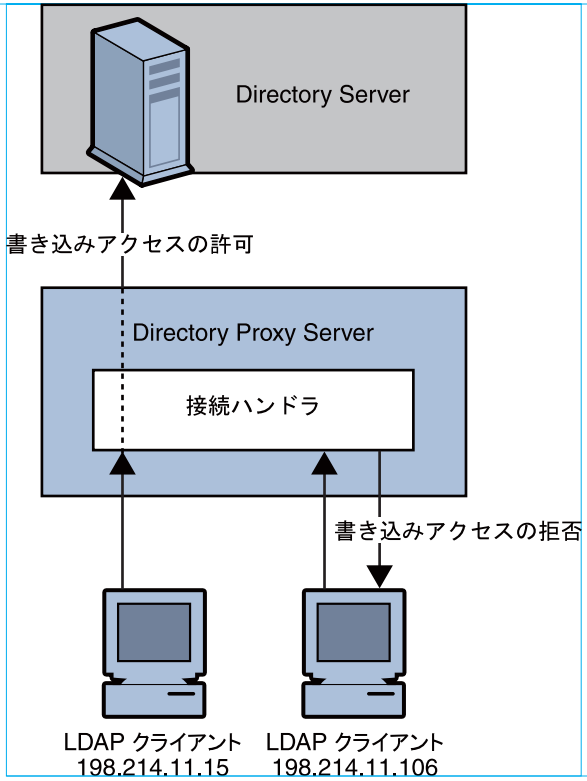


図 7–2　Directory Proxy Server の接続ハンドラのロジック

接続ハンドラの動作

接続ハンドラは、一連の条件と一連のポリシーから構成されます。Directory Proxy Server は、ある接続の起点属性をクラスの条件とマッチングさせることで、その接続のクラスメンバーシップを決定します。接続があるクラスに一致すると、Directory Proxy Server はそのクラスに格納されているポリシーをその接続に適用します。

接続ハンドラの条件には、次の情報を含めることができます。

- クライアントの物理アドレス
- ドメイン名またはホスト名
- クライアント DN のパターン
- 認証方法
- SSL

接続ハンドラには次のポリシーを関連付けることができます。

- 管理制限ポリシー:たとえば、ある特定クラスのクライアントからの接続数を制限できるようにします。
- コンテンツ適応ポリシー:属性の名前変更など、接続が実行できる操作の種類を制限できるようにします。
- データ分散ポリシー:ある接続で特定の配布方式を使用できるようにします。

Directory Proxy Server の接続ハンドラやその設定方法の詳細については、Chapter 20, 「Connections Between Clients and Directory Proxy Server,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

エントリの安全なグループ化

ロールと CoS では、セキュリティーに関する特別な考慮が必要となります。

ロールの安全な使い方

セキュリティーの状況によっては、ロールの使用が適していない場合があります。ロールを作成するときは、エントリへのロールの割り当てやエントリからのロールの削除がどの程度簡単にできるかを考慮します。ユーザーは場合によっては、自身をロールに追加したりロールから削除したりできるべきです。ただし、一部のセキュリティーコンテキストでは、そうしたオープンなロールは不適切です。詳細については、「Directory Server Roles」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

CoS の安全な使い方

読み取り用のアクセス制御は、エントリの実際の属性と仮想属性の両方に適用されます。サービスクラス (CoS) メカニズムによって生成される仮想属性は、通常の属性と同様に読み取られます。したがって、仮想属性には、同じ方法で読み取り保護を付与すべきです。ただし、CoS 値をセキュリティー保護するには、CoS 値が使用する次のすべての情報ソースを保護する必要があります。定義エントリ、テンプレートエントリ、およびターゲットエントリ。更新操作についても同じことが言えます。

各情報ソースから生成される値を保護するには、それらのソースへの書き込みアクセスを制御する必要があります。詳細については、Chapter 9,「Directory Server Class of Service,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

ファイアウォールの使用

ファイアウォールテクノロジは通常、内部ネットワークを出入りするネットワークトラフィックをフィルタリングまたはブロックするために使用されます。LDAP 要求が境界ファイアウォールの外側から送られてくる場合には、ファイアウォールの通過を許可するポートとプロトコルを指定する必要があります。

指定するポートとプロトコルはディレクトリのアーキテクチャーによって異なります。一般的な規則として、ポート 389 および 636 上の TCP および UDP 接続を許可するように、ファイアウォールを設定する必要があります。

Directory Server が稼働しているサーバーに、ホストベースのファイアウォールをインストールできます。ホストベースのファイアウォールの規則は、境界防御ファイアウォールの規則に似ています。

root 以外のユーザーとして実行

サーバーインスタンスを root 以外のユーザーとして作成および実行できます。サーバーインスタンスを `root` 以外のユーザーとして実行することで、悪用によって生じる可能性のあるあらゆる損害を制限できます。さらに、`root` 以外のユーザーとして実行されるサーバーは、オペレーティングシステムのアクセス制限メカニズムの処理対象となります。

その他のセキュリティー関連資料

セキュリティー保護されたディレクトリの設計については、以下のリソースを参照してください。

- Sun Developer セキュリティーリソース
  http://developers.sun.com/techtopics/security/index.html
- 『*Understanding and Deploying LDAP Directory Services*』。T. Howes, M. Smith, G. Good、 Macmillan Technical Publishing 発行、1999 年
- SecurityFocus.com http://www.securityfocus.com/
- Computer Emergency Response Team (CERT) Coordination Center http://www.cert.org/

◆ ◆ ◆

第 8 章

8 管理と監視の要件の特定

Directory Server Enterprise Edition の管理は、5.x バージョンの Directory Server から大幅に変更されています。これらの変更は、『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』で詳細に説明されています。

この章では、これらの変更の概要を示したあと、配備の計画段階で行う必要のある管理上の決定について説明します。

Directory Server Enterprise Edition の管理モデル

Directory Server Enterprise Edition では、管理者がインスタンスの作成や管理をより細かく制御できるようになりました。この制御は、2 つの新しいコマンド `dsadm` と `dsconf` を使用して実現されます。これらのコマンドは、以前の `directoryserver` コマンドで提供されていたすべての機能に加え、追加の機能を提供します。

`dsadm` コマンドを使用すると、管理者は Directory Server インスタンスを作成、起動、および停止できます。このコマンドは、Directory Server インスタンスへのファイルシステムアクセスが必要なすべての操作を行います。このコマンドは、インスタンスをホストするマシン上で実行してください。インスタンスへの LDAP アクセスまたはエージェントへのアクセスが必要な操作は実行しません。

新しい管理モデルでは、Directory Server インスタンスの「*ServerRoot*」への関連付けはなくなりました。各 Directory Server インスタンスは、通常のスタンドアロンディレクトリと同じ方法で操作できるスタンドアロンディレクトリです。

`dsconf` は、`cn=config` への書き込みアクセスが必要な管理操作を行います。`dsconf` コマンドは、LDAP クライアントです。アクティブな Directory Server インスタンス上で

のみ実行できます。このコマンドはリモートから実行することができ、それによって、管理者は複数のインスタンスを単一のリモートマシンから設定できます。

Directory Proxy Server は、dpadm と dpconf という、2 つの類似のコマンドを提供しています。dpadm コマンドを使用すると、管理者は Directory Proxy Server インスタンスを作成、起動、および停止できます。dpconf コマンドを使用すると、管理者は LDAP を使用して Directory Proxy Server を設定し、Directory Proxy Server を介して Directory Server 構成にアクセスできます。

これらのコマンド行ユーティリティーに加えて、Directory Server Enterprise Edition は Java Web コンソールにも統合されています。このコンソールを使用すると、Directory Server Enterprise Edition やその他の Sun 製品を集中管理のユーザーインタフェースから管理できます。Directory Service Control Center (DSCC) は、Directory Server と Directory Proxy Server の管理専用に用意された、Java Web コンソールのサービスです。DSCC は、コマンド行ユーティリティーと同じ機能のほか、複数のサーバーを一度に設定できるウィザードを提供しています。DSCC はまた、レプリケーショントポロジをグラフィカルに監視できる、レプリケーショントポロジの描画ツールも提供します。このツールでは、個々のマスター、ハブ、コンシューマや、それらの間のレプリケーションアグリーメントがリアルタイムに表示されるため、レプリケーションの監視が簡略化されます。

リモート管理

前の節で説明した Directory Server Enterprise Edition の管理モデルでは、トポロジ内の任意の Directory Server または Directory Proxy Server のリモート管理を行うこともできます。サーバーは、コマンド行ユーティリティーと Java Web コンソールの両方を使用して、リモートから管理できます。

dsadm および dpadm ユーティリティーはリモートからは実行できません。これらのユーティリティーは、管理対象のサーバーインスタンスと同じマシンにインストールして実行してください。dsadm と dpadm で提供される機能の詳細については、dsadm(1M) と dpadm(1M) のマニュアルページを参照してください。

dsconf および dpconf ユーティリティーは、リモートから実行できます。dsconf と dpconf で提供される機能の詳細については、dsconf(1M) と dpconf(1M) のマニュアルページを参照してください。

次の図は、新しい管理モデルによるリモート管理の容易性を示しています。この図は、コンソールコマンドや設定コマンドを Directory Server および Directory Proxy Server インスタンスからリモートにインストールして実行できることを示しています。管理コマンドは、これらのインスタンスに対してローカルに実行する必要があります。

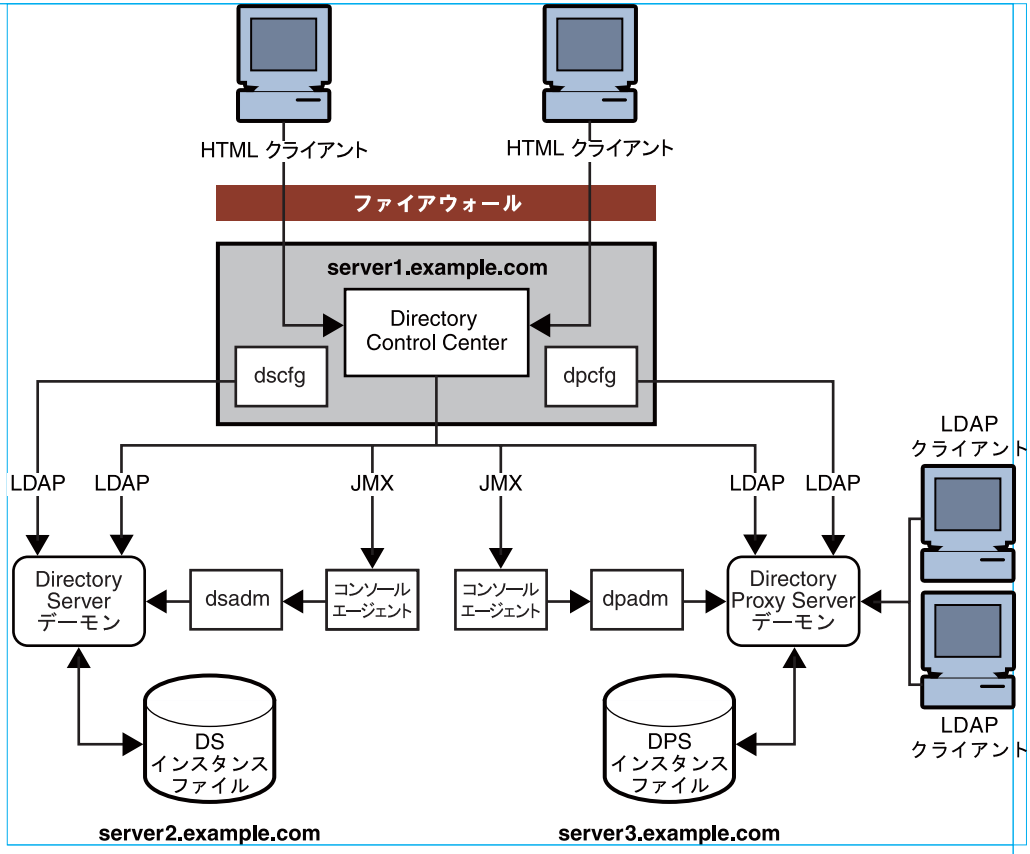


図 8–1　Directory Server Enterprise Edition の管理モデル

バックアップと復元のポリシーの設計

データ破壊やデータ損失をともなう障害の状況では、データの最新のバックアップを保有していることが不可欠です。できるだけ、ほかのサーバーからのサーバーの再初期化は避けてください。データのバックアップ方法については、Chapter 9,「Directory Server Backup and Restore,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

ここでは、バックアップと復元の戦略を計画する上で考慮すべき事項の概要について説明します。

高レベルのバックアップと復旧の原則

バックアップ戦略を設計するときは、次の高レベルの原則を適用します。

- バックアップする必要のあるデータを特定します。

  Directory Server Enterprise Edition の場合、このデータには次のものが含まれます。

  - 共有されているバイナリとプラグイン
  - 証明書データベースのファイル
  - 設定ファイル
  - ログファイルと更新履歴ログデータベース
  - スキーマファイル
  - ユーザーデータ
- バックアップと復元の戦略に、ハードウェア、オペレーティングシステム、およびソフトウェアコンポーネントが含まれている必要があります。
- バイナリバックアップまたは LDIF バックアップを保持するかどうかを決定します。

  一般には、この両方を保持することをお勧めします。詳細については、149 ページの「バックアップ方法の選択」および 152 ページの「復元方法の選択」を参照してください。
- バックアップおよび復旧ツール関連の自動化を確立するとともに、自動スクリプトが保守されている必要があります。

  この戦略によって、緊急にバックアップからの復元が必要になった場合の不必要な遅延が回避されます。
- 保持とローテーションの戦略を決定します。

  この戦略には、バックアップを実行する頻度と、バックアップを保持する期間が含まれます。バックアップの保持とローテーションを決定する場合は、パージ遅延と、それによる、レプリケートされるトポロジ内のバックアップへの影響に注意してください。サプライヤで変更が発生すると、それらの変更は更新履歴ログに記録されます。更新履歴ログを空にするための方法がないと、そのサイズは、すべての使用可能なディスク容量が更新履歴ログによって消費されるまで増え続けます。デフォルトでは、これらの変更は 7 日ごとに消去されます。この期間をパージ遅延と呼びます。変更が消去されると、その変更をレプリケートすることはできなくなります。そのため、データベースは、少なくともパージ遅延と同じ頻度でバックアップされる必要があります。
- 単にシステムのバックアップと復旧を実行するのではなく、Directory Server Enterprise Edition で提供されているバックアップおよび復旧ツールを使用します。

バックアップ方法の選択

Directory Server Enterprise Edition では、バイナリバックアップと、LDIF ファイルへのバックアップという 2 つの方法でデータをバックアップできます。どちらの方法にも利点と制限があるため、効果的なバックアップ戦略を計画するには、それぞれの方法について理解することが役立ちます。

バイナリバックアップ

バイナリバックアップは、データベースファイルのコピーを生成するものであり、ファイルシステムレベルで実行されます。バイナリバックアップの出力は、すべてのエントリ、インデックス、更新履歴ログ、トランザクションログを含むバイナリファイルのセットです。バイナリバックアップには、設定データは含まれません。

バイナリバックアップは、次のいずれかのコマンドを使用して実行されます。

- `dsadm backup` はオフラインで、つまり Directory Server インスタンスが停止されているときに実行してください。このコマンドは、Directory Server インスタンスを含むローカルサーバーで実行してください。
- `dsconf backup` は、オンラインで実行でき、さらに Directory Server インスタンスのリモートからも実行できます。

バイナリバックアップには、次のような利点があります。

- すべてのサフィックスを一度にバックアップできる。
- LDIF へのバックアップと比較して、バイナリバックアップは格段に高速である。
- レプリケーション更新履歴ログがバックアップされる。

バイナリバックアップには、1 つの制限があります。バイナリバックアップからの復元は、「同一の」設定のサーバーだけでしか実行できません。

この制限は次を意味します。

- 両方のマシンが同じハードウェア、同じオペレーティングシステム (サービスパックやパッチも含まれる) を使用している必要がある。
- 両方のマシンに同じバージョンの Directory Server (32 ビットまたは 64 ビットのバイナリ形式、サービスパック、パッチレベルも含まれる) がインストールされている必要がある。
- 両方のサーバーは、同じサフィックスに分岐する同じディレクトリツリーを持つ必要がある。すべてのサフィックスのデータベースファイルをまとめてコピーする必要があり、サフィックスを個別にコピーすることはできない。
- 両方のサーバーの各サフィックスには同じインデックス (仮想リスト表示 (VLV) インデックスも含まれる) が設定されている必要がある。サフィックスのデータベースファイルの名前は同じである必要がある。

- 各サーバーに、レプリカとして同じサフィックスが設定されている必要があります。部分レプリケーションが設定されている場合、部分レプリケーションはすべてのマスターサーバーで同じように設定されている必要がある。
- どちらのサーバーでも、属性の暗号化は使用できません。

少なくとも、整合性のあるマシンの各セットに対して定期的なバイナリバックアップを実行してください。整合性のあるマシンとは、前述のように同一の設定を持つマシンのことです。

注 – ローカルバックアップからの復元の方が簡単なため、各サーバーでバイナリバックアップを実行してください。

この章の残りの図では、次の略語を使用します。

M = マスターレプリカ
RA = レプリケーションアグリーメント

次の図は、M1 と M2 が同一の設定を持ち、M3 と M4 が同一の設定を持つことを前提としています。この例では、M1 と M3 でバイナリバックアップが行われます。障害が発生した場合は、M1 または M2 を M1 (db1) のバイナリバックアップから復元できます。M3 または M4 を M3 (db2) のバイナリバックアップから復元できます。M1 と M2 を M3 のバイナリバックアップから復元することはできません。M3 と M4 を M1 のバイナリバックアップから復元することはできません。

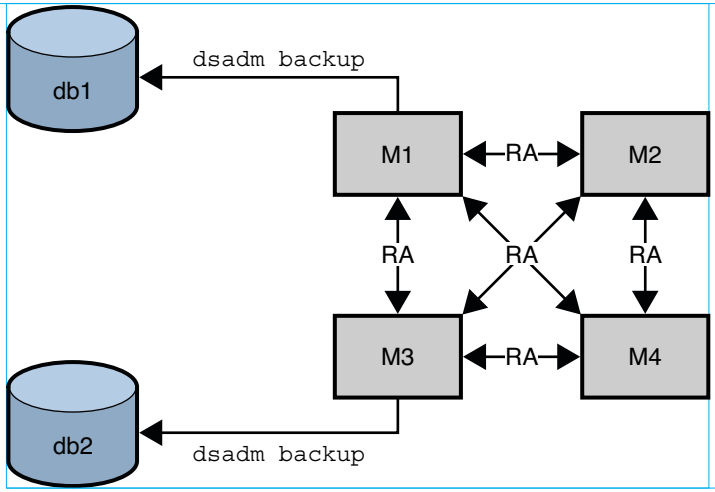


図 8–2 オフラインのバイナリバックアップ

バイナリバックアップのコマンドの使用方法の詳細については、「Binary Backup」in『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

LDIF へのバックアップ

LDIF へのバックアップはサフィックスレベルで行われます。LDIF へのバックアップの出力は、サフィックスに含まれているデータのコピーである、フォーマットされた LDIF ファイルです。このため、このプロセスはバイナリバックアップと比較して時間がかかります。

LDIF へのバックアップは、次のいずれかのコマンドを使用して実行されます。

- `dsadm export` はオフラインで、つまり Directory Server インスタンスが停止されているときに実行してください。このコマンドは、Directory Server インスタンスを含むローカルサーバーで実行してください。
- `dsconf export` は、オンラインで実行でき、さらに Directory Server インスタンスのリモートからも実行できます。

注 – これらのコマンドの実行時に `-Q` オプションを指定しないかぎり、レプリケーション情報はバックアップされます。

LDIF へのバックアップでは、`dse.ldif` 設定ファイルはバックアップされません。以前の設定を復元できるようにするには、このファイルを手動でバックアップしてください。

LDIF へのバックアップには、次のような利点があります。

- LDIF へのバックアップは、設定に関係なくどのサーバーからも実行できる。
- LDIF バックアップからの復元は、設定に関係なくどのサーバーからも実行できる。

LDIF へのバックアップには、1 つの制限があります。迅速なバックアップと復元が必要な状況では、LDIF へのバックアップでは時間がかかり過ぎる可能性があります。

トポロジの単一マスターで、レプリケートされた各サフィックスを LDIF に定期的にバックアップする必要があります。

次の図では、1 つのマスター (M1) でのみ、レプリケートされた各サフィックスに対して `dsadm export` が実行されています。

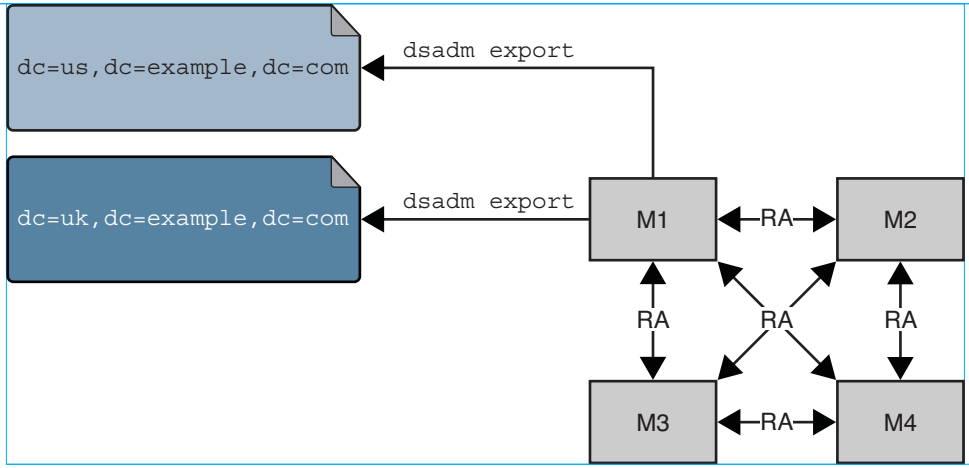


図 8–3 LDIF へのオフラインのバックアップ

LDIF へのバックアップのコマンドの使用方法については、「Backing Up to LDIF」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

復元方法の選択

Directory Server Enterprise Edition では、バイナリ復元と、LDIF ファイルからの復元という 2 つの方法でデータを復元できます。バックアップ方法と同様に、どちらの方法にも利点と制限があります。

バイナリ復元

バイナリ復元では、データベースレベルでデータがコピーされます。バイナリ復元は、次のいずれかのコマンドを使用して実行されます。

- `dsadm restore` はオフラインで、つまり Directory Server インスタンスが停止されているときに実行してください。このコマンドは、Directory Server インスタンスを含むローカルサーバーで実行してください。
- `dsconf restore` は、オンラインで実行でき、さらに Directory Server インスタンスのリモートからも実行できます。

バイナリ復元には、次のような利点があります。

- すべてのサフィックスを一度に復元できる。
- レプリケーション更新履歴ログが復元される。
- LDIF ファイルからの復元と比較して、バイナリ復元は格段に高速である。

バイナリ復元には、次のような制限があります。

- 復元は、同一の設定を持つサーバーでしか実行できない (149 ページの「バイナリバックアップ」を参照)。バイナリ復元機能を使用したデータの復元の詳細については、「Binary Restore」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。
- データベースが破壊されていることに気付かずにバイナリバックアップを実行した場合は、破壊されたデータベースが復元される危険性がある。バイナリバックアップは、データベースの現在の状態をありのままにコピーします。

マシンの設定が同一であり、実行時間を極力減らすことを一番の目的としている場合は、推奨される復元方法はバイナリ復元になります。

次の図は、M1 と M2 が同一の設定を持ち、M3 と M4 が同一の設定を持つことを前提としています。この例では、M1 または M2 を M1 (db1) のバイナリバックアップから復元できます。M3 または M4 を M3 (db2) のバイナリバックアップから復元できます。

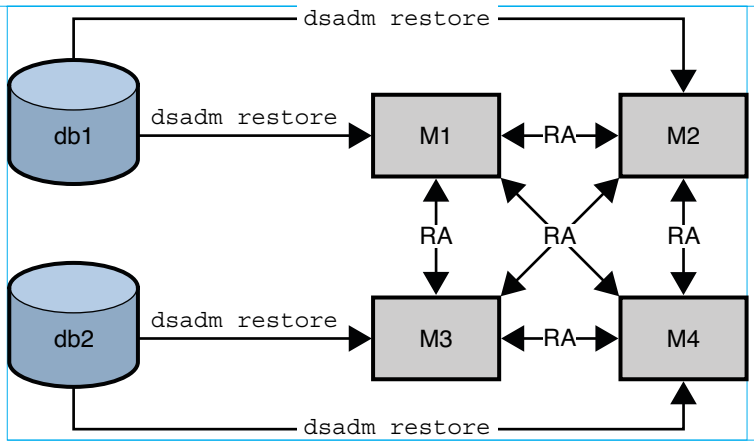


図 8–4 オフラインのバイナリ復元

LDIF からの復元

LDIF ファイルからの復元は、サフィックスレベルで実行されます。このため、このプロセスはバイナリ復元と比較して時間がかかります。LDIF からの復元は、次のいずれかのコマンドを使用して実行できます。

- `dsadm import` はオフラインで、つまり Directory Server インスタンスが停止されているときに実行してください。このコマンドは、Directory Server インスタンスを含むローカルサーバーで実行してください。
- `dsconf import` は、オンラインで実行でき、さらに Directory Server インスタンスのリモートからも実行できます。

LDIF ファイルからの復元には、次のような利点があります。

- このコマンドは、設定に関係なくどのサーバーからも実行できる。
- レプリケーショントポロジに関係なく、ディレクトリサービス全体の配備に単一の LDIF ファイルを使用できる。予定されているビジネスニーズに合わせてディレクトリサービスをダイナミックに拡張または縮小する場合に、この機能は特に便利である。

LDIF ファイルからの復元には、1 つの制限があります。迅速な復元が必要な状況では、この方法は時間がかかり過ぎる場合があります。LDIF ファイルからのデータの復元の詳細については、「Importing Data From an LDIF File」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

次の図では、1 つのマスター (M1) でのみ、レプリケートされた各サフィックスに対して `dsadmin import` が実行されています。

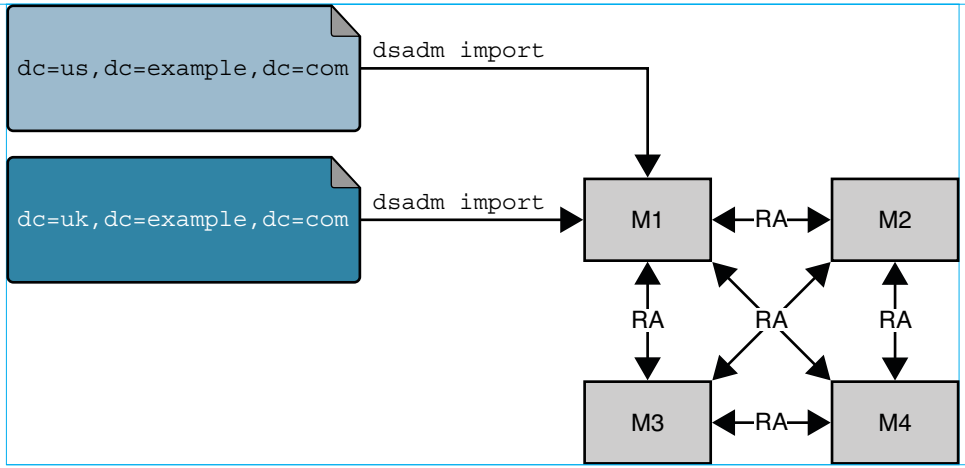


図 8–5 LDIF からのオフラインの復元

ロギング方法の設計

ロギングは、個別のサーバーレベルで管理および設定されます。ロギングは、デフォルトで有効になっていますが、配備の要件に従って再設定したり無効にしたりすることができます。ロギング方法の設計は、ハードウェア要件の計画に役立ちます。詳細については、81 ページの「Directory Server のハードウェアサイジング」を参照してください。

ここでは、Directory Server Enterprise Edition のロギング機能について説明します。

ロギングポリシーの定義

トポロジ内の各 Directory Server は、ロギング情報を次の 3 つのファイルに格納します。

- アクセスログ: サーバーに接続するクライアントと、要求された操作をリストします。
- エラーログ: サーバーエラーに関する情報を提供します。
- 監査ログ: サフィックスの変更と設定の変更に関する詳細情報を提供します。

トポロジ内の各 Directory Proxy Server は、ロギング情報を次の 2 つのファイルに格納します。

- アクセスログ: Directory Proxy Server に接続するクライアントと、要求された操作をリストします。
- エラーログ: サーバーエラーメッセージが含まれています。

Directory Server と Directory Proxy Server の両方のログファイルを次の方法で管理できます。

- ログファイル作成ポリシーの定義
- ログファイル削除ポリシーの定義
- ログファイルの手動での作成と削除
- ログファイルのアクセス権の定義

ログファイル作成ポリシーの定義

ログファイル作成ポリシーを使用すると、現在のログを定期的に保存し、新しいログファイルを開始することができます。ログファイル作成ポリシーは、Directory Control Center からか、またはコマンド行ユーティリティーを使用して Directory Server と Directory Proxy Server に対して定義できます。

ログファイル作成ポリシーを定義する場合は、次の点を考慮してください。

- 保持するログの個数

  このログ数に達すると、新しいログを作成する前に、フォルダ内のもっとも古いログファイルが削除されます。この値が 1 に設定されていると、ログはローテーションされず、かぎりなく増大します。
- 各ログファイルの最大サイズ (M バイト単位)

  ログファイルがこの最大サイズか、または次の項目で定義される最長有効期間に達すると、そのファイルは保存され、新しいログファイルへのロギングが開始されます。
- 現在のログファイルを保存する頻度

  デフォルトは毎日です。
- ログファイルをローテーションする 1 日の中の時間帯

時間に基づいてローテーションすると、各ログファイルが同じ期間をカバーするようになるため、ログの分析や傾向判断などの処理が容易になります。

ログファイルのローテーションを、条件の組み合わせに基づいて行うこともできます。たとえば、ファイルサイズが 10M バイトより大きい場合に「のみ」、23 時 30 分にログをローテーションするように指定できます。

ログファイル作成ポリシーを設定する方法の詳細については、「Configuring Logs for Directory Server」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

ログファイル削除ポリシーの定義

ログファイル削除ポリシーを使用すると、保存された古いログを自動的に削除できます。ログファイル削除ポリシーは、Directory Service Control Center からか、またはコマンド行ユーティリティーを使用して Directory Server と Directory Proxy Server に対して定義できます。ログファイル削除ポリシーは、ログファイル作成ポリシーが定義されていないかぎり適用されません。ログファイルが 1 つしかない場合、ログファイル削除は機能しません。サーバーは、ログのローテーションの時点でログファイル削除ポリシーを評価および適用します。

ログファイル削除ポリシーを定義する場合は、次の点を考慮してください。

- 保存される合計のログの最大サイズ

  この最大サイズに達すると、保存されているもっとも古いログが自動的に削除されます。
- 使用可能にする最小のディスク空き容量

  ディスク空き容量がこの最小値に達すると、保存されているもっとも古いログが自動的に削除されます。
- ログファイルの最長有効期間

  ログファイルがこの最長有効期間に達すると、そのログファイルは自動的に削除されます。

ログファイル削除ポリシーを設定する方法の詳細については、「Configuring Logs for Directory Server」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

ログファイルの手動での作成と削除

手動のファイルローテーションや強制されたログローテーションは、Directory Proxy Server には適用されません。

Directory Server に対して自動的な作成および削除のポリシーを定義したくない場合は、ログファイルを手動で作成および削除することができます。さらに、Directory Server には、定義されている作成ポリシーには関係なく、任意のログをただちにロー

テーションできるタスクが用意されています。この機能は、たとえば、さらに詳細に検証する必要があるイベントが発生した場合に有効なことがあります。この即座のローテーション機能では、サーバーに新しいログファイルが作成されます。これにより、元のログファイルには、今後サーバーからログが追加されない状態となり、そのファイルを検証することができます。

ログを手動でローテーションする方法、およびログのローテーションを強制する方法については、「Rotating Directory Server Logs Manually」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

ログファイルのアクセス権の定義

5.x バージョンの Directory Server では、ログファイルを読み取れるのはディレクトリマネージャーだけでした。Directory Server Enterprise Edition では、ログファイルの作成時にアクセス権をサーバー管理者が定義できます。ログファイルのアクセス権を定義する方法については、「Configuring Logs for Directory Server」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

監視戦略の設計

配備を成功させるには、効果的な監視とイベント管理の戦略が必要です。この戦略は、監視対象となるイベント、使用するツール、そのイベントが発生した場合に実行するアクションを定義します。発生しがちなイベントに対する計画を立てておけば、サービスの停止やレベル低下の可能性を防止できます。この戦略によって、ディレクトリのサービスの可用性と品質が向上します。

監視戦略を設計するには、次のことを実行します。

- 適切な監視ツールを選択します。158 ページの「Directory Server Enterprise Edition で提供される監視ツール」を参照してください。
- ディレクトリアーキテクチャー内の、監視対象となる主要な領域を識別します。

  これらの領域は一般に、サイジングやチューニングの属性と同じです。159 ページの「監視領域の特定」を参照してください。
- パフォーマンス指標を監視する場合に、イベントまたはアラーム状態を始動する条件を定義します。

  この戦略では、パフォーマンスの受容可能レベル、または各パフォーマンス指標に対する処理を定義します。
- アラーム状態が発生したときに実行するアクションを決定します。

Directory Server Enterprise Edition で提供される監視ツール

ここでは、Directory Server Enterprise Edition で使用できる監視ツールや、サーバーアクティビティーの監視に使用できるその他のツールの概要について説明します。

159 ページの「監視領域の特定」で説明されている監視領域は、これらの 1 つ以上のツールを使用して監視できます。

- コマンド行ツール: ディスク使用量などのパフォーマンスを監視するオペレーティングシステムに固有のツール、ディレクトリに格納されているサーバー統計を収集する `ldapsearch` などの LDAP ツール、サードパーティー製ツール、カスタムシェル、Perl スクリプトが含まれます。
- **Directory Server** および **Directory Proxy Server** ログ: アクセス、監査、およびエラーログが含まれます。これらのログを手動で監視したり、カスタムスクリプトを使用して解析することで、配備に関連する監視情報を抽出できます。Directory Server Resource Kit には、アクセスログを解析できるログ分析ツール、`logconv` が用意されています。このログ分析ツールは、利用率統計を抽出し、重要なイベントの発生をカウントします。このツールの詳細については、`logconv`(1) を参照してください。ログファイルの表示と設定については、Chapter 15,「Directory Server Logging,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。
- **Directory Service Control Center (DSCC):** ディレクトリ操作をリアルタイムに監視できるグラフィカルユーザーインタフェースです。DSCC は、リソースの概要、現在のリソース使用状況、接続状態、グローバルデータベースキャッシュ情報など、一般的なサーバー情報を提供します。また、データベースのタイプや状態、エントリキャッシュ統計などの、一般的なデータベース情報も提供します。キャッシュ情報や、データベース内の各インデックスファイルに関連する情報も提供されます。さらに、DSCC には接続と各連鎖サフィックスで実行される操作に関する情報も表示されます。
- レプリケーション監視ツール: コマンド行ツール、`repldisc`、`insync`、および `entrycmp` が含まれます。

  これらのツールを使用すると、次の操作を実行できます。

  - マスターレプリカと 1 つまたは複数のコンシューマレプリカとの間の同期状態を監視する
  - 複数の異なるレプリカの間で同じエントリを比較することで、レプリケーションの状態を確認する
  - 完全なレプリケーショントポロジを描写する (これは複雑なディレクトリ配備で特に有用となる)

  詳細については、`repldisc`(1)、`insync`(1)、および `entrycmp`(1) を参照してください。

DCC を使用してレプリケーションの状態を監視することもできます。レプリケーションの監視の詳細については、「Getting Replication Status」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

- **SNMP (Simple Network Management Protocol):** グローバルネットワーク制御と監視のための標準メカニズムであり、ネットワーク管理者はネットワーク管理作業を一元的に行えます。

  SNMP エージェントを使用した監視については、Chapter 16,「Directory Server Monitoring,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

- **Java ES Monitoring Framework:** JMX を介した、パフォーマンスやその他の統計の監視を可能にします。詳細については、「Directory Server and CMM/JMX」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference』を参照してください。

監視領域の特定

監視する対象や、その対象をどの程度監視するかは、具体的な配備によって異なります。ただし、一般に、監視戦略には次の要素が含まれます。

- リソース使用状況、サーバーの状態、接続情報などの、サーバーアクティビティー
- キャッシュ、トランザクション、ロック、ログなどの情報を含む、データベースアクティビティー
- 使用可能なディスク容量やしきい値情報を含む、ディスクの状態
- レプリケーションが実行されているかどうかの状態や、同期の状態を含む、レプリケーションアクティビティー
- インデックスを使用しない検索、検索フィルタ、よく使用されるインデックスなど、インデックスの効率性に関する情報
- 失敗したバインド試行、開いている接続、実行権限を含む、セキュリティーの状態

Directory Editor を使用したデータ管理

Directory Server Enterprise Edition の Directory Editor コンポーネントは、Web ブラウザを使用してディレクトリデータを管理できる Java Web アプリケーションです。Directory Editor を使用すると、すべてのユーザーが、クライアントソフトウェアをインストールしなくてもディレクトリデータにリモートアクセスできます。

Directory Editor は、次の機能を提供します。

- 管理者やエンドユーザーがディレクトリユーザー、グループ、およびコンテナを作成したり編集したりできるようにする。
- アプリケーションサーバーや配備されているハードウェアに応じて、複数の並行ユーザーをサポートする。
- 大規模な企業ディレクトリのインストールをサポートする。
- インタフェースのカスタマイズ、ブランド化、および埋め込みを可能にする。

  カスタマイズは、Directory Server スキーマに動的に適応します。
- 直接のプログラミングによるのではなく、フォームの設定によるカスタマイズを可能にする。
- クライアントブラウザと Directory Server の間の SSL で暗号化された転送をサポートする。
- ロールに基づいて、メニューや機能へのアクセスを制限する。

  ロールはスキャンされ、グループ名とマッチングされます。ロールには特定の機能へのアクセス権が与えられます。機能とは、参照、設定、デバッグ、編集、作成、検索などの高レベルのアクションです。
- Directory Server 内の既存の ACI に基づいて、データへのアクセスを制限する。Directory Editor に固有の ACI を定義する必要はありません。
- 仮想リスト表示 (VLV) インデックスに基づいて、大量データをページ化して表示する。

Directory Editor のインストール、設定、および使用の詳細については、Directory Editor マニュアル コレクション (http://docs.sun.com/app/docs/coll/DirEdit_05q1)を参照してください。

パート III

論理設計

論理アーキテクチャーは、Directory Server Enterprise Edition 配備のコンポーネントを識別し、コンポーネント間の相互関係を示します。配備で必要なコンポーネントは通常、技術要件フェーズの間に作成されるユースケースによって示されます。ただし、必要なコンポーネントをビジネス要件から直接特定できる場合も数多くあります。

この第 3 部では、Directory Server Enterprise Edition の一般的な配備シナリオをベースにしたサンプルの論理アーキテクチャーを示します。この第 3 部では、基本的な単一サーバー配備から、複数のデータセンターにまたがる複雑な配備に至るまでの広範な情報を扱います。この部の各章で説明するアーキテクチャーは、以前の章で説明した単純なアーキテクチャーが基礎になっています。

次の章で構成されています。

- 第 9 章では、基本的な Directory Server Enterprise Edition 配備について説明します。
- 第 10 章では、追加のサービス要件を満たすための配備拡張について説明します。
- 第 11 章では、複数のデータセンターにまたがる配備についての考慮事項を扱います。
- 第 12 章では、可用性要件を満たすための配備設計について説明します。

◆ ◆ ◆

第 9 章

9 基本的な配備の設計

もっとも単純な Directory Server Enterprise Edition 配備では、1 つのデータセンター内にある、1 台のマシンにインストールされた単一の Directory Server でディレクトリサービスの要件を満たすことができます。組織が小規模な場合や、Directory Server をデモや評価のために実行している場合には、このシナリオが成り立つ可能性があります。前の章で説明されている技術的な要件は、すべての配備に等しく適用されることに注意してください。

この章では、1 台の Directory Server で構成される基本的な配備について説明します。この章の内容は次のとおりです。

基本的な配備のアーキテクチャー

基本的な Directory Server Enterprise Edition 配備には、次の要素が含まれます。

- Directory Server インスタンスファイル
- Directory Server デーモン
- `dsadm` および `dsconf` コマンド行ユーティリティー
- Directory Service Control Center (DSCC) (GUI アクセスが必要な場合)
- コンソールエージェント (DSCC が使用されている場合)

これらの要素をすべて、1 台のマシンにインストールできます。次の図は、基本的な Directory Server Enterprise Edition 配備の高レベルのアーキテクチャーを示しています。

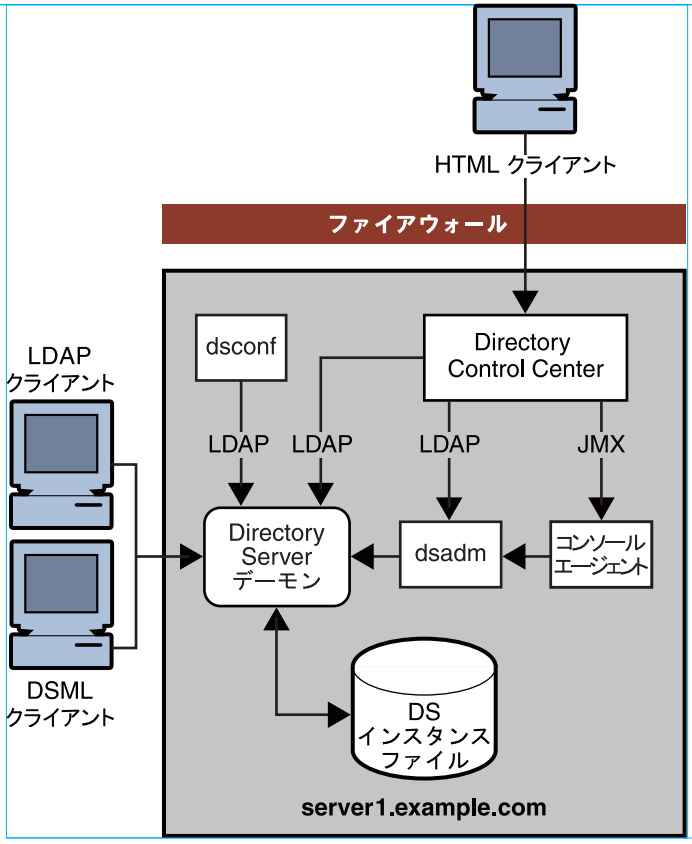


図 9–1　1 台のマシン上の基本的な Directory Server Enterprise Edition のアーキテクチャー

このシナリオでは、内部の LDAP および DSML クライアントを、Directory Server に直接アクセスするように設定できます。外部の HTML クライアントは、ファイアウォールを介して DSCC にアクセスするように設定できます。

先に説明したコンポーネントをすべて 1 台のマシンにインストールすることは可能ですが、それは実際の配備では現実的でありません。より標準的なシナリオでは、DSCC と `dsconf` コマンド行ユーティリティーを別のリモートマシンにインストールします。それによって、すべての Directory Server ホストを、これらのマシンからリモートに設定することも可能になります。次の図は、この、より標準的なシナリオを示しています。

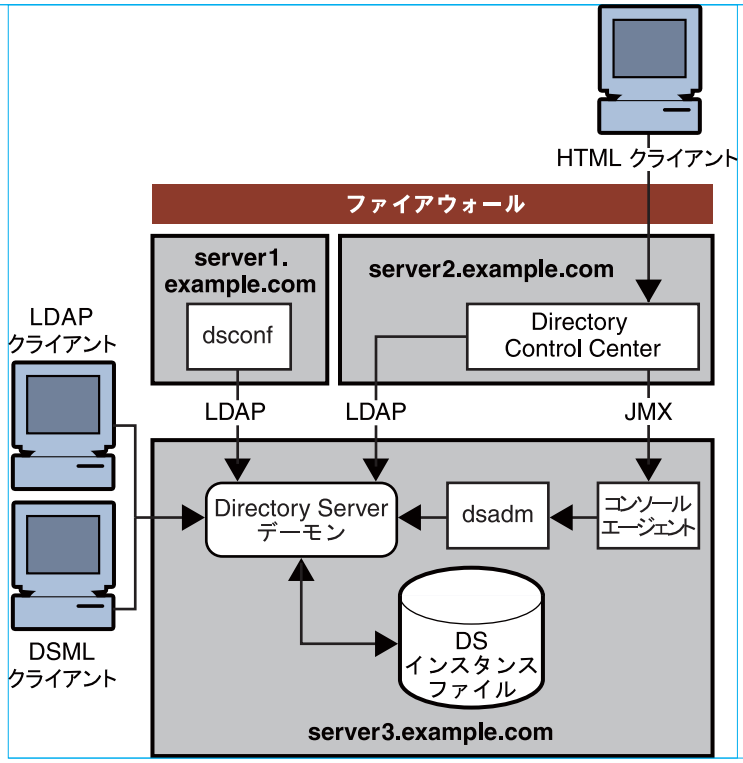


図 9–2　Directory Service Control Center がリモートにインストールされた基本的な Directory Server Enterprise Edition のアーキテクチャー

Directory Server インスタンスには、サーバーとアプリケーションの設定や、ユーザー情報が格納されます。一般に、サーバーとアプリケーションの設定情報は Directory Server の 1 つのサフィックスに格納され、ユーザーとグループのエントリは別のサフィックスに格納されます。サフィックスはディレクトリツリー内のエントリの名前で、データはその下に格納されます。

Directory Service Control Center (DSCC) は、すべてのサーバーを集中管理する Web ベースのユーザーインタフェースであり、また Java Web コンソールのディレクトリコンポーネントです。DSCC に登録されているすべてのサーバーとアプリケーションは、グラフィカルユーザーインタフェース上で表示され、そこでサーバーを管理したり設定したりできます。コマンド行インタフェースを通してすべての機能が提供されるため、小規模な配備では Directory Service Control Center が必要ない場合もあります。

以降の章では、Directory Service Control Center が別のマシンにインストールされていることを前提としています。この前提については、各章であらためて言及しません。

基本的な配備の設定

完全なインストール情報は、『Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide』で提供されています。この節の目的は、基本的な配備を構成する各要素を明確に示し、これらの要素が連携して動作するようすを説明することにあります。

この節では、前の節で説明した基本的な配備を設定するためのメインタスクを示します。

- 必要な共有コンポーネント (セキュリティーパッケージを含む) をインストールします。
- Directory Server、コンソールエージェント、およびコマンド行インタフェースをインストールします。
- コマンド行ユーティリティーを使用してサーバーを管理する場合は、次の手順を実行します
  - `dsadm` コマンドを使用して、スタンドアロンの Directory Server インスタンスを作成して起動します。
  - `dsconf` コマンドを使用して、新しいインスタンス内にサフィックスを作成して設定します。
- グラフィカルユーザーインタフェースを通してサーバーを管理する場合は、次の手順を実行します
  - Directory Service Control Center を初期化します。
  - Directory Service Control Center を使用して、Directory Server インスタンスを作成します。
  - Directory Service Control Center を使用して、新しいインスタンス内にサフィックスを作成して設定します。

基本的な配備でのパフォーマンスの向上

もっとも基本的な配備でも、特定の領域のパフォーマンスを向上させるために Directory Server の調整が必要になる場合があります。以降の節では、単純な単一サーバー配備に適用できる、基本的なチューニング戦略について説明します。これらの戦略は、トポロジ全体のパフォーマンスを向上させるために、より大規模で、より複雑な配備内の各サーバーにも適用できます。

検索を高速化するためのインデックスの使用

インデックスを使用すると、検索で一致するものを見つけるためにチェックする必要のあるエントリの数が実質的に削減されるため、検索が高速化されます。インデックスには、値のリストが含まれています。各値は、エントリ識別子のリストに

関連付けられています。Directory Server は、インデックス内のエントリ識別子のリストを使用して、各エントリをすばやく検索できます。Directory Server が、インデックスのない状態でエントリのリストを管理するには、検索で一致するものを見つけるためにサフィックス内のすべてのエントリをチェックする必要があります。

Directory Server は、各検索要求を次のように処理します。

1. Directory Server がクライアントから検索要求を受信します。
2. Directory Server は要求を検査して、その検索を処理できることを確認します。

   Directory Server が検索を実行できない場合は、クライアントにエラーを返します。また、その検索を Directory Server の別のインスタンスに任せる場合もあります。
3. Directory Server は、その検索に対応する 1 つ以上のインデックスを管理しているかどうかを判定します。
   - Directory Server がその検索に対応するインデックスを管理している場合、サーバーは、対応するすべてのインデックスを検索して候補エントリを見つけます。候補エントリとは、検索要求に一致する可能性のあるエントリです。
   - Directory Server がその検索に対応するインデックスを管理していない場合、サーバーは、データベース内のすべてのエントリをチェックすることによって候補エントリのセットを生成します。

     Directory Server がインデックスを使用できない場合は、この処理で消費される時間とシステムリソースが増えます。
4. Directory Server は各候補エントリを検査して、そのエントリが検索条件に一致するかどうかを判定します。
5. Directory Server は、一致するエントリを見つけると、そのエントリをクライアントアプリケーションに返します。

次のことを行うことによって、検索のパフォーマンスを最適化できます。

- インデックスが生成されていないエントリに対して Directory Server が検索を実行しないようにします。
- キャッシュサイズが適切に調整されていることを確認します。
- インデックスの長さを制限します。

インデックスの動作の包括的な概要については、Chapter 6, 「Directory Server Indexing,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。インデックスの定義については、Chapter 13, 「Directory Server Indexing,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

検索のパフォーマンス向上のためのキャッシュの最適化

検索のパフォーマンスを向上させるには、メモリー内にできるだけ多くのディレクトリデータをキャッシュします。ディレクトリサーバーがディスクから情報を読み取る回数を減らすことで、ディスクの I/O ボトルネックが軽減されます。これを実行するための方法は、ディレクトリツリーのサイズ、使用可能なメモリーの量、および使用されているハードウェアによって異なります。配備によっては、検索のパフォーマンスを最適化するために、エントリキャッシュやデータベースキャッシュに割り当てるメモリーを増やしたり減らしたりすることを選択する場合があります。あるいは、異なるサーバー上の Directory Server コンシューマに検索を分散させることを選択する場合もあります。

次のシナリオを検討してみます。

- 168 ページの「すべてのエントリとインデックスがメモリーに収まる」
- 169 ページの「32 ビット Directory Server のための十分なメモリーがある」
- 169 ページの「メモリー不足」

すべてのエントリとインデックスがメモリーに収まる

最適な場合は、データベースキャッシュとエントリキャッシュが使用可能な物理メモリーに収まります。エントリキャッシュは、ディレクトリ内のすべてのエントリを保持するための十分な大きさがあります。データベースキャッシュは、すべてのインデックスとエントリを保持するための十分な大きさがあります。この場合、検索対象はすべてキャッシュ内で見つかります。Directory Server は、エントリを取得するために、ファイルシステムキャッシュまたはディスクにアクセスする必要がまったくありません。

更新や拡張のあとも、データベースキャッシュにすべてのデータベースインデックスが含まれている必要があります。データベースキャッシュ内のインデックスの領域が不足すると、検索を行うたびに、Directory Server がディスクからインデックスを読み取らなければならなくなり、スループットが大幅に低下します。ページングやキャッシュのアクティビティーは、DSCC またはコマンド行を使用して監視できます。

適切なキャッシュサイズは、代表的なデータでの実験的なテストを通して決定してください。一般に、データベースキャッシュサイズは、(データベースファイルの合計サイズ) x 1.2 として計算できます。最初は、キャッシュに大量のメモリーを割り当ててください。次に、Directory Server を実行して監視し、その結果を観察します。この処理を、必要に応じて繰り返します。エントリキャッシュは特に、これらのキャッシュに割り当てた分よりはるかに多くのメモリーを使用する可能性があります。

32 ビット Directory Server のための十分なメモリーがある

エントリキャッシュとデータベースキャッシュ内のすべてのデータを保持するための十分なメモリーを備えているが、64 ビット Directory Server プロセスをサポートしていないシステムを考えてみます。ハードウェアの制約によって、64 ビットをサポートする Solaris システム上に Directory Server を配備できない場合は、32 ビットプロセスのメモリー制限を考慮してキャッシュサイズを適切に決定してください。次に、残りのメモリーをファイルシステムキャッシュに割り当てます。

パフォーマンスベンチマークの出発点として、エントリキャッシュのサイズを、できるだけ多数のエントリを保持するように決定します。データベースキャッシュのサイズは、完全に最小化せずに 100M バイト程度に比較的小さくしますが、ファイルシステムキャッシュにはデータベースページが保持されるようにします。

注 – ファイルシステムキャッシュは、システム上のほかのプロセス、特にファイルベースの操作と共有されます。そのため、特に Directory Server 専用のシステムでない場合、ファイルシステムキャッシュの制御はほかのキャッシュの制御より困難です。

システムがファイルシステムキャッシュをほかのプロセスに再割り当てする可能性があります。

インポートキャッシュは Directory Server プロセスに関連付けられているため、この状況でのオンラインインポートは避けてください。

メモリー不足

エントリキャッシュとデータベースキャッシュ内のすべてのデータを保持するにはメモリーが不足しているシステムを考えてみます。この場合は、エントリキャッシュとデータベースキャッシュの合計サイズが使用可能な物理メモリーを超えることのないようにしてください。もし超えてしまうと、仮想記憶のページングが大量に発生し、システムが事実上停止してしまう可能性があります。

小規模なシステムのベンチマークでは、最初、使用可能なメモリーをすべてエントリキャッシュとデータベースキャッシュに割り当てください (サイズはそれぞれ 100M バイト以上)。`mount_ufs` コマンドの `-o forcedirectio` オプションを使用して Solaris UFS ファイルシステムをマウントすることによって、ファイルシステムキャッシュの無効化を試みてください。詳細については、`mount_ufs(1M)` のマニュアルページを参照してください。ファイルシステムキャッシュを無効にすると、Directory Server に必要なメモリーがファイルシステムキャッシュでは使用されないようにすることができます。

大規模なマシン上で実行されている大規模な Directory Server では、ファイルシステムキャッシュを最大化し、データベースキャッシュを減らします。実験的なテストを通してこの前提を確認し、修正してください。

書き込みのパフォーマンス向上のためのキャッシュの最適化

配備に書き込みのスケーラビリティーを持たせるように最初から計画することに加えて、データベースキャッシュがメモリー内の更新を処理できるだけの十分なメモリーを用意してください。また、ディスクアクティビティーも最小限に抑えてください。データベースキャッシュの有効性は、Directory Service Control Center でヒット率を読み取ることによって監視できます。

Directory Server がしばらく実行されたあとは、キャッシュ内に、それ以上ディスク読み取りが必要ないほど十分なエントリとインデックスが含まれているようにしてください。更新の結果は、メモリー内の大規模なデータベースキャッシュにあるデータに反映され、データベースキャッシュはたまにしかフラッシュされないはずです。

チェックポイント中のディスクへのデータのフラッシュは、ボトルネックになる場合があります。データベースキャッシュサイズが大きければ大きいほど、このボトルネックは大きくなります。Sun StorEdge™ ディスクアレイなどの別の RAID システムにデータベースを格納すると、更新のパフォーマンス向上に役立つ場合があります。潜在的な I/O ボトルネックを分離するには、Solaris システムにおける `iostat` などのユーティリティーを使用できます。詳細については、`iostat(1M)` のマニュアルページを参照してください。

次の表は、2、3、および 4 つのディスクを備えたシステムでのデータベースとログの配置方法に関する推奨事項を示しています。

表 9–1 別のディスクへのデータベースとログの分離

使用可能なディスクの数	推奨事項
2	■ Directory Server データベースを 1 番目のディスクに配置します。 ■ トランザクションログ、アクセスログ、監査ログ、エラーログ、および旧バージョン対応更新履歴ログを 2 番目のディスクに配置します。
3	■ Directory Server データベースを 1 番目のディスクに配置します。 ■ トランザクションログを 2 番目のディスクに配置します。 ■ アクセスログ、監査ログ、エラーログ、および旧バージョン対応更新履歴ログを 3 番目のディスクに配置します。

表 9–1 別のディスクへのデータベースとログの分離 (続き)

使用可能なディスクの数	推奨事項
4	■ Directory Server データベースを 1 番目のディスクに配置します。 ■ トランザクションログを 2 番目のディスクに配置します。 ■ アクセスログ、監査ログ、エラーログを 3 番目のディスクに配置します。 ■ 旧バージョン対応更新履歴ログを 4 番目のディスクに配置します。

◆ ◆ ◆ 第 10 章

10 拡張配備の設計

第 9 章で説明した基本的な配備では、1 つの Directory Server で、組織の読み取り要件および書き込み要件を十分に満たせることを前提としています。読み取りまたは書き込み要件がそれよりも厳しい (多数のクライアントがディレクトリデータに同時アクセスを試みる) 組織では、拡張配備を使用する必要があります。

一般に、すべてのデータをキャッシュするための十分なメモリーが搭載されているという仮定のもとで、Directory Server インスタンスが 1 秒あたりに実行できる検索の件数は、サーバーの CPU の数および速度と直接関係します。水平的な読み取りの拡張容易性は、2 台以上のサーバーに負荷を分散させることによって達成できます。これは通常、データの追加コピーを用意して、クライアントが複数のソースからデータを読み取れるようにすることを意味します。

書き込み操作は水平的には拡張しません。それは、マスターサーバーへの書き込み操作の結果として、すべてのレプリカへの書き込み操作が発生するためです。書き込み操作を水平的に拡張する唯一の方法は、ディレクトリデータを複数のデータベースに分割し、それらのデータベースを複数の異なるサーバーに配置することです。

この章では、読み取りおよび書き込みの処理可能件数を増やすために Directory Server Enterprise Edition 配備を拡張する各種の方法について説明します。この章の内容は次のとおりです。

読み取りの拡張容易性のための負荷分散の使用

負荷分散は、読み取り負荷を複数のサーバーに拡散させることによってパフォーマンスを高めます。負荷分散は、レプリケーション、Directory Proxy Server、またはこの両者の組み合わせを使用して実現できます。

負荷分散のためのレプリケーションの使用

レプリケーションは、ディレクトリデータをコピーし、あるディレクトリサーバーから別のディレクトリサーバーに切り替える処理を自動的に行う機構です。レプリケーションを使用することで、ディレクトリツリー、またはそのツリーに固有のサフィックスに格納されるサブツリーをサーバー間でコピーできます。

注 – 設定または監視情報のサブツリーはコピーできません。

ディレクトリデータをサーバー間でレプリケートすることにより、1 台のマシンにかかるアクセス負荷を軽減して、サーバーの応答時間を短縮し、読み取りの拡張容易性を提供することができます。ディレクトリのエントリをユーザーに近い場所にレプリケートすることで、ディレクトリの応答時間を短縮することもできます。レプリケーションは一般的に、書き込みの拡張容易性のためのソリューションではありません。

レプリケーションの基本概念

レプリケーション機構については、Chapter 4,「Directory Server Replication,」 in『Sun Java System Directory Server Enterprise Edition 6.3 Reference』で詳しく説明しています。次の節では、この章の後半で説明するサンプルトポロジを検討する前に理解しておく必要がある基本的な情報を示します。

マスターレプリカ、コンシューマレプリカ、ハブレプリカ

レプリケーションに関与するデータベースを「レプリカ」と呼びます。

Directory Server では、3 種類のレプリカを区別しています。

- マスターレプリカ (読み書き可能レプリカ): ディレクトリデータのマスターコピーを含む読み書き可能データベース。マスターレプリカは、ディレクトリクライアントからの更新要求を処理できます。複数のマスターを含むトポロジのことを「マルチマスター」トポロジと呼びます。
- コンシューマレプリカ: マスターレプリカ内の情報のコピーを保持する読み取り専用データベース。コンシューマレプリカは、ディレクトリクライアントからの検索要求を処理できますが、更新要求はマスターレプリカを照会します。

- ハブレプリカ: コンシューマレプリカと同様の読み取り専用データベース。1 つ以上のコンシューマレプリカを「供給」する Directory Server 上に格納されます。

次の図は、レプリケーショントポロジ内でのこれらの各レプリカのロールを示したものです。

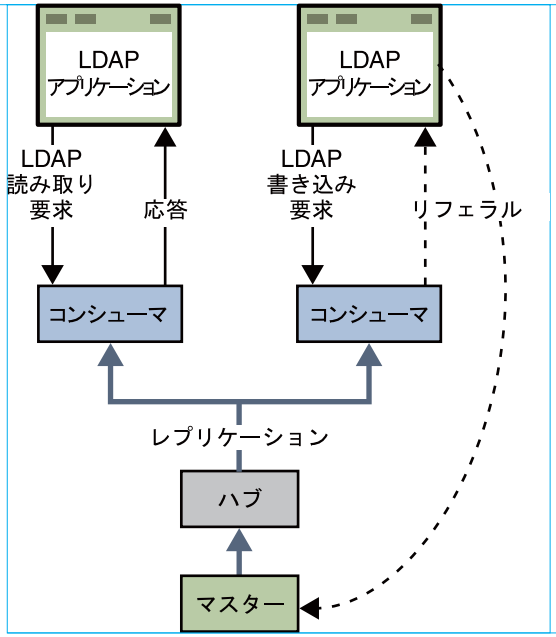


図 10–1 レプリケーショントポロジ内でのレプリカのロール

注 – この図は例示のみを目的としたものであり、必ずしも推奨されるトポロジではありません。マルチマスタートポロジで、Directory Server 6.x がサポートするマスター数は無制限です。ほとんどの場合、マスターのみのトポロジが推奨されます。

サプライヤとコンシューマ

ほかのサーバーにレプリケートする Directory Server のことを「サプライヤ」と呼びます。また、ほかのサーバーによって更新される Directory Server のことを「コンシューマ」と呼びます。サプライヤは、特別に設計された LDAP v3 拡張処理により、コンシューマ上のすべての更新を再現します。このためサプライヤは、パフォーマンスの面ではコンシューマに対する要件の多いクライアントアプリケーションに似ています。

次のような状況で、サーバーはサプライヤとコンシューマの両方になることができます。

- 2 つの異なる Directory Server 上にそれぞれマスターレプリカが配置されているマルチマスターレプリケーション環境では、一方のサーバーは、他方のサーバーのサプライヤとして、またコンシューマとしての役割を果たします。
- サーバーにハブレプリカが含まれるとき、サーバーはサプライヤから更新を受け取り、変更内容をコンシューマにレプリケートします。

コンシューマのロールのみを果たすサーバーのことを「専用コンシューマ」と呼びます。

「マスターレプリカ」の役割をするサーバーは次のことを行う必要があります。

- ディレクトリクライアントからの更新要求に応答する
- 履歴情報および変更履歴ログを維持する
- コンシューマへのレプリケーションを開始する

マスターレプリカを含むサーバーは、マスターレプリカに対して行われたすべての変更を記録する処理と、これらの変更をコンシューマにレプリケートする処理を受け持ちます。

「ハブレプリカ」の役割をするサーバーは次のことを行う必要があります。

- 読み取り要求に応答する
- マスターレプリカを保持するサーバーに更新要求を送信する
- 履歴情報および変更履歴ログを維持する
- コンシューマへのレプリケーションを開始する

「コンシューマレプリカ」の役割をするサーバーは次のことを行う必要があります。

- 読み取り要求に応答する
- 履歴情報を維持する
- マスターレプリカを保持するサーバーに更新要求を送信する

マルチマスターレプリケーション

マルチマスターレプリケーション設定では、データは異なる場所で同時に更新することができます。各マスターは、そのレプリカの変更履歴ログを維持します。各マスター上で発生した変更は、ほかのサーバーにレプリケートされます。

マルチマスター設定には、次の利点があります。

- 1 つのマスターにアクセスできなくなった場合でも、自動的に書き込み処理のフェイルオーバーが発生します。
- 地理的に分散されている環境では、ローカルのマスターで更新処理を実行できます。

マルチマスターレプリケーションでは、疎整合レプリケーションモデルを使用します。つまり、同一のエントリを別々のサーバーから同時に変更できます。2つのサーバー間で更新が送信された場合、更新の競合を解決する必要があります。待ち時間などのWANの各種特性により、レプリケーション競合の可能性が高まる可能性があります。競合の解決は通常、自動的に行われます。どの変更が優先されるかは、競合規則によって決定されます。競合を手動で解決しなければならない場合もあります。詳細は、「Solving Common Replication Conflicts」 in『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide』を参照してください。

マルチマスタートポロジでサポートされるマスターの数は、理論上は無制限です。同様に、コンシューマとハブの数も理論上は無制限です。ただし、1つのサプライヤから何個のコンシューマにレプリケーションを実行できるかは、サプライヤサーバーの能力に依存します。サプライヤサーバーの能力を評価するために、SLAMD分散負荷生成エンジン(SLAMD)を使用できます。SLAMDの詳細およびSLAMDソフトウェアのダウンロードについては、http://www.slamd.com (`http://www.slamd.com/`) を参照してください。

レプリケーションの単位

レプリケーションの最小単位はサフィックスです。レプリケーション機構では、サフィックスとデータベースが1対1で対応している必要があります。カスタム分散ロジックを使用している2つ以上のデータベースにまたがって分散されているサフィックス(またはネームスペース)はレプリケートできません。レプリケーション単位は、コンシューマとサプライヤの両方に適用されます。つまり、1つのサフィックスだけを保持するコンシューマに2つのサフィックスをレプリケートすることはできません。

更新履歴ログ

サプライヤとして機能するすべてのサーバーは更新履歴ログを維持します。「更新履歴ログ」は、マスターレプリカに対して行われた変更を記述した記録です。サプライヤは、この変更をコンシューマ上で再現します。エントリの変更、名称変更、追加、または削除が行われると、LDAP操作を記述する変更レコードが更新履歴ログに記録されます。

レプリケーションアグリーメント

Directory Serverでは、レプリケーションアグリーメントを使用して、2つのサーバー間でレプリケーションがどのように行われるかを定義します。「レプリケーションアグリーメント」は、1つのサプライヤと1つのコンシューマの間のレプリケーションを定義します。

レプリケーションアグリーメントは次のものを識別します。

- レプリケートするサフィックス
- データがレプリケートされるコンシューマサーバー
- レプリケーションを実行できる時間帯
- サプライヤがコンシューマへのバインドに使用する必要があるバインド DN と資格
- 接続をセキュリティー保護する方法 (SSL やクライアント認証など)
- この特定のアグリーメントのレプリケーションの状態に関する情報
- レプリケーションフィルタについての情報

複製の優先順位

Directory Server 6.x よりも前のバージョンの Directory Server では、更新は時系列順にレプリケートされていました。本バージョンでは、更新のレプリケーションに優先順位を付けることができます。優先度はブール型の機能であり、オンまたはオフを選択できます。優先度のレベルはありません。レプリケーションを待機している更新のキュー内で、優先度がオンの更新は、優先度がオフの更新よりも先にレプリケートされます。レプリケーションを待機している更新のキュー内で、優先度がオンの更新は、優先度がオフの更新よりも先にレプリケートされます。

優先度規則は、次のパラメータに従って設定されます。

- クライアントのアイデンティティー
- 更新のタイプ
- 更新されたエントリまたはサブツリー
- 更新によって変更される属性

詳細については、「Prioritized Replication」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

初期レプリケーション要件の評価

ディレクトリサービスのレプリケーションを成功させるには、本稼働環境での徹底的なテストおよび分析が必要です。ただし、次の基本的な計算を使用して、レプリケートされるトポロジの設計を開始することができます。以降の節では、レプリケートされるトポロジ設計の基礎として、この計算の結果を使用します。

▼ 初期レプリケーション要件を決定するには

1 使用状況がピークに達する時間帯で、1 秒あたり最大何件の検索を処理できる必要があるかを見積もります。

この見積もり値を「総検索数」とします。

2. 単一のホストで処理できる1秒あたりの検索数をテストします。

   この見積もり値を「ホストあたりの検索数」とします。このテストは「レプリケーションを有効にした」状態で行う必要があります。

   ホストが処理できる検索数は、さまざまな変動要因の影響を受けます。特に影響が大きいのは、エントリのサイズ、ホストの容量、およびネットワークの速度です。これらのテスト作業に役立つパフォーマンステストツールが、サードパーティーから多数提供されています。SLAMD 分散負荷生成エンジン (SLAMD) は、ネットワークベースアプリケーションの負荷テストおよびパフォーマンス分析の用途向けに設計された、オープンソースの Java アプリケーションです。SLAMD を利用することで、レプリケーション評価のこの段階の作業を効率的に実行できます。SLAMD の詳細および SLAMD ソフトウェアのダウンロードについては、http://www.slamd.com (http://www.slamd.com/) を参照してください。

3. 必要なホスト数を計算します。

   ホスト数 = 総検索数/ホストあたりの検索数

単一データセンターでのマルチマスターレプリケーションを利用した負荷分散

レプリケーションにより、Directory Server への負荷を次のように分散させることができます。

- 検索アクティビティーを複数のサーバーに分散する
- 特定のサーバーを特定のタスクまたはアプリケーションの処理に専念させる

一般に、完全接続型トポロジにおいて、178 ページの「初期レプリケーション要件の評価」で計算した「ホスト数」が 16 前後であるか、それよりも極端に大きくなければ、トポロジにはマスターサーバーのみを含めることをお勧めします。完全接続型とは、すべてのマスターが、トポロジ内のほかのマスターすべてにレプリケートするという意味です。

注 – 計算で得られた「ホスト数」は概算値であり、ハードウェア構成や、配備のその他の詳細条件によって変動します。

次の図では、「ホスト数」が 2 であると仮定しています。クライアントアプリケーションのタイプに基づいて、LDAP 操作が 2 つのマスターサーバーに分割されます。この方針により、各サーバーにかかる負荷が減少し、配備全体で処理できる合計の操作数は増加します。

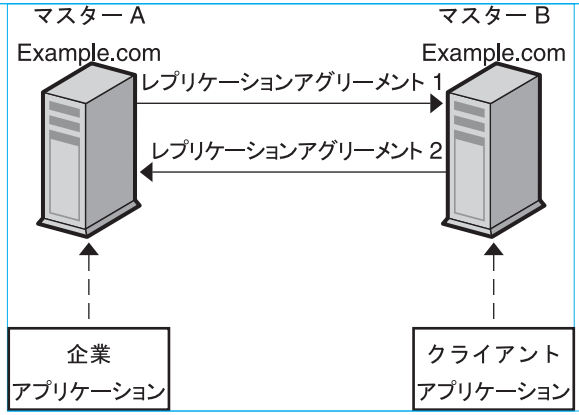


図 10–2 負荷分散のためのマルチマスターレプリケーションの使用

グローバル配備における類似のシナリオについては、195 ページの「WAN を介したマルチマスターレプリケーションの使用」を参照してください。

大規模配備でのレプリケーションによる負荷分散

対象の配備で必要な「ホスト数」が 16 を大きく超える場合、専用コンシューマをトポロジに追加しなければならない場合があります。

次の図では「ホスト数」が 24 であると仮定し、簡略化のためにトポロジの一部のみを示しています。残り 10 台のサーバーも同一の構成であり、合計で 8 個のマスターと 16 個のコンシューマが存在します。

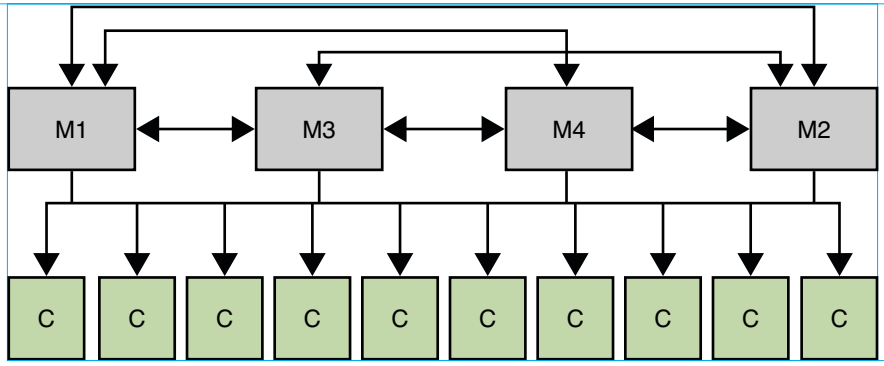


図 10–3 大規模配備でのマルチマスターレプリケーションを使用した負荷分散

次のことを行う必要がある場合、該当する任意のコンシューマに対して更新履歴ログを有効にできます。

- 機能停止の発生時にコンシューマをマスターにプロモートする
- マスターからどれか 1 つのコンシューマへのバイナリ初期化を実行する

「ホスト数」が数百に達する場合は、トポロジにハブを追加することを検討してください。そのような場合、ハブの数はマスターの数よりも多くしてください。具体的には、マスター 1 個あたり最大で 10 個のハブを使用し、各ハブでは 20 個程度までのコンシューマへのレプリケーションを処理するようにします。

どのようなトポロジでも、ハブ数とマスター数、またはハブ数とコンシューマ数が同じになるような構成は避けてください。

マルチマスタートポロジを簡素化するためのサーバーグループの使用

「ホスト数」が大きい場合、「サーバーグループ」を使用してトポロジを簡素化し、リソースの使用効率を改善することができます。16 個のマスターが存在するトポロジで、4 つのサーバーグループを使用してそれぞれに 4 つのマスターを配置すると、16 個のマスターを完全メッシュ構成にする場合よりも管理が容易になります。

そのようなトポロジを設定するために必要な手順は、次のとおりです。

- 16 個のマスターを設定します。この時点ではレプリケーションアグリーメントは設定しません。
- 4 つのサーバーグループを作成し、各グループに 4 つのマスターを含めます。
- 単一グループ内のすべてのマスター間でレプリケーションアグリーメントを設定します。
- 各グループの最初のマスター間、各グループの 2 番目のマスター間、というように順次レプリケーションアグリーメントを設定します。

次の図に最終的なトポロジを示します。

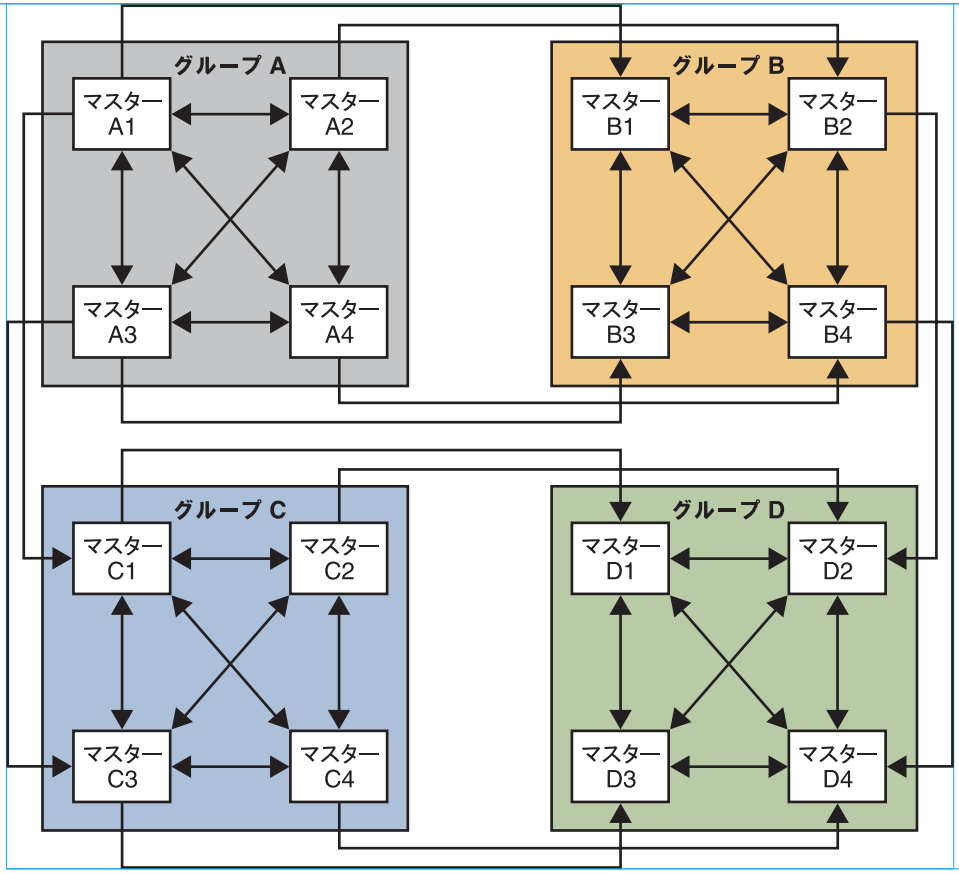


図 10–4　マルチマスタートポロジでのサーバーグループ

Directory Proxy Server を使用した負荷分散

Directory Proxy Server では、複数のサーバーを使用することにより、単一データソースの負荷を分散させることができます。また Directory Proxy Server は、サーバーのうち 1 台が停止した場合でもデータの継続的な可用性を保証できます。Directory Proxy Server はデータの分散だけでなく、操作のタイプに基づいた負荷分散機能も提供します。この機能では、サーバーは操作の「タイプ」に基づいて、クライアント操作を特定の Directory Server に経路指定できます。

Directory Proxy Server は操作のタイプに基づいた負荷分散に加えて、作業負荷が Directory Server 間でどのように共有されるかを決定する各種の負荷分散アルゴリズムをサポートします。これらの各アルゴリズムの詳細は、Chapter 16,「Directory Proxy Server Load Balancing and Client Affinity,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

次の図は、比例アルゴリズムを使用して 2 つのサーバー間で読み取り負荷を分散させる方法を例示しています。操作のタイプに基づいた負荷分散では、マスター 1 のサーバーで障害が発生しないかぎり、すべての書き込みがマスター 1 に経路指定されます。マスター 1 のサーバーで障害が発生すると、すべての読み取りと書き込みはマスター 2 に経路指定されます。

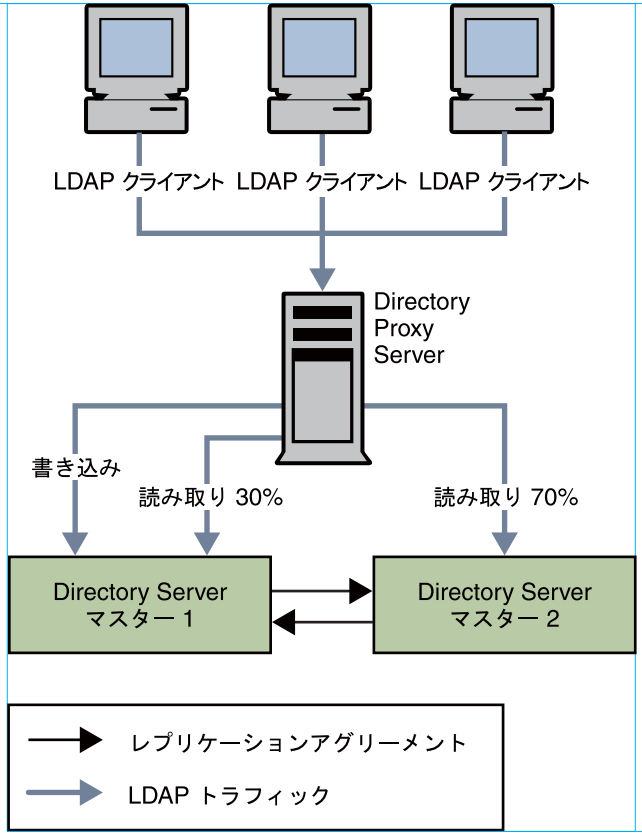


図 10–5　拡張配備での比例および操作ベース負荷分散の使用

1 つのサーバーインスタンスで障害が発生しても、負荷分散の設定は再計算されません。比例型の負荷分散を使用し、サーバーの負荷分散ウェイトを 0 に設定しても、「ホットスタンバイ」サーバーを作成することはできません。

たとえば、3 つのサーバー A、B、および C が存在するとします。また、サーバー A と B がそれぞれ 50% の負荷を受け持つように比例型負荷分散が設定されています。サーバー C は、スタンバイ専用のサーバーとして、0% の負荷を受け持つように設定されています。サーバー A で障害が発生すると、負荷の 100% が自動的にサーバー B に振り分けられます。サーバー B でも障害が発生した場合にはじめて、サーバー C に負荷が分散されます。そのため、負荷分散に常時参加しているインスタンスが、

常に負荷を受け持ち、それらのインスタンスすべてで障害が発生した場合にかぎり、スタンバイ状態のサーバーに負荷が振り分けられることになります。

飽和負荷分散アルゴリズムを使用し、スタンバイサーバーに低いウェイトを適用することにより、ホットスタンバイと同様の効果を達成できます。そのようなサーバーは本来の意味のスタンバイサーバーではありませんが、メインのサーバーの負荷が極端に高まった場合にのみこのサーバーに要求が分散されるように、アルゴリズムを設定することができます。メインサーバーの 1 つが無効になり、それに伴ってほかのメインサーバーの受け持つ負荷がかなり上昇すると、スタンバイサーバーに対して負荷の分散が行われます。

書き込みの拡張容易性のための分散の使用

書き込み操作はリソースの消費が大きい操作です。クライアントが書き込み操作を要求すると、データベース上で次の一連のイベントが発生します。

- バックエンドデータベースがロックされる
- データベースキャッシュ内でエントリがロックされる
- アクセス制御チェックプラグインが呼び出される
- バックエンド操作前プラグインが呼び出される
- データベーストランザクションが開始される
- データベースファイルが更新される
- 古いエントリキャッシュが新しいデータに置き換えられる
- データベーストランザクションがコミットされる
- バックエンド操作後プラグインが呼び出される
- バックエンドデータベースがロック解除される

このような複雑な手順により書き込み操作が行われるため、書き込み数が増加するとパフォーマンスも著しく影響を受けます。

企業の規模が拡大すると、より多くのクライアントアプリケーションがディレクトリへの高速な書き込みアクセスを要求するようになります。また、単一の Directory Server に格納される情報が増加すればするほど、ディレクトリデータベースのエントリを追加または変更するためのコストも上昇します。これは、インデックスが肥大化して、インデックスに含まれる情報の操作にかかる時間が長くなることが原因です。

サービスレベル契約を達成するために、すべてのデータをメモリー上にキャッシュとして保持しなければならない場合もあります。ただしその場合、データが大きすぎて 1 台のマシンのメモリーに収まりきれないことも考えられます。

ディレクトリデータの分量がこのレベルにまで増加した時点で、複数のサーバーに格納できるようにデータを分割する必要が生じます。1 つのアプローチは、階層を使用して情報を分割するというものです。何らかの基準に従って情報を複数の分岐に

分離することにより、各分岐を別個のサーバーに格納できます。その後、連鎖またはリフェラルを使用して各サーバーを設定し、クライアントが単一点からすべての情報にアクセスできるようにします。

この種の分割では、各サーバーはディレクトリツリーの一部分にのみ責任を持ちます。分散ディレクトリサービスは、ドメインネームサービス (DNS) に似た方法で機能します。DNS では、DNS ネームスペースの個々の部分を特定の DNS サーバーに割り当てます。同様に、ディレクトリのネームスペースを複数のサーバーに分散し、クライアント側からは単一のディレクトリツリーとして見えるように管理することができます。

階層ベースの分散機構には、いくつかの短所があります。主な問題は、この機構では、情報が存在する場所をクライアントの側で正確に把握している必要があるということです。そうでない場合、クライアントは広範な検索を実行してデータを見つけ出す必要があります。もう 1 つの問題は、一部のディレクトリ対応アプリケーションで、複数の分岐に分割された情報を適切に処理できないというものです。

Directory Server では、連鎖機構およびリフェラル機構との組み合わせで階層ベースの分散をサポートします。ただし、分散機能はスマートルーティングをサポートする Directory Proxy Server でも提供されます。この機能により、企業ごとに最適な分散機構を決定することができます。

複数のデータベースの使用

Directory Server では、パフォーマンスの高いディスクベースの LDBM データベースにデータを格納します。各データベースは 1 つのファイルセットで構成され、ファイルセットには、このセットに割り当てられたすべてのデータが含まれます。ディレクトリツリーの異なる部分を別々のデータベースに格納できます。たとえば、次の図に示すように、ディレクトリツリーに 3 つのサブサフィックスが含まれていると仮定します。

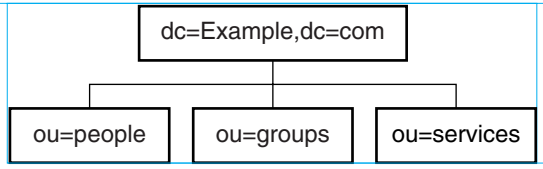


図 10–6　3 つのサブサフィックスを持つディレクトリツリー

次の図に示すように、3 つのサブサフィックスのデータを 3 つの独立したデータベースに格納できます。

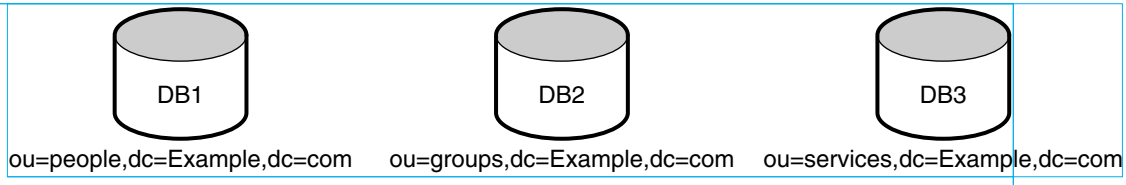


図 10–7　3 つの異なるデータベースに格納された 3 つのサブサフィックス

複数のデータベースにディレクトリツリーを分割するとき、複数のサーバー間にデータベースを分散させることができます。またパフォーマンス改善のために、複数の物理マシンに分散することができます。前の例で示した 3 つのデータベースを、次の図に示すように 2 つのサーバーに格納することができます。

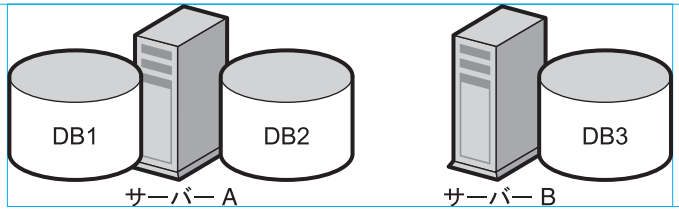


図 10–8　2 つの独立したサーバーに格納される 3 つのデータベース

データベースを複数のサーバーに分散させると、各サーバーで処理しなければならない作業量は減ります。そのため、ディレクトリを拡張して、1 つのサーバーで保持できる数よりはるかに多くのエントリを保持できるようにすることができます。Directory Server はデータベースの動的追加をサポートするため、ディレクトリ全体を停止させることなく、必要に応じて新しいデータベースを追加できます。

Directory Proxy Server を使用した分散

Directory Proxy Server はディレクトリ情報を複数のサーバーに分割しますが、その際にデータの階層を変更する必要はありません。データ分散にとって重要なことは、データセットを論理的な方法で分割することです。ただし、あるクライアントアプリケーションに対して適切に機能する分散ロジックが、別のクライアントアプリケーションに対しても同じように機能するとはかぎりません。

この理由から、Directory Proxy Server では、データがどのように分散されるか、またディレクトリ要求がどのように経路指定されるべきかをユーザーが指定できます。たとえば、ディレクトリ情報ツリー (DIT) の階層に基づいて、LDAP 操作を複数の異なるディレクトリサーバーに経路指定することができます。操作タイプやカスタムの分散アルゴリズムに基づいて操作を経路指定することもできます。

Directory Proxy Server は、分散の詳細をクライアントアプリケーションから実質的に隠蔽します。クライアントの側からは、発行したディレクトリクエリーは 1 つの

ディレクトリによって処理されるように見えます。クライアント要求は、特定の分散方法に従って分散されます。以降の節で説明するように、DIT の部分ごとに異なるルーティング方針を各部分に関連付けることができます。

DIT に基づくルーティング

この方針は、DIT の構造に基づいてディレクトリエントリを分散させるために使用できます。たとえば、サブツリー「`o=sales,dc=example,dc=com`」内のエントリを、Directory Server A に、サブツリー「`o=hr,dc=example,dc=com`」内のエントリを Directory Server B にそれぞれ経路指定できます。

カスタムアルゴリズムに基づくルーティング

DIT に基づくルーティング以外の方法でエントリを複数のディレクトリサーバーに分散させたい場合もあるでしょう。たとえば、あるサービスプロバイダで、登録者を表すエントリを「`ou=subscribers,dc=example,dc=com`」内に格納するとします。登録者の数が増えるにつれ、登録者 ID の範囲に基づいて登録者を複数のサーバーに分散させることが必要になる場合があります。カスタムのルーティングアルゴリズムを使用して、ID が 1〜10000 の範囲の登録者エントリを Directory Server A に、ID が 10001 以降の範囲の登録者エントリを Directory Server B に配置できます。サーバー B 上のデータが増えすぎた場合、分散アルゴリズムを変更して、ID が 20001 以降のエントリを新しいサーバー C に配置することができます。

Directory Proxy Server の `DistributionAlgorithm` インタフェースを使用し、独自のルーティングアルゴリズムを実装することができます。

Directory Proxy Server を使用したバインド DN ベースの要求分散

このシナリオでは、企業は地理的な位置を基準にして、3 つのマスターサーバーに顧客データを分散させます。イギリスに在住する顧客のデータを、ロンドンにあるマスターサーバーに格納します。フランスの顧客のデータはパリにあるマスターサーバーに格納します。日本の顧客のデータは東京にあるマスターサーバーに格納します。顧客は自分自身のデータを、Web ベースの統一されたインタフェースを通して更新できます。

ユーザーはディレクトリ内の自分自身の情報を、Web ベースのアプリケーションを使用して更新できます。認証フェーズの間、ユーザーは電子メールアドレスを入力します。イギリスの顧客の電子メールアドレスは `*@uk.example.com` という形式です。フランスの顧客の形式は `*@fr.example.com` であり、日本の顧客は `*@ja.example.com` です。Directory Proxy Server は、LDAP 対応のクライアントアプリケーション経由でこれらの要求を受信します。Directory Proxy Server はその後、認証時に入力された電子メールアドレスに基づいて、適切なマスターサーバーに要求を経路指定します。

次の図はこのシナリオを例示したものです。

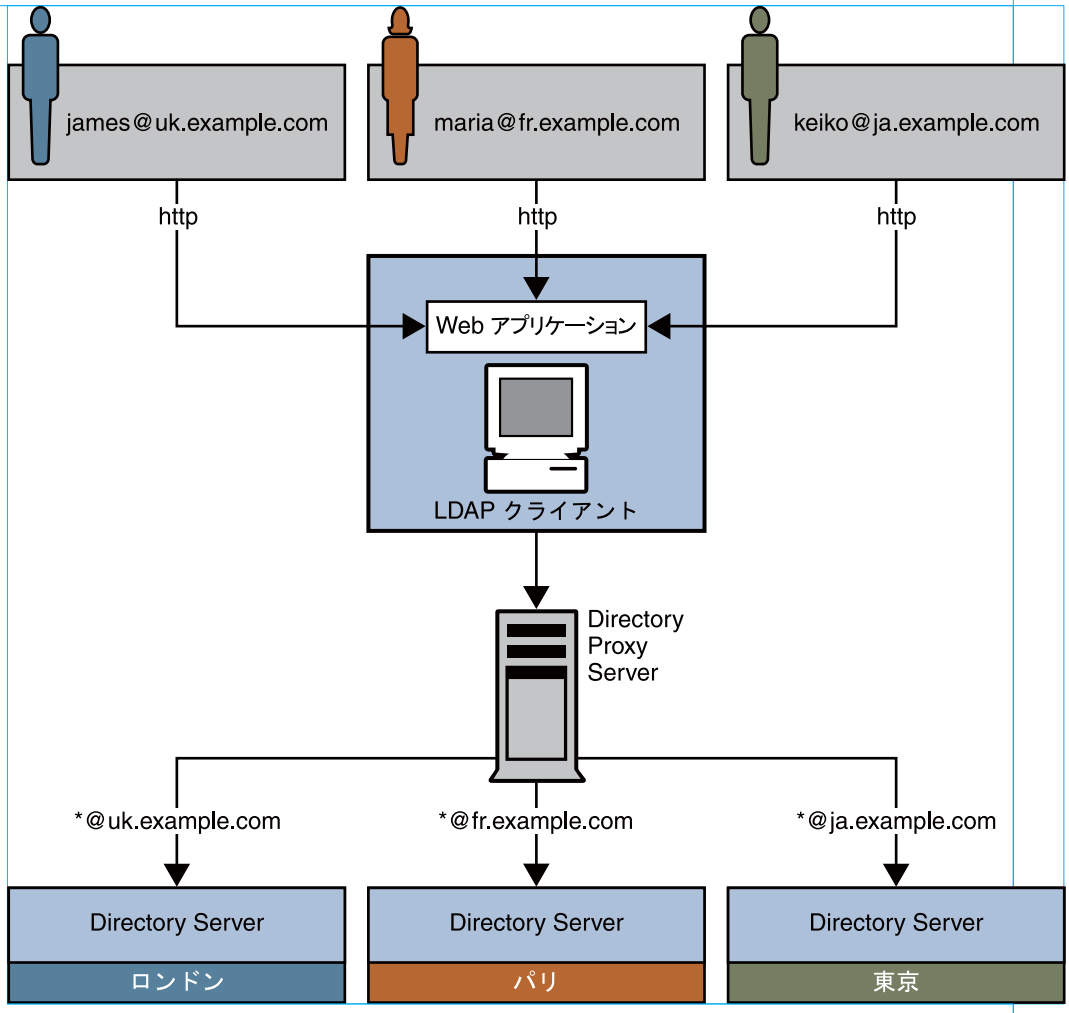


図 10–9　Directory Proxy Server を使用したバインド DN ベースの要求ルーティング

DIT の下位方向へのデータ分散

多くの場合、DIT の頂点でのデータ分散は必要ありません。ただし、ツリーの上位にあるエントリが、ツリーの分散済み部分にあるエントリによって必要とされる場合があります。この節では、サンプルのシナリオを通して、このようなケースでの分散方針を設計する方法を示します。

分散されるデータの論理ビュー

Example.com には、グループに対応する 1 つのサブツリーと、人に対応する 1 つの独立したサブツリーがあります。グループ定義の数は少なくあまり変化しませんが、人を表すエントリの数は多く、増加し続けます。そのため Example.com では、人のエントリだけを 3 つのサーバーに分散させる必要があります。ただし、people サブツリー下のすべてのエントリにアクセスするには、グループ定義、グループ定義の ACI、およびネーミングコンテキストの最上位に位置する ACI が必要です。

次の図は、データ分散要件の論理ビューを示したものです

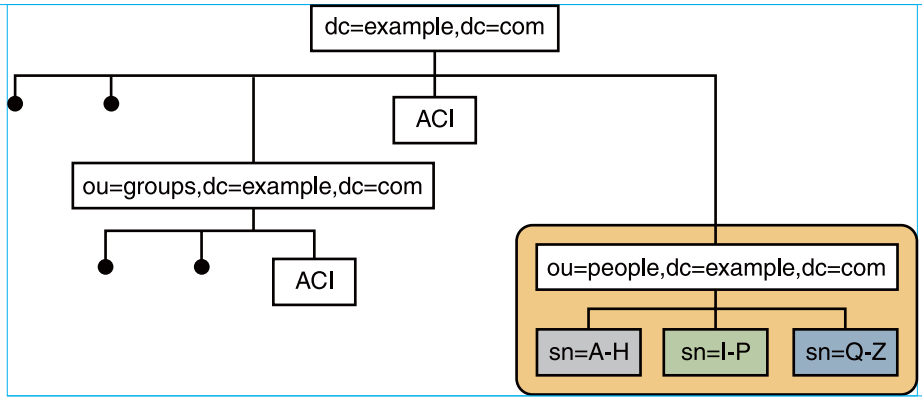


図 10–10 分散されるデータの論理ビュー

データ記憶領域の物理ビュー

`ou=people` サブツリーは、各エントリの `sn` 属性の先頭文字に従って、3 つのサーバーに分散されます。ネーミングコンテキスト (`dc=example,dc=com`) および `ou=groups` コンテナは、各サーバー上の 1 つのデータベースに格納されます。`ou=people` 下のエントリがこのデータベースにアクセスできます。`ou=people` コンテナは、このコンテナ専用のデータベースに格納されます。

次の図は、個々の Directory Server 上にデータがどのように格納されるかを示したものです。

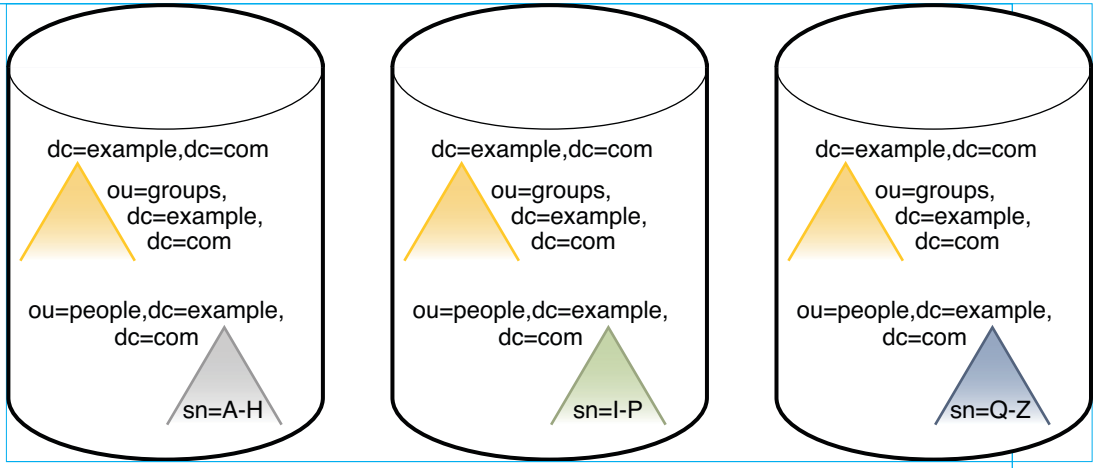


図 10–11 データ記憶領域の物理ビュー

`ou=people` コンテナはトップコンテナのサブサフィックスではありません。

サンプル配備シナリオ用の Directory Server 設定

これまでに説明した各サーバーは、分散チャンク (chunk) として理解することができます。ネーミングコンテキストおよび `ou=groups` 下のエントリを含むサフィックスは、各チャンク上で同じです。したがって、3 つのチャンクのそれぞれにまたがって、このサフィックスに対してマルチマスターレプリケーションアグリーメントを設定します。

また、可用性のために各チャンクはレプリケートされています。したがって、各チャンクに対して少なくとも 2 つのマスターレプリカが定義されています。

次の図は、各チャンクに対して 3 つのレプリカが定義された Directory Server 設定を示しています。簡略化のため、1 つのチャンクに対するレプリケーションアグリーメントのみを示していますが、ほかの 2 つのチャンクに対しても同じアグリーメントが定義されています。

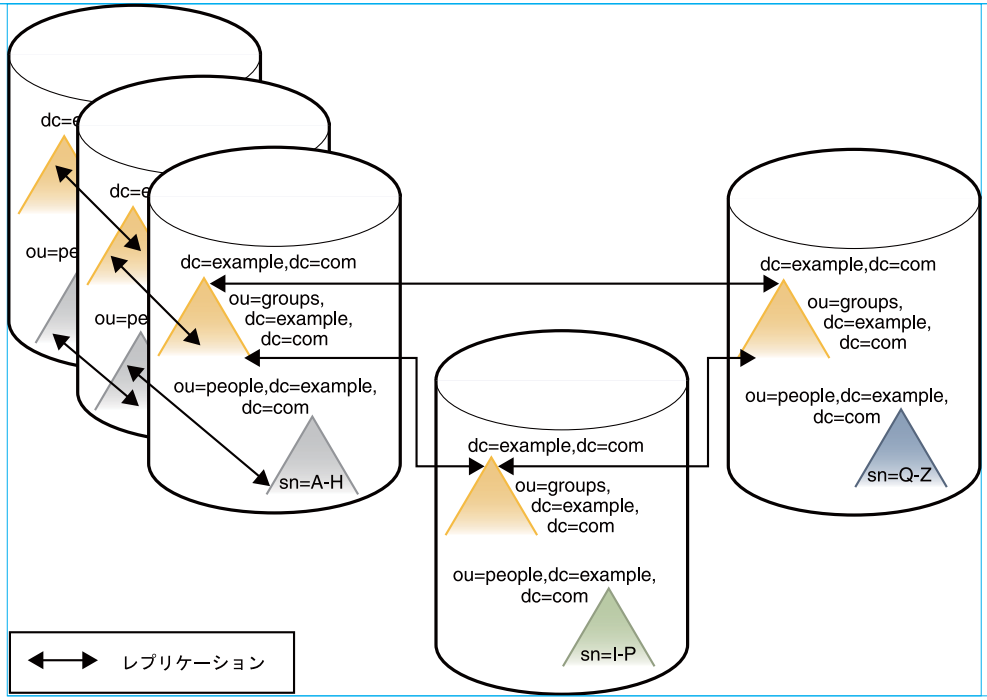


図 10–12　Directory Server 設定

サンプル配備シナリオ用の Directory Proxy Server 設定

クライアントによる Directory Proxy Server 経由でのディレクトリデータへのアクセスは「データビュー」を通して提供されます。データビューの詳細は、Chapter 17, 「Directory Proxy Server Distribution,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

このシナリオでは、分散される各サフィックスに対して 1 つのデータビューが必要であり、ネーミングコンテキスト (`dc=example,dc=com`) および `ou=groups` サブツリーに対して 1 つのデータビューが必要です。

次の図では、分散データへのアクセスを提供するための Directory Proxy Server データビューの設定を示しています。

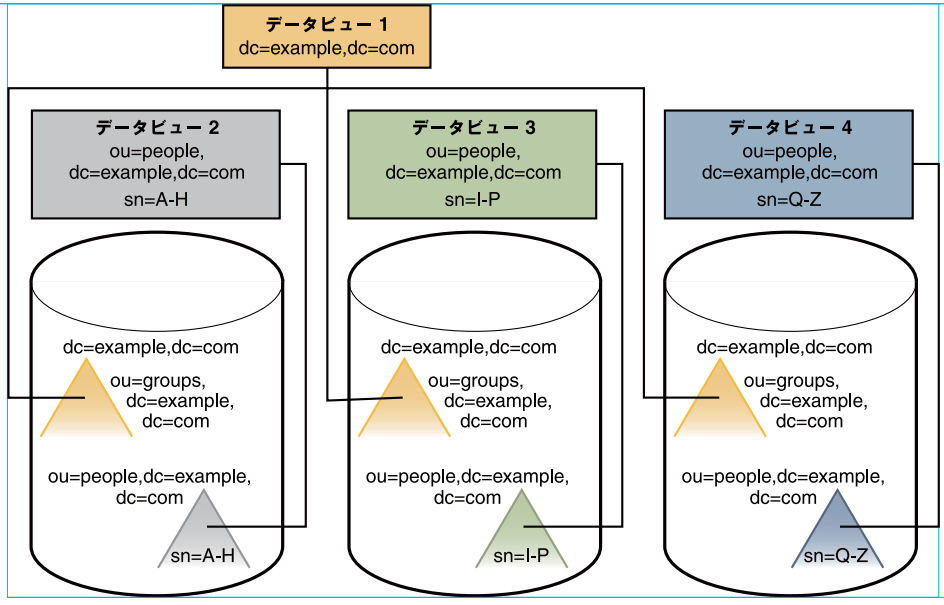


図 10–13　Directory Proxy Server 設定

データ増加に関する考慮事項

分散されるデータは、分散アルゴリズムに従って分割されます。使用する分散アルゴリズムを決定するときは、データの分量が変化する可能性があることと、分散方針は拡張性のあるものでなければならないことに留意してください。データの完全な再分散を必要とするようなアルゴリズムを使用しないでください。

たとえば、`uid` などの数値に基づく分散アルゴリズムは、比較的簡単に拡張できます。`uid=0-999` および `uid=1000–1999` の 2 つのデータセグメントから開始する場合、`uid=2000–2999` という 3 番目のセグメントをあとから追加することは容易です。

リフェラルを使用した分散

リフェラルはサーバーによって返される情報であり、操作要求を進めるためにどのサーバーと通信すればよいかをクライアントアプリケーションに通知します。分散ロジックの管理に Directory Proxy Server を使用しない場合、分散されるデータ間の関係を別の方法で定義する必要があります。関係を定義する 1 つの方法は「リフェラル」の使用です。

Directory Server では、リフェラル返送の形式とタイミングを 3 つの方法で設定できます。

- デフォルトリフェラル: クライアントアプリケーションが提示した DN に対応するサフィックスがサーバーにない場合、ディレクトリはデフォルトリフェラルを返します。
- サフィックスリフェラル: サフィックス全体がメンテナンスまたはセキュリティー上の理由でオフラインになった場合、サーバーはサフィックスに定義されているリフェラルを返します。クライアントが書き込み処理を要求する場合、サフィックスの読み取り専用レプリカも、マスターサーバーにリフェラルを返します。
- スマートリフェラル: スマートリフェラルは、ディレクトリ自体のエントリに格納されています。このリフェラルは、そのスマートリフェラルが格納されているエントリの DN と一致する DN を持つサブツリーに関する情報を保有している Directory Server を指します。

次の図は、リフェラルを使用して、グローバルトポロジ内の適切なサーバーにイギリスの顧客を誘導する方法を例示したものです。このシナリオでは、クライアントアプリケーションがリフェラルに従うためには、クライアントアプリケーションがトポロジ内のすべてのサーバーに (TCP/IP レベルで) 接続できる必要があります。

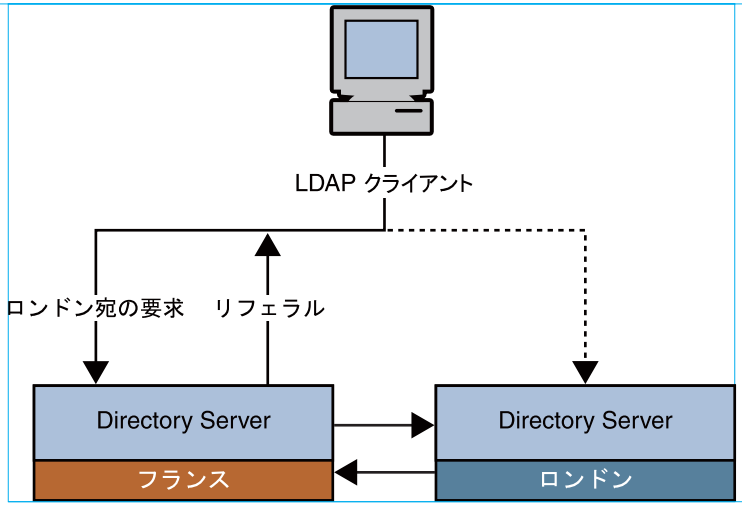


図 10–14 リフェラルを使用した特定サーバーへのクライアントの誘導

Directory Proxy Server でのリフェラルの使用

Directory Proxy Server とリフェラル機構を組み合わせて使用することにより、同じ結果を達成することができます。この目的のために Directory Proxy Server を使用することの利点は、クライアントアプリケーションの負荷および複雑さが軽減されること

です。クライアントアプリケーションは Directory Proxy Server の URL のみを認識します。何らかの理由で分散ロジックが変更された場合でも、この変更はクライアントアプリケーションに対して透過的です。

次の図は、前に説明したシナリオを、Directory Proxy Server を使用することによって簡略化する方法を例示しています。クライアントアプリケーションは常に Proxy Server と通信し、リフェラル自体は Proxy Server によって処理されます。

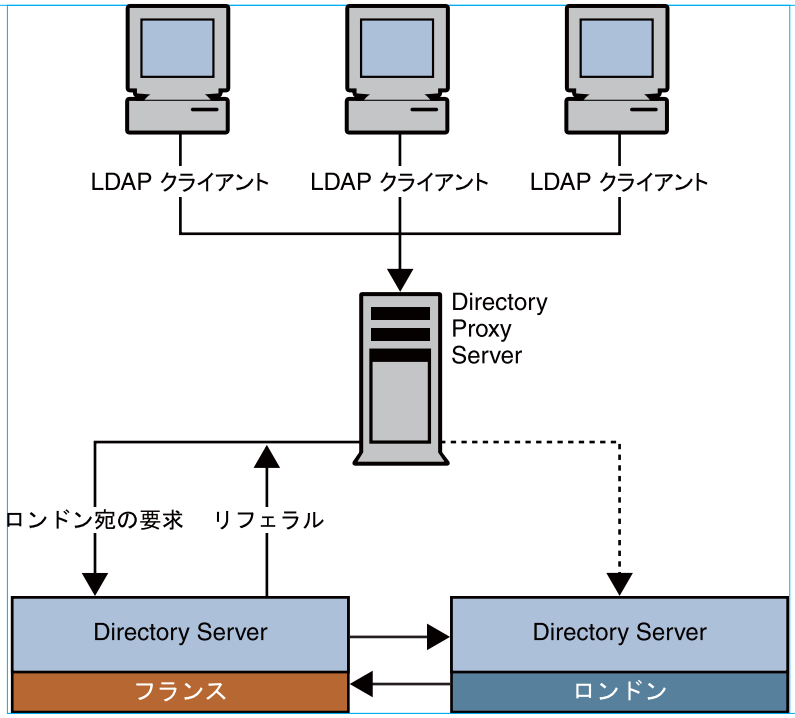


図 10–15　Directory Proxy Server でのリフェラルの使用

◆ ◆ ◆ 第 11 章

11 グローバル配備の設計

グローバル配備では、1 つ以上の地域にあるデータセンターのディレクトリサービスへのアクセスが必要になります。この章では、複数のデータセンターにわたって Directory Server Enterprise Edition を効率的に配備するための戦略について説明します。これらの戦略によって、第 5 章で説明されているサービスの品質に対する要件は満足されることが保証されます。

この章の内容は次のとおりです。

複数のデータセンターにわたるレプリケーションの使用

レプリケーションの 1 つの目標は、LDAP サービスの地理的な分散を可能にすることです。レプリケーションを使用すると、複数のサーバー上や、複数のデータセンターにわたって情報の同一のコピーを保持することができます。レプリケーションの概念については、このガイドの第 10 章でその概要が、さらにChapter 4, 「Directory Server Replication,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』でその詳細が説明されています。この章では、グローバル配備で使用されるレプリケーション機能に焦点を絞ります。

WAN を介したマルチマスターレプリケーションの使用

Directory Server では、WAN を介したマルチマスターレプリケーションをサポートしています。この機能により、マルチマスターレプリケーション設定を地理的に離れた場所にある複数のデータセンターに国際的に配備できます。

一般に、178 ページの「初期レプリケーション要件の評価」で計算されるホストの数が 16 より小さいか、またはそれより大幅に大きくはない場合、トポロジを、完全に接続されたトポロジ内にマスターサーバーのみが含まれる、つまり、すべてのマスターがトポロジ内のほかのすべてのマスターにレプリケートする状態にしてください。WAN 構成を介したマルチマスターレプリケーションでは、WAN で分離されたすべての Directory Server インスタンスが、Directory Server 5.2 より前のバージョンを実行していないようにしてください。4 つを超えるマスターを含むマルチマスタートポロジの場合は、Directory Server 6.x が必要です。

レプリケーションプロトコルでは、完全な非同期サポートのほか、ウィンドウ、グループ化、および圧縮のメカニズムが提供されます。これらの機能によって、WAN を介したマルチマスターレプリケーションが実行可能になります。レプリケーションのデータ転送速度は、帯域幅の点から見て、使用可能な物理媒体で可能になる速度を常に下回ります。レプリカ間の更新の量が、物理的に、使用可能な帯域幅に収まりきらない場合は、チューニングを行なっても、更新の重い負荷の下でのレプリカの発散を避けることはできません。レプリケーションの遅延や更新のパフォーマンスは、変更の頻度、エントリサイズ、サーバーハードウェア、平均待ち時間、平均帯域幅など (ただし、これらには限定されない) を含む多くの要因に依存します。

デフォルトでは、レプリケーションメカニズムの内部パラメータは WAN に合わせて最適化されています。ただし、上で述べた要因のためにレプリケーションの速度低下が発生している場合は、ウィンドウサイズやグループサイズのパラメータを実験的に調整することをお勧めします。また、ネットワークのピーク時間帯を避けるようにレプリケーションをスケジュールして、全体的なネットワーク使用率を向上させることができる可能性もあります。最後に、Directory Server は、帯域幅の使用を最適化するためにレプリケーションデータの圧縮に対応しています。

WAN リンクを介してデータをレプリケートする場合は、データの完全性と機密性を保証する何らかの形式のセキュリティーを確保することをお勧めします。Directory Server で使用可能なセキュリティー手段の詳細については、Chapter 2,「Directory Server Security,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。

グループとウィンドウのメカニズム

Directory Server は、レプリケーションの流れを最適化するためのグループとウィンドウのメカニズムを提供しています。グループのメカニズムを使用すると、変更を個別にではなく、グループで送信するように指定できます。グループサイズは、1 つの更新メッセージにまとめることのできるデータ変更の最大数を表します。ネットワーク接続がレプリケーションのボトルネックになっているように見える場合は、グループサイズを増やし、レプリケーションのパフォーマンスをもう一度チェックしてください。グループサイズの設定については、「Configuring Group Size」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

ウィンドウのメカニズムは、サプライヤが処理継続のためのコンシューマからの受信通知を待つことなく、コンシューマに特定の数の更新要求を送信するように指定します。ウィンドウサイズは、コンシューマからの即座の受信通知がなくても送信できる更新メッセージの最大数を表します。メッセージごとに受信通知を待つのではなく、多数のメッセージをすばやく連続して送信する方がより効率的です。適切なウィンドウサイズを使用することにより、レプリカがレプリケーションの更新または受信通知の到着を待つために費やす時間を削除できます。コンシューマレプリカがサプライヤよりも遅れている場合、詳細な調整を行う前に、ウィンドウサイズをデフォルトよりも大きい数字 (100 など) に設定して、レプリケーションのパフォーマンスをもう一度確認してみます。レプリケーションの更新頻度が高く、そのため更新の間隔が短い場合は、LAN で接続されているレプリカでもウィンドウサイズを大きくすると利点が得られます。ウィンドウサイズの設定については、「Configuring Window Size」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

グループとウィンドウのメカニズムはどちらも、変更のサイズに基づいています。そのため、変更のサイズが大幅に変動する場合、これらのメカニズムを使用してレプリケーションのパフォーマンスを最適化することは実用的でない場合があります。変更のサイズが比較的均一である場合は、グループとウィンドウのメカニズムを使用して、差分更新と完全更新を最適化することができます。

レプリケーションの圧縮

グループ化とウィンドウのメカニズムに加えて、Solaris および Linux プラットフォームではレプリケーションの圧縮も設定できます。レプリケーションの圧縮は、レプリケーションの流れを効率化します。それによって、WAN を介したレプリケーションでのボトルネックの発生率が大幅に削減されます。レプリケートされるデータを圧縮すると、CPU 性能は十分だが帯域幅が狭いネットワークや、一括変更をレプリケートする場合など、特定の場合でのレプリケーションのパフォーマンスを向上させることができます。また、大きなエントリを含むリモートレプリカを初期化する場合も、レプリケーションの圧縮によって利点が得られます。広いネットワーク帯域幅が存在する LAN (ローカルエリアネットワーク) ではこのパラメータを設定しないでください。圧縮と圧縮解除の計算によってレプリケーションの速度が低下するためです。

レプリケーションメカニズムは、Zlib 圧縮ライブラリを使用します。予測されるレプリケーション使用率に対して WAN 環境で最高の結果が得られる圧縮レベルを実験的にテストし、選択してください。

レプリケーションの圧縮を設定する方法の詳細については、「Configuring Replication Compression」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

部分レプリケーションの使用

グローバルなトポロジ (データセンターが各国に存在する) には、セキュリティーまたは遵守の理由から、レプリケーションの制限が必要になる場合があります。たとえば、法律上の規制で、特定の従業員情報を米国外にはコピーできないと規定されている可能性があります。または、オーストラリア内のサイトにはオーストラリアの従業員の詳細情報のみが必要になる可能性があります。

部分レプリケーション機能を使用すると、エントリ内に存在する属性のサブセットのみをレプリケートできます。属性リストは、レプリケートできる属性とレプリケートできない属性を判定するために使用されます。部分レプリケーションは、読み取り専用のコンシューマに対してのみ適用できます。

部分レプリケーションの動作の詳細については、「Fractional Replication」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。部分レプリケーションを設定する方法については、「Fractional Replication」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

優先順位付きレプリケーションの使用

優先順位付きレプリケーションは、特定の属性に関してレプリケートされるデータのより厳密な一貫性を確保する、強いビジネス要件が存在する場合に使用できます。5.x バージョンの Directory Server の場合、更新は、受信された順序でレプリケートされていました。優先順位付きレプリケーションでは、トポロジ内のほかのサーバーにレプリケートするときに、特定の属性に対する更新を優先させるように指定できます。

優先順位付きレプリケーションには、次の利点があります。

- セキュリティーの強化。優先順位付きレプリケーションは、アカウントのロックアウトのためにデフォルトで使用されます。たとえば、ある従業員が組織を離れたために、その従業員のアカウントをロックする場合を考えてみます。その従業員が、アカウントのロックアウトがまだレプリケートされていないリモートサーバーにログインできてしまうことがないように、アカウントのロックアウトの変更は、ほかの変更がレプリケートされる前にレプリケートされます。
- 一貫性の向上。Directory Server のレプリケーションは、一貫性に関して厳密ではありません。優先順位付きレプリケーションを使用すると、組織内で重要であると見なされる特定の属性のより強力な一貫性を保証できます。

国際的な企業のレプリケーション戦略のサンプル

このシナリオでは、ある企業の 2 つの主要なデータセンターがロンドンとニューヨークに 1 つずつあり、WAN で分離されています。ここでは、通常業務時間内はネットワークが非常に混み合っていることを前提にしています。

このシナリオでは、ホストの数が 8 と計算されています。2 つのデータセンターのそれぞれに、完全に接続された 4 方向のマルチマスタートポロジが配備されています。これらの 2 つのトポロジもまた、相互に完全に接続されています。簡単のために、次の図には 2 つのデータセンター間のすべてのレプリケーションアグリーメントは示されていません。

このシナリオでのレプリケーション戦略には、次の内容が含まれます。

- ディレクトリデータのマスターコピーを、両方のデータセンターのそれぞれのサーバーで保持する。
- 配備全体にわたって高可用性と書き込みフェイルオーバーを実現するために、データセンター間にマルチマスターレプリケーショントポロジを配備する。
- 帯域幅を最適化するために、WAN リンク全体にわたるレプリケーションをスケジュールして、ピークを外れた時間帯にのみ実行されるようにする。
- パフォーマンスを向上させるために、クライアントアプリケーションをローカルサーバーに振り分ける。米国内のクライアントは、ニューヨークのデータセンター内のマスターとの間で読み取り/書き込みを行う。英国内のクライアントは、ロンドンのデータセンター内のマスターとの間で読み取り/書き込みを行う。

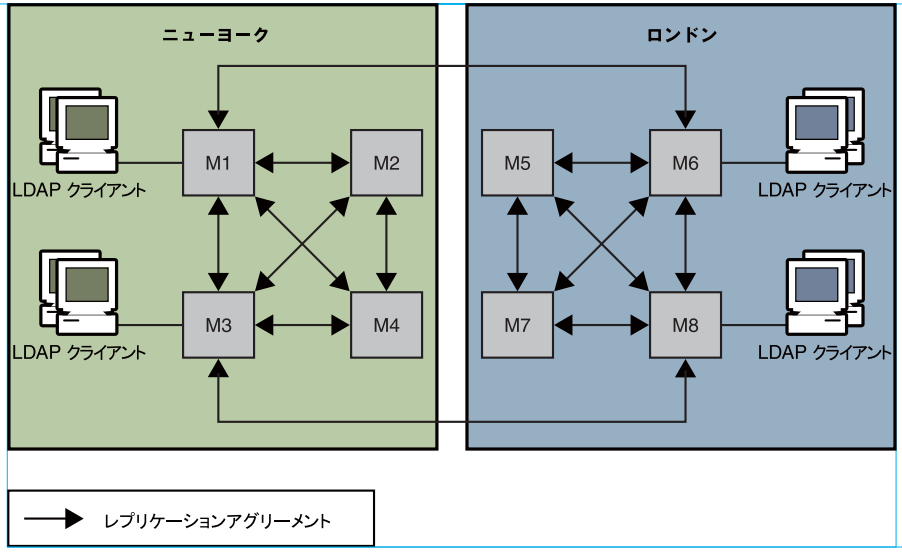


図 11–1　2 つのデータセンターでの負荷分散のためのマルチマスターレプリケーションの使用

グローバル配備での Directory Proxy Server の使用

グローバル企業では、集中管理データモデルのために、スケーラビリティーとパフォーマンスの問題が発生する場合があります。このような状況で Directory Proxy Server を使用すると、データを効率的に分散させ、検索要求や更新要求を適切に経路指定することができます。

グローバル企業のための分散戦略のサンプル

ここに示されているアーキテクチャーでは、大きな金融機関の本社がロンドンに存在します。この組織は、ロンドン、ニューヨーク、および香港にデータセンターを保有しています。現在、従業員が使用可能なデータの大部分は、ロンドンにある旧バージョンの RDBMS リポジトリに集中して格納されています。この金融機関のクライアントコミュニティーからのこのデータへのすべてのアクセスが WAN を介して行われます。

この組織は、この集中管理モデルでスケーラビリティーとパフォーマンスの問題を経験しており、分散データモデルに移行することに決定しました。この組織はまた、同時に LDAP ディレクトリインフラストラクチャーも配備することに決定しました。問題のデータは「ミッションクリティカル」であると見なされるため、可用性の高い、フォールトトレラントなインフラストラクチャーに配備する必要があります。

クライアントアプリケーションのプロファイル分析によって、データが顧客ベースのものであることがわかりました。そのため、地域のクライアントコミュニティーによってアクセスされるデータの 95 パーセントは、そのコミュニティーに固有のデータです。アジアのクライアントが北米の顧客のデータにアクセスすることもありますが、この状況はまれにしか発生しません。また、クライアントコミュニティーはときどき、顧客情報の更新も行う必要があります。

次の図は、分散ソリューションの論理的なアーキテクチャーを示しています。

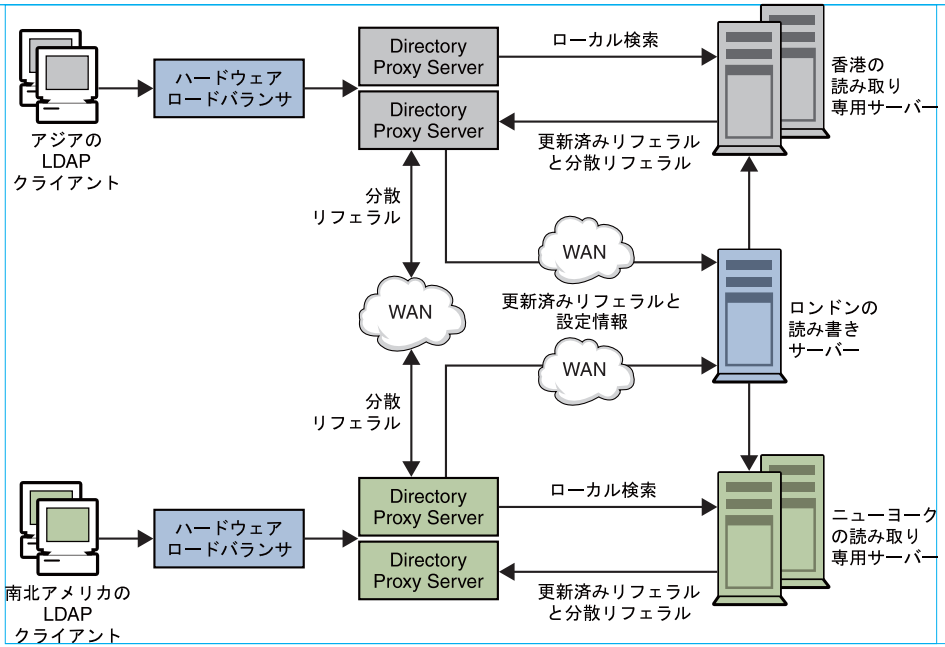


図 11–2　分散したディレクトリインフラストラクチャー

95 パーセントがローカルデータへのアクセスであるというプロファイル結果から、この組織は、ディレクトリインフラストラクチャーを地理的に分散することに決定しました。香港、ニューヨーク、およびロンドンのそれぞれに、複数のディレクトリコンシューマが配備されます。簡単のために、この図にはロンドンのコンシューマは示されていません。これらの各コンシューマは、各地域固有の顧客データを保持するように設定されます。ヨーロッパと中近東の顧客のデータは、ロンドンのコンシューマに保持されます。北アメリカと南アメリカの顧客データは、ニューヨークのコンシューマに保持されます。アジアと環太平洋地域の顧客のデータは、香港のコンシューマに保持されます。

この配備では、コミュニティーのデータ要求のほとんどがローカルのコミュニティーに対するものです。そのため、集中管理モデルを大幅に上回るパフォーマンスが実現されます。クライアント要求がローカルに処理されるため、ネットワークのオーバーヘッドが軽減されます。ローカルのディレクトリサーバーがディレクトリインフラストラクチャーを効率的に区分化するため、ディレクトリサーバーのパフォーマンスとスケーラビリティーが向上します。コンシューマディレクトリサーバーの各セットは、クライアントが更新要求を送信した場合はリフェラルを返すように設定されます。また、クライアントが、どこか別の場所に格納されているデータに対する検索要求を送信した場合にもリフェラルが返されます。

クライアントの LDAP 要求は、ハードウェアロードバランサを介して Directory Proxy Server に送信されます。このハードウェアロードバランサによって、クライアントが

常に、少なくとも 1 つの Directory Proxy Server にアクセスできることが保証されます。ローカルに配備された Directory Proxy Server は最初に、すべての要求を、ローカルの顧客データを保持するローカルのディレクトリサーバーの配列に経路指定します。Directory Proxy Server のインスタンスは、ディレクトリサーバーの配列全体にわたって負荷分散するように設定されます。この負荷分散によって、自動フェイルオーバーおよびフェイルバックが実現されます。

ローカルの顧客情報に対するクライアントの検索要求は、ローカルディレクトリによって満足されます。Directory Proxy Server を介して、クライアントに適切な応答が返されます。地理的に「外部にある」顧客情報に対するクライアントの検索要求は、最初に、Directory Proxy Server にリフェラルを戻すことによって、ローカルのディレクトリサーバーによって満足されます。

このリフェラルには、地理的に分散した対応する Directory Proxy Server インスタンスをポイントする LDAP URL が含まれています。ローカルの Directory Proxy Server は、このリフェラルを、ローカルクライアントの代わりに処理します。ローカルの Directory Proxy Server は次に、この検索要求を Directory Proxy Server の対応する分散インスタンスに送信します。分散した Directory Proxy Server は、この検索要求を分散した Directory Server に転送し、適切な応答を受信します。次に、この応答が、Directory Proxy Server の分散インスタンスとローカルインスタンスを介してローカルクライアントに返されます。

ローカルの Directory Proxy Server が受け取った更新要求に対して、最初に、ローカルの Directory Server がリフェラルを返します。Directory Proxy Server は、このリフェラルを、ローカルクライアントの代わりに処理します。ただし、この場合、プロキシはこの更新要求をロンドンに配置されているサプライヤディレクトリサーバーに転送します。サプライヤ Directory Server は、この更新をサプライヤデータベース適用し、その応答をローカルの Directory Proxy Server を介してローカルクライアントに戻します。その後、サプライヤ Directory Server は、この更新を適切なコンシューマ Directory Server に伝えます。

◆ ◆ ◆ 第 12 章

12 高可用性配備の設計

高可用性とは、ディレクトリサービスに関して同意された最小の「稼働時間」とパフォーマンスのレベルを示しています。同意されたサービスレベルは、組織ごとに異なります。サービスレベルは、システムがアクセスされる 1 日の中の時間帯、システムを保守のために停止できるかどうか、組織にとっての停止時間のコストなどの要因によって異なる可能性があります。ここでは、ディレクトリサービスがこの最小レベルのサービスを提供できなくなるあらゆる要因を障害と定義します。

この章の内容は次のとおりです。

可用性とシングルポイント障害

高可用性を提供する Directory Server Enterprise Edition 配備は、障害からすばやく回復できます。高可用性配備では、コンポーネント障害が個別のディレクトリクエリーに影響する可能性はありますが、完全なシステム障害には陥りません。シングルポイント障害 (SPOF) のシングルポイントとは、システム内でそのコンポーネントに障害が発生すると、システム全体が使用不可になるか、または信頼できなくなってしまうようなコンポーネントを意味します。高可用性配備を設計する場合は、潜在的な SPOF を識別し、どのようにしたらこれらの SPOF を軽減できるかを調査します。

SPOF は、次の 3 つのカテゴリに分類できます。

- サーバーのクラッシュ、ネットワーク障害、停電、ディスクドライブのクラッシュなどのハードウェア障害
- Directory Server または Directory Proxy Server のクラッシュなどのソフトウェア障害
- データベース破壊

SPOF の軽減

「冗長性」を使用することにより、単一コンポーネントの障害によってディレクトリサービス全体が障害に陥ることのないように保証できます。冗長性には、冗長性のあるソフトウェアコンポーネント、ハードウェアコンポーネント、またはその両方の提供が含まれます。この例には、個別のホストへの Directory Server の複数のレプリケートされたインスタンスの配備や、Directory Server データベースのストレージでの RAID (Redundant Arrays of Independent Disks) の使用などがあります。レプリケートされた Directory Server による冗長性は、高可用性を実現するためのもっとも効率的な方法です。

また、クラスタリングを使用して高可用性サービスを提供することもできます。クラスタリングには、パッケージ化済みの高可用性ハードウェアおよびソフトウェアの提供が含まれます。この例として、Sun Cluster ハードウェアおよびソフトウェアの配備があります。

冗長性とクラスタリングのどちらを使用するかの決定

この章の残りでは、高可用性を確保するための冗長性とクラスタリングの使用についてさらに詳細に説明します。この節では、各ソリューションの利点と欠点の概要について説明します。

冗長性の利点と欠点

高可用性ディレクトリサービスを提供するためのより一般的なアプローチは、冗長性のあるサーバーコンポーネントとレプリケーションの使用です。冗長ソリューションは通常、クラスタリングソリューションより安価であり、実装もより簡単です。また、これらのソリューションは一般に、管理も容易です。冗長ソリューションの一部としてのレプリケーションには、可用性以外にも多くの機能があることに注意してください。レプリケーションの主な利点は読み取り負荷を複数のサーバーにわたって分割できることですが、サーバー管理の点から見ると、この利点のためにオーバーヘッドが増えます。レプリケーションではまた、読み取り操作に関するスケーラビリティーと、正しく設計されている場合は、一定の制限内で書き込み操作に関するスケーラビリティーも提供されます。レプリケーションの概念の概要については、Chapter 4, 「Directory Server Replication,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference』 を参照してください。

障害の間に冗長システムが提供する可用性は、クラスタリングソリューションより劣る可能性があります。たとえば、負荷が 2 つの冗長性のあるサーバーコンポーネント間で共有されている環境を考えてみます。1 つのサーバーコンポーネントの障害によってもう一方のサーバーに過剰な負荷がかかり、それによって、クライアント要求へのこのサーバーの応答が遅くなることがあります。応答の遅さは、すばやい応答時間に依存しているクライアントには障害と見られる場合があります。つまり、サービスの可用性が (そのサービスが運用に入っているにもかかわらず)、クライアントの可用性の要件を満足しない可能性があります。

クラスタリングの利点と欠点

クラスタ化されたソリューションの主な利点は、障害からの自動復旧、つまりユーザーの介入を必要としない復旧です。クラスタリングの欠点は、複雑さと、データベース破壊からは回復できない点です。

クラスタ化された環境では、どのクラスタノードが実際にサービスを実行しているかには関係なく、クラスタは Directory Server と Directory Proxy Server に同じ IP アドレスを使用します。つまり、IP アドレスは、クライアントアプリケーションには影響がありません。レプリケートされた環境では、トポロジ内の各マシンに独自の IP アドレスが割り当てられます。この場合は、Directory Proxy Server を使用して、ディレクトリトポロジへのシングルポイントアクセスを提供できます。したがって、レプリケーショントポロジは、クライアントアプリケーションからは実質的に隠されます。この透過性を向上させるために、Directory Proxy Server を、リフェラルと検索参照を自動的に処理するように設定できます。Directory Proxy Server ではまた、負荷分散や、1 台に障害が発生した場合に別のマシンに切り替える機能も提供されます。

冗長性とクラスタリングでの SPOF の処理方法

この章の最初に説明した SPOF の点から見ると、冗長性とクラスタリングは、障害を次の方法で処理します。

- 単一のハードウェア障害: クラスタ化された環境では、この種の障害はディレクトリサービスに影響を与えません。クラスタ内のサービスに影響を与えるのは、複数のハードウェア障害だけです。

  クラスタ化された環境にはないマシンにとって、単一のハードウェア障害は致命的です。そのため、冗長性のあるハードウェアが存在する場合でも、障害を修復するために手動の介入が必要です。

- **Directory Server** または **Directory Proxy Server** の障害: クラスタ化された環境では、サーバーは自動的に再起動されます。ソフトウェア障害が短期間で連続して複数回発生しないと、クラスタ内の別のノードへのサービスグループの切り替えをトリガーできない場合があります。ソフトウェア障害に対するこの処理は、冗長性のある環境にも当てはまります。
- データベース破壊: クラスタは、この種の障害に耐えることができません。構成によっては、冗長ソリューションはデータベース破壊に耐えることができるはずです。

ハードウェアレベルでの冗長性

ここでは、ハードウェア冗長性に関する基本的な情報について説明します。多くの刊行物が、高可用性を実現するためのハードウェア冗長性の使用に関する包括的な情報を提供しています。特に、John Wiley & Sons, Inc. によって発行された『Blueprints for High Availability』を参照してください。

ハードウェア SPOF は、次のように広範囲に分類できます。

- ネットワーク障害
- Directory Server または Directory Proxy Server が実行されている物理サーバーの障害
- ロードバランサの障害
- ストレージサブシステムの障害
- 電源装置の障害

ネットワークレベルでの障害は、冗長性のあるネットワークコンポーネントを配置することによって軽減できます。配備を設計する場合は、次の冗長性のあるコンポーネントを配置することを検討してください。

- インターネット接続
- ネットワークインタフェースカード
- ネットワーク配線
- ネットワークスイッチ
- ゲートウェイとルーター

冗長性のあるロードバランサをアーキテクチャーに追加することによって、SPOF としてのロードバランサを軽減できます。

データベース破壊の発生については、可用性を確保するためのデータベースフェイルオーバー手法を確立するようにしてください。冗長性のあるサーバーコントローラを使用することにより、ストレージサブシステム内の SPOF を軽減できます。また、コントローラとストレージサブシステムの間の冗長性のある配線、冗長性のあるストレージサブシステムコントローラ、または RAID を使用することもできます。

電源装置が 1 つしかない場合は、この電源がなくなるとサービス全体が使用不可になる可能性があります。この状況を回避するには、ハードウェアへの冗長電源装置の供給 (可能な場合) や、電源の分散化を考慮してください。電源装置内の SPOF を軽減するためのその他の方法には、サージプロテクタ、複数の電源供給元、およびローカルバッテリバックアップの使用や、ローカルでの電源の生成などがあります。

データセンター全体の障害は、たとえば、自然災害が特定の地域を襲った場合に発生します。この場合は、適切に設計された複数データセンターレプリケーショントポロジにより、分散ディレクトリサービス全体が使用不可になることを回避できます。詳細については、207 ページの「高可用性を実現するためのレプリケーションと冗長性の使用」を参照してください。

ソフトウェアレベルでの冗長性

Directory Server または Directory Proxy Server の障害には、次のものが含まれる可能性があります。

- 過剰な応答時間
- 書き込みのオーバーロード
  - ファイル記述子の飽和
  - ファイルシステムの飽和
  - ストレージ設定の不具合
  - 過剰なインデックス
- 読み取りのオーバーロード
- キャッシュの問題
- CPU の制約
- レプリケーションの問題
  - 同期
  - レプリケーションの伝播遅延
  - レプリケーションの流れ
  - レプリケーションのオーバーロード
- 大規模なワイルドカード検索

これらの SPOF は、Directory Server と Directory Proxy Server の冗長性のあるインスタンスを配置することによって軽減できます。ソフトウェアレベルでの冗長性には、レプリケーションの使用が含まれます。レプリケーションによって、冗長性のあるサーバーの同期が維持されること、および要求を停止時間なく再経路指定できることが保証されます。詳細については、207 ページの「高可用性を実現するためのレプリケーションと冗長性の使用」を参照してください。

高可用性を実現するためのレプリケーションと冗長性の使用

レプリケーションを使用すると、1 つのサーバーの停止により、ディレクトリサービスが使用不可になることを回避できます。信頼できるレプリケーショントポロジを構築することで、サーバー障害が発生した場合でも、データセンターにアクセスするすべてのクライアントは最新のデータに確実にアクセスできます。最低でも、ローカルのディレクトリツリーは、少なくとも 1 つのバックアップサーバーにレプリケートする必要があります。ディレクトリの設計者の中には、データの信頼性を最大限保証するために物理的な場所ごとに 3 回はレプリケートしておく必要があると言う人もいます。フォールトトレランスのためにレプリケーションをどれだけ使用するかを決定する場合は、ディレクトリで使用されているハードウェアとネットワークの品質を考慮してください。信頼性の低いハードウェアでは、より多くのバックアップサーバーが必要になります。

通常のデータバックアップポリシー代わりにレプリケーションを使用しないでください。ディレクトリデータのバックアップについては、147 ページの「バックアップと復元のポリシーの設計」およびChapter 9, 「Directory Server Backup and Restore,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

LDAP クライアントアプリケーションは通常、1 つの LDAP サーバーだけを検索するように設定します。異なる DNS ホスト名に配置されている LDAP サーバーを循環するように、カスタムクライアントアプリケーションを作成することができます。それ以外の場合、LDAP クライアントアプリケーションは、Directory Server の単一の DNS ホスト名を検索するようにしか設定できません。バックアップ用の Directory Server へのフェイルオーバーを実現するには、Directory Proxy Server、DNS ラウンドロビン、またはネットワークソートを使用できます。DNS ラウンドロビンまたはネットワークソートの設定と使用については、DNS のマニュアルを参照してください。このコンテキストで Directory Proxy Server がどのように使用されるかについては、209 ページの「冗長ソリューションの一部としての Directory Proxy Server の使用」を参照してください。

ディレクトリ内のデータを読み取る機能を維持するには、適切な負荷分散戦略を配備する必要があります。読み取りの負荷を複数のレプリカに分散するには、ソフトウェアとハードウェアの両方の負荷分散ソリューションを利用できます。また、これらの各ソリューションは、各レプリカの状態を判定し、それらの負荷分散トポロジへの参加を管理することもできます。これらのソリューションは、総合性や精度の点ではそれぞれ異なっている可能性があります。

地理的に離れた場所にまたがって書き込みフェイルオーバーを維持するには、WAN を介した複数データセンターレプリケーションを使用します。それには、各データセンターに少なくとも 2 つのマスターサーバーを設定し、WAN を介して完全にメッシュ化されるようにそれらのサーバーを設定することが必要です。この戦略によって、トポロジ内のどのマスターに障害が発生してもサービスが失われることはありません。書き込み可能なサーバーが使用不可になった場合は、書き込み操作を代替のサーバーに経路指定する必要があります。書き込み操作の経路を再指定するには、Directory Proxy Server を使用するなど、さまざまな方法があります。

以降の節では、高可用性を確保するためにレプリケーションと冗長性がどのように使用されるかについて説明します。

冗長性のあるレプリケーションアグリーメントの使用

冗長レプリケーションアグリーメントを使用すると、障害発生時の迅速な復旧が可能になります。レプリケーションアグリーメントを有効にしたり無効にしたりできるということは、元のレプリケーショントポロジに障害が発生した場合にのみ使用されるレプリケーションアグリーメントを設定できることを意味します。有効/無効の切り替えは手動で行う必要がありますが、この方法は、レプリケーションアグリーメントが必要になった時点ではじめて設定する場合に比べて、費やす時間がはるかに短くて済みます。冗長レプリケーションアグリーメントの使用は、210 ページの「高可用性を実現するために冗長性を使用したサンプルのトポロジ」で図を使用して説明されています。

レプリカの昇格と降格

レプリカの昇格または降格によって、レプリケーショントポロジでのそのロールが変更されます。専用のコンシューマとハブを含む非常に大規模なトポロジでは、レプリカのオンライン昇格および降格によって、高可用性戦略の一部が形成される場合があります。たとえば、負荷分散とフェイルオーバーのために 2 つのハブが設定されたマルチマスターのレプリケーション環境を例に考えてみます。1 つのマスターがオフラインになった場合は、読み取り/書き込みの最適な可用性を維持するために、いずれかのハブをマスターに昇格できます。マスターレプリカがオンラインに復帰したときは、ハブレプリカに降格させるだけで元のトポロジを回復できます。

詳細については、「Promoting or Demoting Replicas」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

冗長ソリューションの一部としての **Directory Proxy Server** の使用

Directory Proxy Server は、高可用性ディレクトリ配備をサポートするように設計されています。このプロキシによって、レプリケートされた一連の Directory Server 間での自動負荷分散や、自動フェイルオーバーおよびフェイルバックが実現されます。トポロジ内の 1 つ以上の Directory Server が使用不可になった場合は、その負荷が残りのサーバー間で比例的に再分散されます。

Directory Proxy Server は、Directory Server をアクティブに監視して、これらのサーバーが常にオンラインになっていることを保証します。このプロキシはまた、実行されている各操作の状態も検査します。すべてのサーバーが、スループットやパフォーマンスの点で同等であるとはかぎりません。主サーバーが使用不可になった場合、副サーバーに一時的にリダイレクトされたトラフィックは、主サーバーが使用可能になるとすぐにふたたび主サーバーに送信されるようになります。

データを分散する場合は、切り離された複数のレプリケーショントポロジを管理する必要があり、それによって管理が複雑になることに注意してください。さらに、Directory Proxy Server は、ユーザー承認の管理をプロキシ認証制御に大きく依存しています。この分散に関与している各 Directory Server で、特定の管理ユーザーを作成する必要があります。これらの管理ユーザーには、プロキシアクセス制御の権限を許可する必要があります。

アプリケーション分離による高可用性の実現

Directory Proxy Server を使用すると、レプリケートされたディレクトリサービスを問題のあるクライアントアプリケーションによる障害から保護することもできます。可用性を向上させるために、各アプリケーションには、マスターまたはレプリカのセットをいくつか限定して割り当てられます。

問題のあるアプリケーションが特定のアクションを実行して、サーバーの停止を引き起したとします。そのアプリケーションが各レプリカに次々とフェイルオーバーした場合に、1 つのアプリケーションでの単一の問題によって、レプリケートされるトポロジ全体が障害に陥る可能性があります。このような状況を回避するために、各アプリケーションのフェイルオーバーや負荷分散を、制限された数のレプリカに限定することができます。起こりうる障害の影響範囲が割り当てられたレプリカのセットにのみ限定されるため、ほかのアプリケーションへの障害の影響は軽減されます。

高可用性を実現するために冗長性を使用したサンプルのトポロジ

次のサンプルトポロジは、障害が発生した場合も継続的なサービスを提供するために冗長性をどのように使用するかを示しています。

1 つのデータセンターでの可用性向上のためのレプリケーションの使用

次の図に示されているデータセンターには、3 つのマスターを含むマルチマスタートポロジが存在します。このシナリオの場合、3 番目のマスターは、障害が発生した場合の可用性のためにのみ使用されます。読み取りおよび書き込み操作は、問題が発生しないかぎり、Directory Proxy Server によってマスター 1 とマスター 2 に経路指定されます。復旧を高速化し、レプリケーションアグリーメントの数を最小限に抑えるために、復旧レプリケーションアグリーメントが作成されています。これらのアグリーメントは、デフォルトでは無効になっていますが、障害が発生した場合にはすぐに有効にできます。

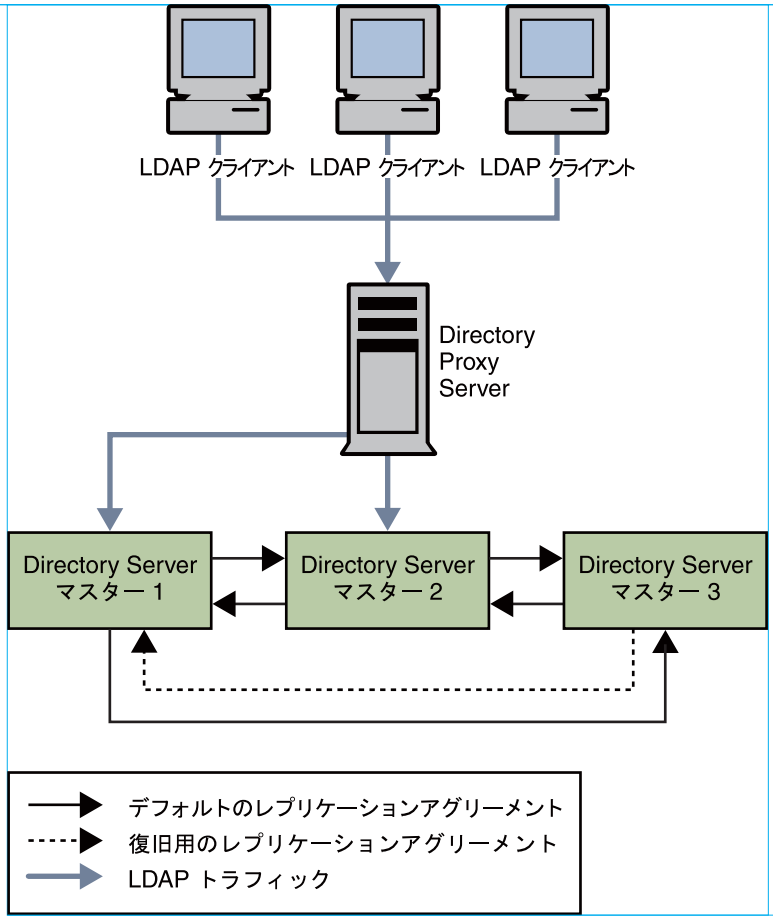


図 12–1　1 つのデータセンターでのマルチマスターレプリケーション

1 つのデータセンターの障害マトリックス

図 12–1 に示されているシナリオでは、さまざまなコンポーネントが使用不可になる可能性があります。これらの可能性のある障害の場所と、それに対応する復旧のアクションを次の表に示します。

表 12–1　1 つのデータセンターの障害マトリックス

障害が発生したコンポーネント	アクション
マスター 1	読み取りおよび書き込み操作は、マスター 1 が修復されている間、Directory Proxy Server を介してマスター 2 とマスター 3 に再経路指定されます。マスター 3 への更新がマスター 2 にレプリケートされるように、マスター 2 とマスター 3 の間の復旧レプリケーションアグリーメントが有効になります。
マスター 2	読み取りおよび書き込み操作は、マスター 2 が修復されている間、マスター 1 とマスター 3 に再経路指定されます。マスター 3 への更新がマスター 1 にレプリケートされるように、マスター 1 とマスター 3 の間の復旧レプリケーションアグリーメントが有効になります。
マスター 3	マスター 3 はバックアップサーバーでしかないため、このマスターに障害が発生しても、ディレクトリサービスは影響を受けません。サービスを中断することなく、マスター 3 をオフラインにしたり、修復したりできます。
Directory Proxy Server	Directory Proxy Server の障害は、重大なサービス中断を引き起します。このトポロジでは、Directory Proxy Server の冗長性のあるインスタンスを作成することをお勧めします。このようなトポロジの例については、216 ページの「複数の Directory Proxy Server の使用」を参照してください。

1 つのデータセンターでの復旧手順

1 つのデータセンターに 3 つのマスターが含まれている場合は、1 つのマスターに障害が発生しても、読み取りおよび書き込み機能が維持されます。ここでは、障害が発生したコンポーネントの復旧に適用できる復旧戦略の例について説明します。

次のフローチャートと手順では、マスター 1 という 1 つのコンポーネントに障害が発生したと仮定しています。2 つのマスターに同時に障害が発生した場合は、その問題を修正している間、読み取りおよび書き込み操作を残りのマスターに経路指定する必要があります。

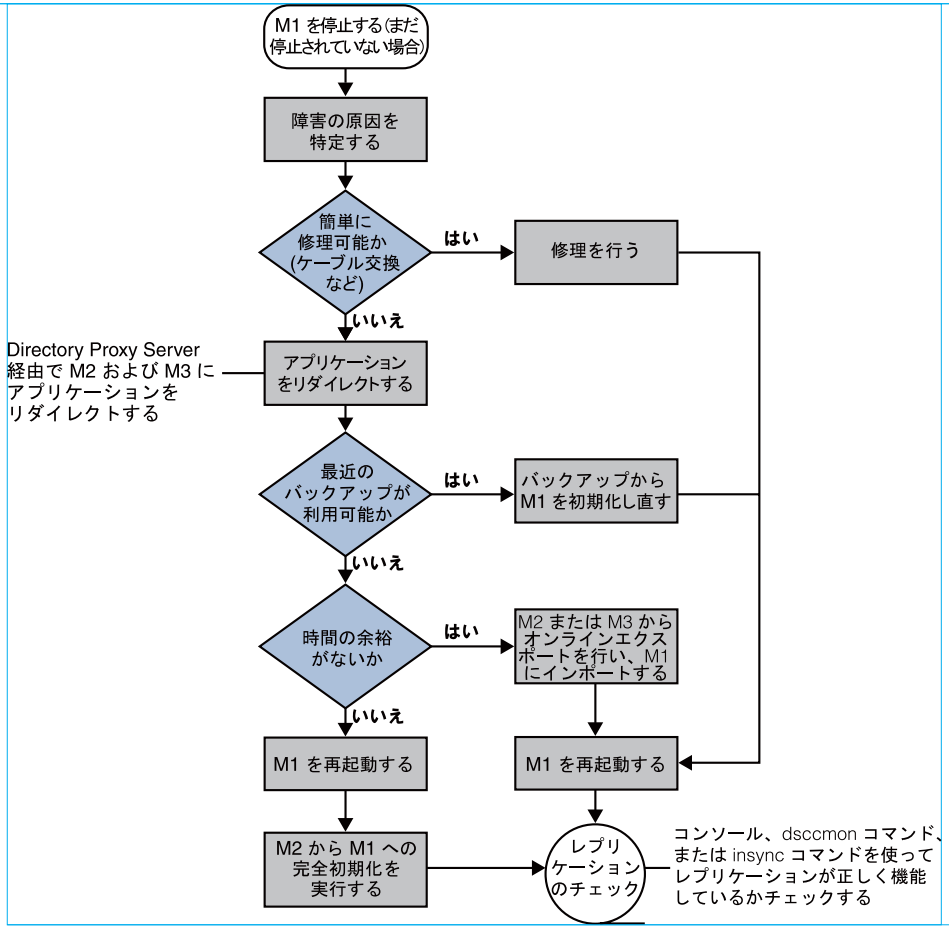


図 12–2　1 つのデータセンターでのサンプルの復旧手順

▼ 1 つのコンポーネントの障害から回復するには

1. マスター 1 がまだ停止されていない場合は、停止します。
2. 障害の原因を特定します。
   - 障害が容易に (たとえば、ネットワークケーブルの置き換えなどにより) 修復される場合は、修復を行い、手順 3 に進みます。

- より重大な問題の場合は、障害の修正にさらに多くの時間がかかる可能性があります。

a. マスター 1 にアクセスするすべてのアプリケーションが、**Directory Proxy Server** を介して、マスター 2 またはマスター 3 をポイントするようにリダイレクトされる必要があります。

b. 最新のバックアップが使用できることを確認します。

- 最新のバックアップが使用できる場合は、そのバックアップからマスター 1 を再初期化し、手順 3 に進みます。
- 最新のバックアップが使用「できない」場合は、次のいずれかの操作を行います
  - マスター 1 を再起動し、マスター 2 またはマスター 3 からマスター 1 への全体的な初期化を実行します。

    この手順の詳細については、「Initializing Replicas」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。
  - 全体的な初期化の実行に時間がかかりすぎる場合は、マスター 2 またはマスター 3 からのオンラインエクスポートを実行し、マスター 1 にインポートします。

3 マスター 1 がまだ起動されていない場合は、起動します。

4 マスター 1 が読み取り専用モードになっている場合は、読み取り/書き込みモードに設定します。

5 レプリケーションが正しく機能していることを確認します。

レプリケーションの確認には、DSCC の `dsccmon view-suffixes` または `insync` コマンドを使用できます。

詳細については、「Getting Replication Status」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』、`dsccmon`(1M)、および `insync`(1) を参照してください。

2 つのデータセンターにわたる可用性向上のためのレプリケーションの使用

一般に、2 つのデータセンターを含む配備では、1 つのデータセンターについて説明したのと同じ復旧戦略を適用できます。1 つ以上のマスターが使用不可になった場合は、Directory Proxy Server が、ローカルの読み取りおよび書き込みを残りのマスターに自動的に再経路指定します。

先に説明した 1 つのデータセンターのシナリオの場合と同様に、復旧レプリケーションアグリーメントを有効にすることができます。これらのアグリーメントによって、障害が発生した場合も、レプリケートされた更新を両方のデータセンターが引き続き受信できることが保証されます。この復旧戦略は、図 12–3 に示されています。

復旧レプリケーションアグリーメントを使用する代わりに、すべてのマスターが変更をほかのすべてのマスターにレプリケートする、完全にメッシュ化されたトポロジを使用する方法があります。レプリケーションアグリーメントは少ない方が管理しやすくなる可能性はありますが、完全にメッシュ化されたトポロジは使用しない方がよいという技術的な理由は何もありません。

このシナリオでの唯一の SPOF は、各データセンター内の Directory Proxy Server になります。図 12–4 に示すように、この問題を解消するために、冗長性のある Directory Proxy Server を配備することができます。

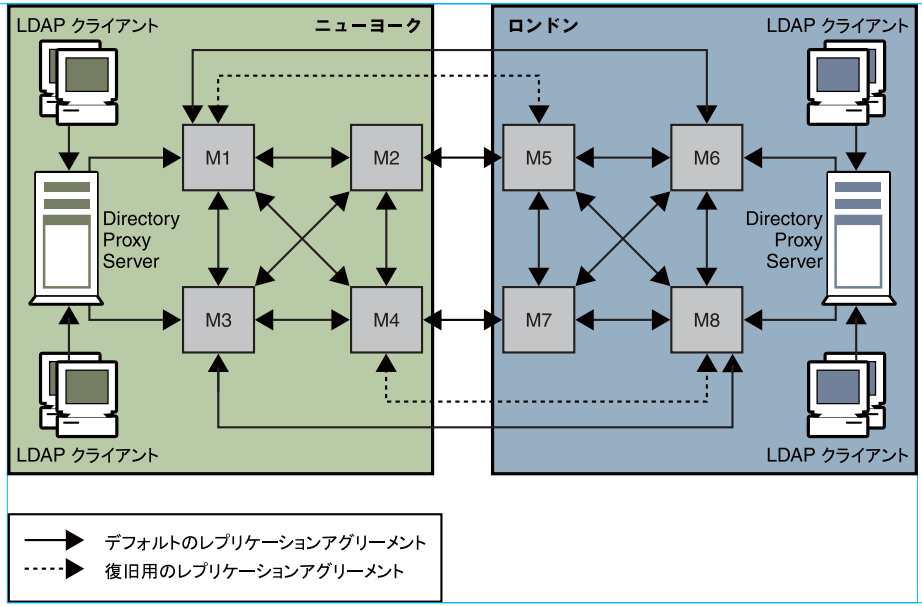


図 12–3　2 つのデータセンターのための復旧レプリケーションアグリーメント

復旧戦略は、どのような組み合わせでコンポーネントに障害が発生するかによって異なります。しかし、複数の障害に対する基本的な戦略を準備しておけば、その他のコンポーネントに障害が発生した場合にもそれを適用できます。

図 12–3 に示されているサンプルトポロジでは、ニューヨークのデータセンター内のマスター 1 とマスター 3 に障害が発生したと仮定しています。

このシナリオでは、Directory Proxy Server が、ニューヨークのデータセンター内の読み取りおよび書き込みをマスター 2 とマスター 4 に自動的に再経路指定します。これにより、ニューヨークのサイトのローカルの読み取りおよび書き込み機能が維持されることが保証されます。

複数の Directory Proxy Server の使用

次の図に示されている配備には、内部の LDAP サービスへの外部からのアクセスを拒否する企業ファイアウォールが含まれています。内部で開始されたクライアントの LDAP 要求は、ネットワークロードバランサを経由して Directory Proxy Server に転送されるため、IP レベルでの高可用性が確保されます。Directory Proxy Server を実行しているホストを除き、Directory Server への直接アクセスは阻止されます。プロキシが SPOF になるのを回避するために、2 つの Directory Proxy Server が配備されています。

完全にメッシュ化されたマルチマスタートポロジによって、ほかのどのマスターに障害が発生した場合でも、常にすべてのマスターを使用できることが保証されます。簡単のために、この図にはすべてのレプリケーションアグリーメントは示されていません。

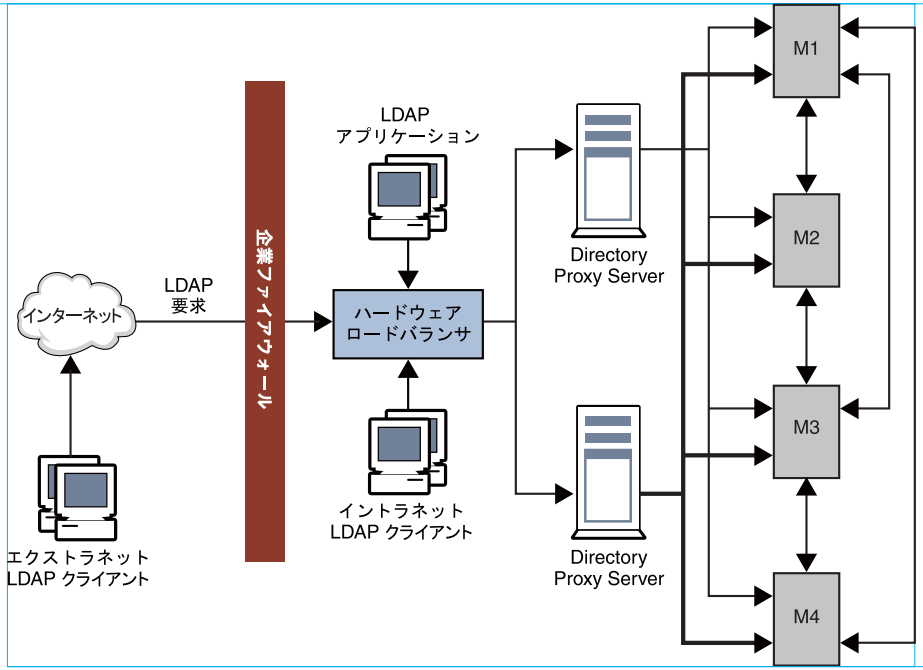


図 12–4　ファイアウォール内での高可用性設定

アプリケーション分離の使用

次の図に示されているシナリオでは、アプリケーション 1 のバグによって Directory Server に障害が発生しています。このプロキシ設定によって、アプリケーション 1 からの LDAP 要求がマスター 1 とマスター 3 にしか送信されないことが保証されます。このバグが発生すると、マスター 1 とマスター 3 に障害が発生します。ただし、アプリケーション 2、3、および 4 は、機能している Directory Server に引き続きアクセスできるため無効になりません。

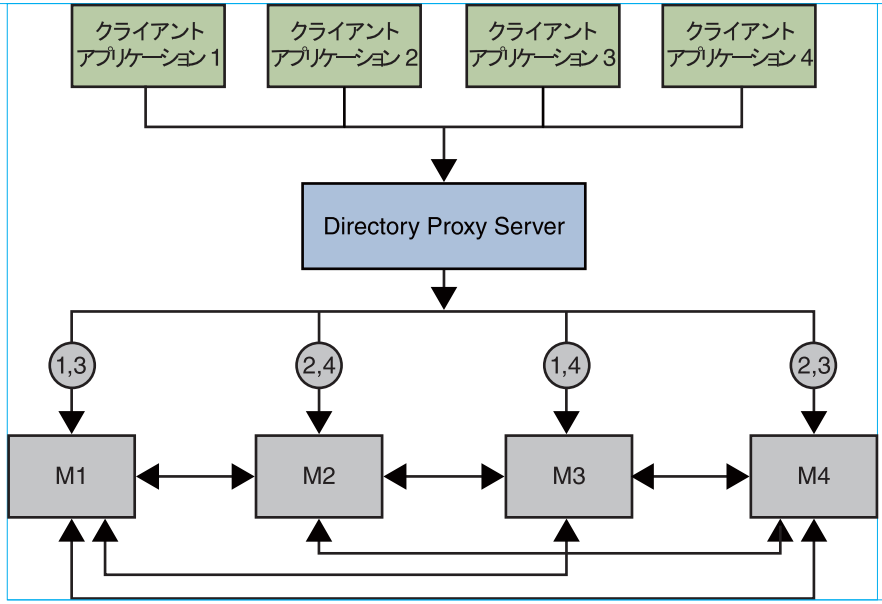


図 12–5　拡張配備でのアプリケーション分離の使用

高可用性を実現するためのクラスタリングの使用

物理的な観点から見ると、クラスタは、1 つのエンティティーとして連携して動作する 1 ～ 8 個のサーバーで構成されます。これらのサーバーは、アプリケーション、システムリソース、およびデータへの高可用性アクセスを提供するために連携して動作します。各サーバーを、複数の CPU を備えた対称型マルチプロセッサにすることができます。

クラスタリングソリューションは、次のコンポーネントに対して高可用性を提供できます。

- サーバーとソフトウェア
- ストレージサブシステム
- ネットワークアダプタ

クラスタリングによって、ディレクトリアーキテクチャー内のすべての SPOF が軽減されるわけではありません。外部ネットワーク、電源生成、およびデータセンターでの障害は、クラスタリングソリューション以外の手法で軽減するようにしてください。

ディレクトリサービスの可用性向上のために Sun Cluster 3.2 または 3.1 を使用するには、Sun Cluster HA for Directory Server データサービスをフェイルオーバーデータサービスとしてインストールおよび設定する必要があります。この戦略を使用すると、Sun Cluster 環境で Directory Server を安全にフェイルオーバーさせることができます。

次の図は、Sun Cluster アーキテクチャー内の Sun Cluster HA for Directory Server データサービスの位置付けを示しています。

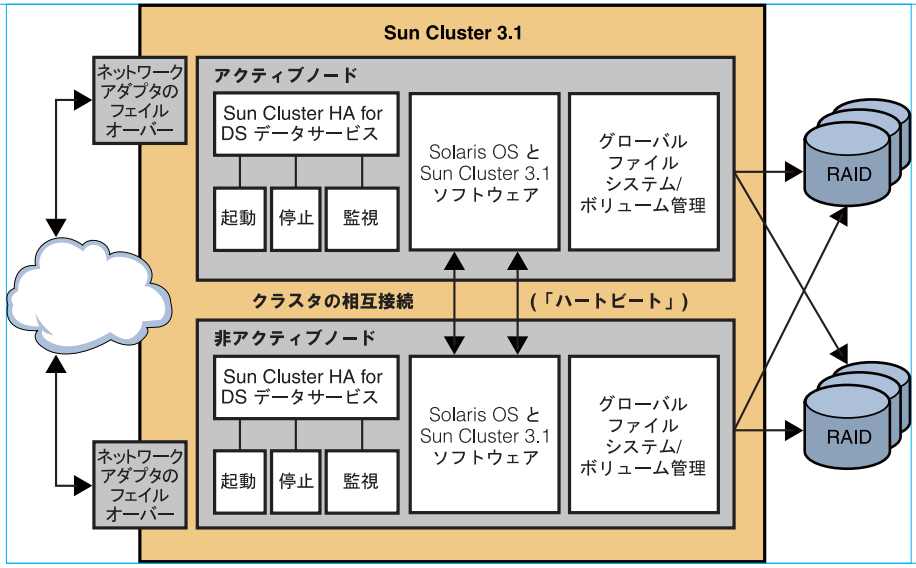


図 12–6　Sun Cluster アーキテクチャー

ハードウェア冗長性

Sun Cluster ハードウェアシステムのアーキテクチャーは、SPOF によってクラスタが使用不可にならないように設計されています。冗長性のある高速インターコネクト、ストレージシステム接続、およびパブリックネットワークによって、クラスタ接続に単一障害が発生しないことが保証されます。

クライアントは、パブリックネットワークインタフェースを介してクラスタに接続します。ネットワークアダプタカードに複数のハードウェアインタフェースがある場合、そのカードは 1 つ以上のパブリックネットワークに接続できます。複数の

ネットワークインタフェースカードを含むようにノードを設定できます。これらのカードは、1 枚のカードがアクティブであり、もう 1 枚のカードがバックアップとして機能するように設定されます。

クラスタファイルシステムは、1 つ以上のノード上のカーネルと、配下のファイルシステムおよびボリュームマネージャーの間のプロキシです。クラスタファイルシステムは、ディスクへの物理的な接続が存在するノード上で実行されます。クラスタファイルシステムの可用性を向上させるには、ディスクを複数のノードに接続してください。ローカルファイルシステムで構成したクラスタファイルシステムの可用性は高くありません。ローカルファイルシステムとは、ノードのローカルディスクに格納されているファイルシステムを指します。

ボリュームマネージャーによって、多重ホストディスクのデータ冗長性を向上させるためのミラー化構成または RAID 5 構成が可能になります。多重ホストディスクをディスクのミラー化およびストライプ化と組み合わせることにより、ノード障害と個別のディスク障害の両方から保護できます。

クラスタインターコネクトは、クラスタノード間のクラスタプライベート通信やデータサービス通信を転送するプライベートネットワークです。冗長性のある NIC、接続中継点、およびケーブルによって、ネットワーク障害から保護されます。

クラスタ化されたソリューションでの監視

クラスタでは、すべてのメンバーが継続的に監視されます。障害の発生したノードがクラスタに参加しないようにブロックされるため、破壊されたデータの交換が回避されます。また、クラスタはアプリケーションも監視し、障害が発生した場合は、そのアプリケーションをフェイルオーバーまたは再起動します。

Sun Cluster ソフトウェアのサブシステムである Public Network Management は、アクティブなインタフェースを監視します。アクティブなアダプタに障害が発生した場合は、そのインタフェースをいずれかのバックアップアダプタにフェイルオーバーするために Network Adapter Failover ソフトウェアが呼び出されます。

Cluster Membership Monitor (CMM) は、クラスタメンバーまたはノードごとに 1 セットが割り当てられた、エージェントの分散されたセットです。これらのエージェントは、クラスタインターコネクトを介してメッセージを交換することにより、すべてのノード間の完全な接続を保証します。CMM がノード障害などによるクラスタメンバーシップの変更を検出した場合は、CMM によってクラスタが再構成されます。CMM がノードに関する重大な問題を検出した場合、CMM はクラスタフレームワークに連絡します。次に、クラスタフレームワークがそのノードを強制的に停止し、クラスタメンバーシップからそのノードを削除します。

システム保守

保守が必要なデータやアプリケーションを現在のコンポーネントからシステム上の別のコンポーネントに移動することによって、システム保守の計画的な停止時間を最小限に抑えることができます。保守が完了したら、そのデータやアプリケーションを元のコンポーネントに戻すことができます。

Directory Server Failover Data Service

Directory Server Failover Data Service は、クラスタ内の 1 つのノード上で実行されます。ただし、ノードには、スケーラビリティー向上のために複数の CPU を実装できます。障害モニターが、このフェイルオーバーサービスを定期的に監視します。

Resource Group Manager (RGM) は、データサービスをリソースとして管理します。CMM がクラスタのメンバーシップを変更した場合は、RGM がそのクラスタのオンラインまたはオフラインリソースへの変更を要求する可能性があります。RGM は、フェイルオーバーデータサービスを開始および停止します。

障害からの回復

以降の節では、Directory Server Data Service に障害が発生した場合や、サーバーに障害が発生した場合にサービスがどのように回復されるかについて説明します。

アプリケーション障害が発生した場合の復旧

障害モニターが Directory Server Data Service に障害が発生したと判定した場合、モニターはそのサービスを再開するためのアクションを開始します。実行されるアクションは、そのサービスの設定によって異なります。

フェイルオーバーデータサービスを、障害の発生したサービスの、同じノード上での再開を試みるように設定できます。あるいは、障害の発生したサービスを別のノード上でただちに開始するようにデータサービスを設定することもできます。データサービスが、同じノード上での再開を試みるように設定されている場合、障害モニターはローカルの RGM に連絡します。ローカルの RGM は次に、障害の発生したサービスの再開を試みます。このアクションに失敗すると、ローカルの RGM は、別のノード上でのサービスの開始を試みます。

障害の発生したデータサービスを同じノード上で再開できない場合、ローカルノードの RGM は、別のノードにあるサービスのバージョンの検索を試みます。このアクションは、そのデータサービスが、障害のあと別のノード上で開始するように設定されている場合にも実行されます。ローカルの RGM がそのサービスのバージョンを見つけると、ローカルの RGM はローカルの CMM に連絡して、クラスタインターコネクトを介してリモートノードに連絡するよう要求します。リモートの CMM は次に、そこのローカルの RGM に連絡して、サービスを開始するよう指示します。

次の図は、アプリケーション障害が発生した場合の復旧を示しています。

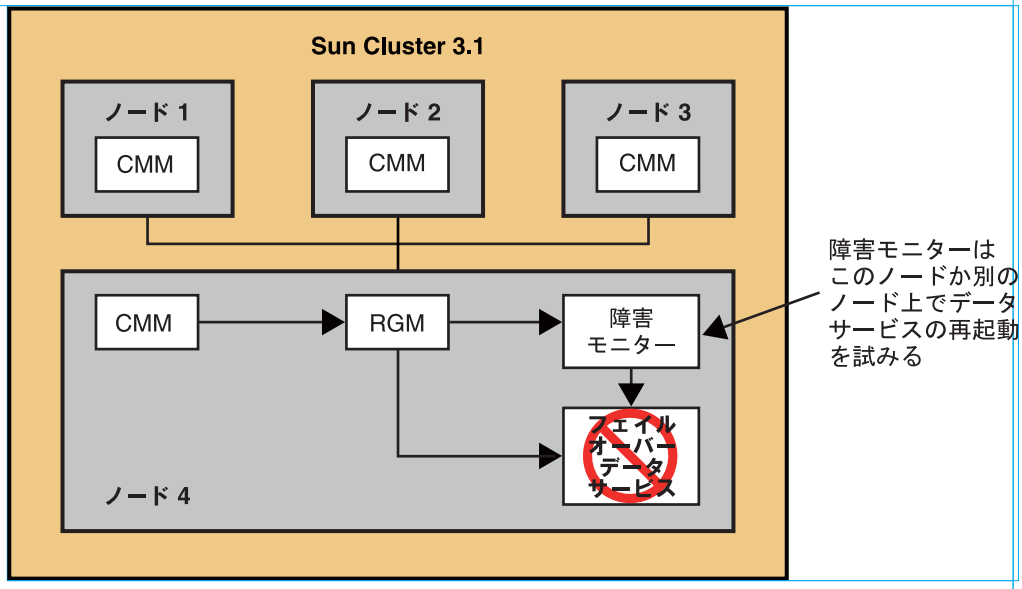


図 12–7　Sun Cluster アーキテクチャーでのアプリケーション障害と復旧

サーバー障害が発生した場合の復旧

Directory Server Data Service が実行されているサーバーまたはノードに障害が発生した場合、サービスは機能している別のノードに移行されます。ユーザーの介入は必要ありません。このサービスは、フェイルオーバーリソースグループ、Directory Server インスタンスを定義しているコンテナ、およびフェイルオーバーの要件をサポートしているホストを使用します。

次の図は、サーバー障害が発生した場合の復旧を示しています。

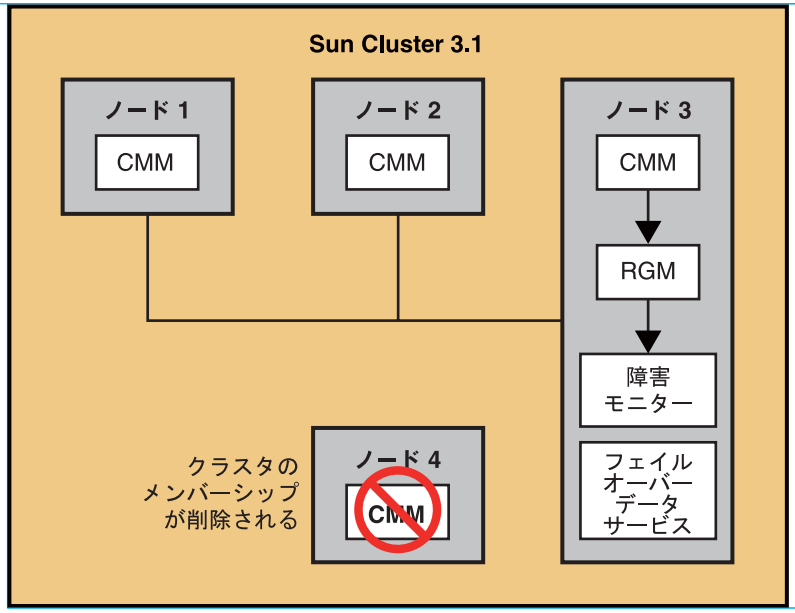


図 12–8　Sun Cluster アーキテクチャーでのサーバー障害と復旧

パート IV

高度な配備のトピック

この第 4 部では、特殊な配備に関するトピックを扱います。次の章で構成されています。

- 第 13 章では、LDAP ベースのネームサービスについて説明します。
- 第 14 章では、Directory Proxy Server の仮想化機能について説明します。
- 第 15 章では、Identity Synchronization for Windows を使用する配備について説明します。

◆ ◆ ◆ 第 13 章

13 Solaris での LDAP ベースネームサービスの使用

この章では、Solaris™ オペレーティングシステム (Solaris OS) で提供される LDAP ネームサービスの概要を示します。Solaris OS でサポートされるネームサービスについては、Part I, 「About Naming and Directory Services,」 in 『System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP) 』で詳しく説明しています。

この章の内容は次のとおりです。

LDAP ベースのネームサービスを使用する理由

ネームサービスは、ユーザー、マシン、およびアプリケーションがネットワーク上で互いに通信するために必要な情報を 1 つの場所にまとめて格納します。この情報には、マシン (ホスト) の名前とアドレス、ユーザー名、パスワード、アクセス権、グループメンバーシップ、プリンタなどがあります。集中型のネームサービスがなければ、各マシンがこの情報のコピーを個別に維持しなければなりません。ネームサービスの情報はファイル、マップ、またはデータベーステーブルに格納できます。すべてのデータを集中化すれば、管理が容易になります。

Solaris OS では、次のネームサービスをサポートします。

- DNS (ドメインネームシステム)
- `/etc` ファイル (UNIX® オリジナルのネームシステム)
- NIS (ネットワーク情報サービス)
- NIS+ (ネットワーク情報サービスプラス)
- LDAP (Lightweight Directory Access Protocol)

ただし、Sun の戦略的な方向性は、LDAP ベースのネームサービスへの移行です。

LDAP ネームサービスには、ほかのネームサービスにない次の利点があります。

- アプリケーション固有のデータベースを置き換えることによって情報を統合し、管理が必要なデータベースの数を減らすことができる
- 複数の異なるネームサービス間でデータを共有できる
- データの集中リポジトリを提供する
- マスターサーバーとレプリカの間で、より頻繁なデータ同期が可能になる
- マルチプラットフォームおよびマルチベンダー互換

LDAP ネームサービスには、次の制限があります。

- Solaris 8 よりも前のクライアントはサポートされていません。
- LDAP ネームサービスの設定と管理はほかに比べて複雑であり、慎重な計画が必要です。
- 同じクライアントマシン上で NIS クライアントとネイティブ LDAP クライアントが共存できません。

Solaris OS は、LDAP ディレクトリサーバーに加えて、Sun Java System Directory Server との組み合わせで LDAP ネームサービスをサポートします。Sun Java System Directory Server の使用も推奨されていますが、必須ではありません。

NIS から LDAP への移行

NIS から LDAP への移行は、データ移行とクライアント移行の 2 つのステップからなるプロセスです。Solaris OS には、NIS から LDAP への移行サービス (N2L サービス) が用意されており、このサービスを利用して両方のステップを実行できます。

N2L サービスは、NIS マスターサーバー上の既存の NIS デーモンを、NIS から LDAP への移行デーモンに置き換えます。またこのサービスは、NIS から LDAP へのマッピングサービスをそのサーバー上に作成します。マッピングファイルでは、NIS のマップエントリと、それに対応する LDAP のディレクトリ情報ツリー (DIT) エントリの間のマッピングを指定します。この移行処理を経た NIS マスターのことを N2L サーバーと呼びます。

NIS スレーブサーバーは、通常どおりに機能し続けます。スレーブサーバーは N2L サーバーを通常どおりの NIS マスターとして認識し、N2L サーバーから定期的にデータを更新します。`inityp2l` スクリプトは、これらの設定ファイルの初期定義を支援します。N2L サーバーが確立されたら、設定ファイルを直接編集することによって N2L を保守できます。

N2L サービスは次の機能をサポートします。

- LDAP DIT への NIS マップのインポート
- NIS と同等の速度および拡張性を備えた、DIT 情報へのクライアントアクセス

NIS から LDAP への移行方法の詳細は、Chapter 15, 「Transitioning From NIS to LDAP (Overview/Tasks),」 in 『System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP) 』を参照してください。

NIS+ から LDAP への移行

NIS+ のデータと LDAP の同期を保つことができますが、そのような同期には、以前は外部エージェントが必要でした。しかしながら、新しい NIS+ デーモンにより、LDAP サーバーを NIS+ データのデータリポジトリとして使用できるようになりました。この機能により、NIS+ クライアントと LDAP クライアントが同じネームサービス情報を共有できます。したがって、メインのネームサービスとして NIS+ を使用する構成から、LDAP を使用する構成への移行が容易になりました。

NIS+ から LDAP への移行方法の詳細は、Chapter 16, 「Transitioning From NIS+ to LDAP,」 in 『System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP) 』を参照してください。

◆ ◆ ◆ 第 14 章

14 仮想ディレクトリの配備

「仮想ディレクトリ」は Directory Proxy Server の高度な機能の 1 つであり、複数のデータリポジトリの情報をリアルタイムで集約します。この章では、Directory Server Enterprise Edition 配備における仮想ディレクトリの使用方法について説明します。

仮想ディレクトリのアーキテクチャー上の概念については、Chapter 18,「Directory Proxy Server Virtualization,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Reference 』を参照してください。仮想ディレクトリの設定手順については、Chapter 23,「Directory Proxy Server Virtualization,」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

この章の内容は次のとおりです。

どのような場合に仮想ディレクトリを使用すべきか

ディレクトリサービスの要件に次のいずれかが含まれる場合に、仮想ディレクトリの機能を配備できます。

- 複数のデータリポジトリ上に分散するエントリの集約されたビューを、クライアントアプリケーションが必要としている

  たとえば、いくつかのディレクトリサーバーが存在しており、それらのサーバーには同じユーザーが含まれているが、それらがすべて異なるデータであるとします。仮想ディレクトリを使えば、あるユーザーのすべてのディレクトリにわたるエントリの単一ビューを作成できます。さらに、仮想ディレクトリは、個々のすべてのディレクトリを管理するための単一点として機能します。

  サポートされるデータリポジトリの種類としては、LDAP ディレクトリ、MySQL などの Java Database Connectivity (JDBC™) に準拠したソース、LDIF フラットファイルなどが挙げられます。
- ユーザー資格情報とアプリケーション固有データにそれぞれ異なるデータストアが必要である。

  たとえば、アプリケーションによっては、企業ディレクトリ内に格納したくない固有データが存在する可能性があります。仮想ディレクトリを使えば、そうしたデータを分離しても、それがアプリケーションのソースの 1 つとして見えるようにすることができます。これにより、アプリケーション開発やデータ管理が単純化されます。なぜなら、アプリケーションがデータインフラストラクチャーの詳細を知る必要がなくなるからです。さらに、バックエンドデータソースに対する変更をアプリケーションから抽象化できます。
- ユーザーの企業が別の企業を買収したか、別の企業と合併した。

  仮想ディレクトリを使えば、2 つの企業のディレクトリをマージし、それらが単一のディレクトリとして表示されるようにすることができます。たとえば、2 つのディレクトリ `dc=example,dc=com` および `dc=acquisition,dc=com` があるとします。どちらのディレクトリも `dc=example,dc=com` のように見えることを要求するクライアントアプリケーションも存在します。
- データベースのテーブルが DIT 階層の形式で表示されることを、クライアントアプリケーションが要求する。
- 複数のデータリポジトリに対する読み取りおよび書き込み操作が必要である。
- 類似点のない属性名に基づいて複数のフィールドを結合する条件が必要となる。
- 複数の LDAP または JDBC バックエンドのディレクトリやデータベース上に分散する複数値属性のサポートを、クライアントアプリケーションが要求する。
- 属性の名前変更、DN の書き換え、および DN 構文属性の属性値の書き換えが必要となる。
- クライアントアプリケーションごとに、ある単一データリポジトリの異なるビューが必要となる。

仮想ディレクトリの典型的なシナリオ

この節では、仮想ディレクトリがどのようにして特定のビジネス要件を実現するかを示す単純なシナリオを提供します。より複雑なサンプルシナリオや仮想ディレクトリ構成の詳細については、「Sample Virtual Configurations」 in 『Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide 』を参照してください。

異なるデータソースからのユーザー ID の結合

Example.com ストアでは、異なる 3 つのデータリポジトリ内にユーザーデータを格納します。Example.com の Directory Server には、ユーザーデータの大部分が格納されています。ユーザーの電子メールアドレスは Active Directory 内に、HR データは MySQL データベース内に、それぞれ格納されています。

Example.com には、すべてのユーザーデータベースの完全なビューを必要とするクライアントアプリケーションがいくつか存在しています。次の図は、あるユーザーの ID の完全なビューを仮想ディレクトリがどのようにしてクライアントアプリケーションに提供するかを示したものです。

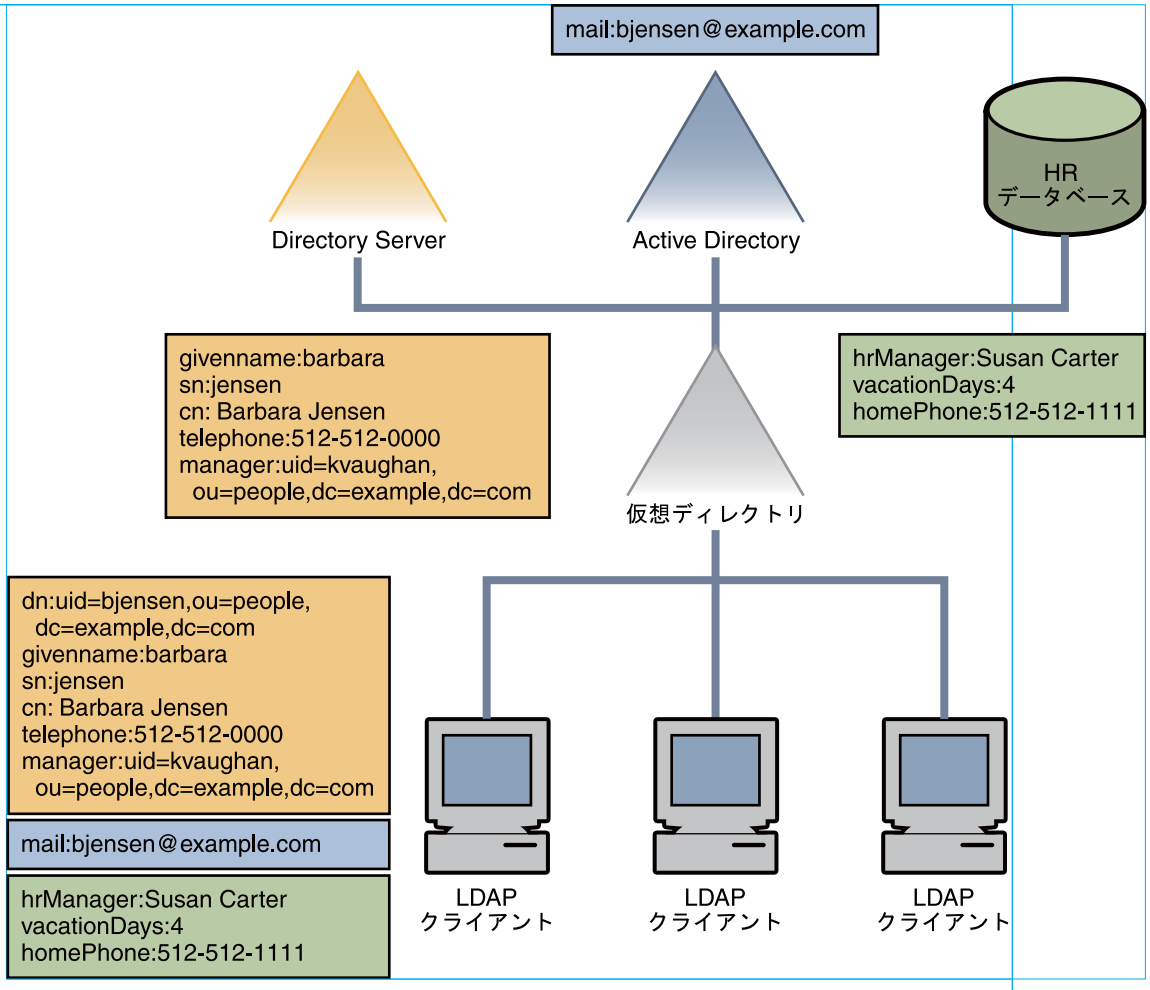


図 14–1　複数リポジトリからの集約データの仮想ビュー

既存ディレクトリ構造への新しい企業データのマージ

このシナリオでは、Example.com が新しい企業 Acquisition.com を買収します。この新しい企業のユーザーデータは、専用の Directory Server 内に格納されています。管理上の都合により、Example.com はこのディレクトリの構造を維持することを望んでいます。ただし、特定のクライアントアプリケーションでは、Acquisition.com のユーザーデータを「あたかも」Example.com のユーザーデータであるかのように表示する必要があります。

次の図は、仮想ディレクトリがどのようにして、買収した企業のデータを既存ディレクトリの構造内に仮想化マージするかを示したものです。

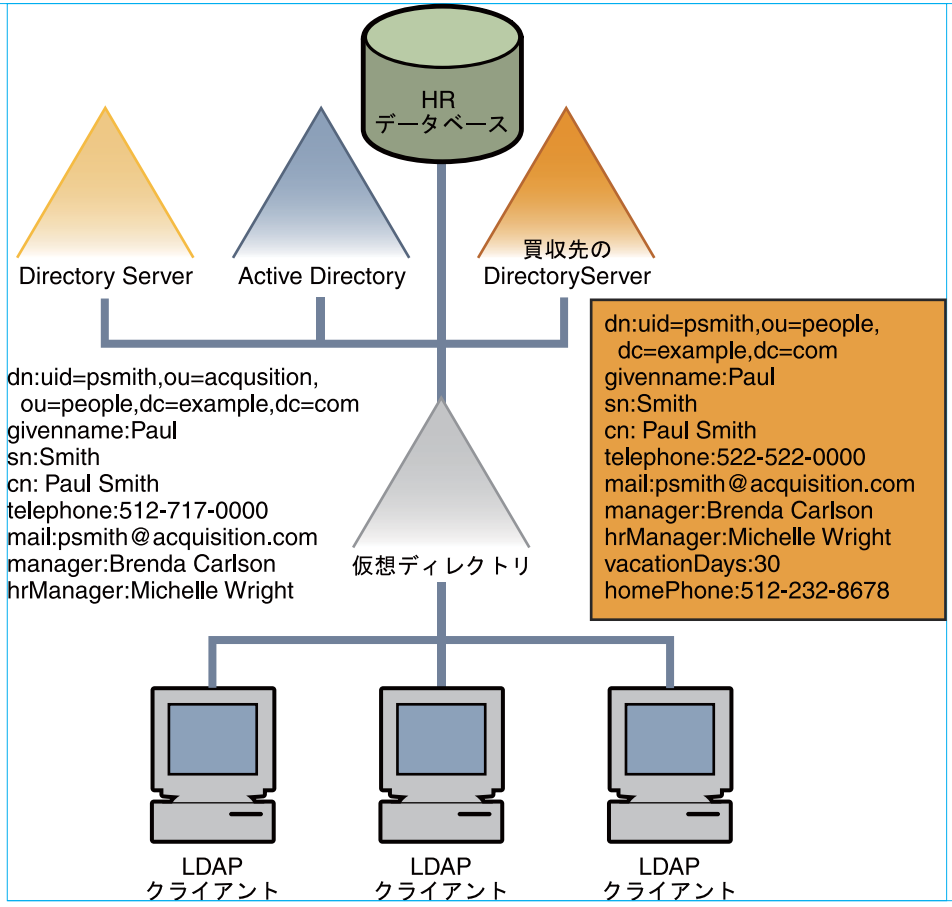


図 14–2 買収先ディレクトリのユーザーデータのマージ

Acquisition.com のディレクトリは、`ou=people` ブランチの下の個別のブランチとして表示されます。Acquisition.com のディレクトリ内のエントリの DN は、仮想ディレクトリ経由で表示される際に変換されます。

◆ ◆ ◆ 第 15 章

15 同期されるデータを持つ配備の設計

Directory Server Enterprise Edition のコンポーネントである Identity Synchronization for Windows は、パスワードなどのユーザーアカウント情報を Directory Server と Windows の間で同期します。Windows Active Directory と Windows NT の両方がサポートされています。Identity Synchronization for Windows は、あらゆる規模の企業にとって、拡張性と強化されたセキュリティーを備えたパスワード同期ソリューションの構築に役立ちます。

Identity Synchronization for Windows のマニュアルの一覧については、`http://docs.sun.com/coll/isw_04Q3` を参照してください。配備で Identity Synchronization for Windows の使用を計画している場合、この章で説明する問題に対処する必要があります。

Identity Synchronization for Windows の配備に関する考慮事項

- パスワードの同期方向: Directory Server から Active Directory の方向または双方向にパスワードを同期する場合、Windows 2000 に High Encryption Pack をインストールする必要があります。このインストールにより、Active Directory over LDAP でパスワードを設定するときに必要な 128 ビット SSL が有効になります。
- 新規ユーザー作成の同期: Identity Synchronization for Windows が新規ユーザーの作成を同期しない場合、`idsync resync` コマンドを定期的に実行して、新しく作成されたユーザー間のリンクを確立する必要があります。`idsync resync` を実行することによってユーザーが明示的にリンクされるまでの間、新しく作成されたユーザーへの変更は同期されません。
- 生成サイズ: Identity Synchronization for Windows では、同期可能なユーザー数の上限は設定されていませんが、ユーザーの総数は配備に影響を及ぼします。主に影響を受けるのは、同期を開始する前に実行する必要がある `idsync resync` コマンドです。同期されるユーザー数が 100,000 を超える場合、`idsync resync` コマンド

をバッチで実行します。このバッチモードは、最適なパフォーマンスを保証し、Sun Java System Message Queue にかかる負荷を制限します。

- パフォーマンス要件**:** Identity Synchronization for Windows のパフォーマンスを制限する要因としては、ユーザー総数よりも同期率のほうが重要です。この要件のただ 1 つの例外は、`idsync resync` コマンドを実行するときです。
- ピーク時の予測変更率**:** 同じシステム上でコアと 2 つのコネクタが稼働する Identity Synchronization for Windows の配備では、毎秒 10 件の同期という変更率が持続することは珍しくありません。必要な同期率がこの率を超える場合、Identity Synchronization for Windows を複数のマシンに分散させることによってパフォーマンスの向上を達成できます。たとえば、Identity Synchronization for Windows コアとは別のマシンにコネクタをインストールできます。
- 同期対象の **Windows** ドメイン数**:** 複数の Windows ドメインを同期する場合、`activedirectorydomainname` 属性または `USER_NT_DOMAIN_NAME` 属性を Directory Server 属性に同期する必要があります。この同期は、同期ユーザーリスト定義間のあいまいさを解決するために必要です。
- 配備内の **Directory Server** マスター、ハブ、および読み取り専用レプリカの数**:** 複数の Directory Server が存在する配備では、個々のマスターレプリカ、ハブレプリカ、および読み取り専用レプリカ上で Identity Synchronization for Windows Directory Server プラグインを有効にする必要があります。Identity Synchronization for Windows を設定するとき、1 つの Directory Server マスターが優先マスターとして指定されます。マスターの実行中、Directory Server コネクタは優先マスターで発生する変更を検出および適用します。このサーバーが停止した場合、コネクタは必要に応じて 2 番目のマスターに変更を適用できます。優先マスター上では Retro Changelog プラグイン (旧バージョン形式の更新履歴ログプラグイン) を有効にする必要があります。このマスターは、Identity Synchronization for Windows コアと同じ LAN 上に存在する必要があります。
- セキュリティー**:** Directory Server コネクタまたは Active Directory コネクタが SSL を使用して Directory Server または Active Directory に接続する場合、これらのサーバー上で SSL を有効にする必要があります。信頼できる証明書のみを受け入れるようにコネクタが設定されている場合、追加の設定手順を実行する必要があります。この手順では、適切な認証局証明書をコネクタの証明書データベースにインポートします。Directory Server プラグインと Active Directory の間で SSL が必要な場合、Directory Server で SSL を有効にする必要があります。加えて、Active Directory SSL 証明書に署名するために使用される認証局証明書を、Directory Server の証明書データベースにインポートする必要があります。

Identity Synchronization for Windows を組み込んだ詳細な配備シナリオについては、『Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide 』を参照してください。

索引