SiteMinder

Java 用プログラミング ガイド

12.52 SP1

ca technologies

CA Technologies 製品リファレンス

このマニュアルが参照している CA Technologies の製品は以下のとおりです。

- SiteMinder

CA への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

マニュアルの変更点

SiteMinder r12.51.または 12.52 では、このガイドに変更はありませんでした。

目次

第 4 章: ポリシー管理 API 65

CA Technologies 製品リファレンス

このマニュアルが参照している CA Technologies の製品は以下のとおりです。

- CA Technologies SiteMinder®

CA への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

第 1 章: Java API の使用

このセクションには、以下のトピックが含まれています。

Java API の目的

SiteMinder SDK は以下のタスクを実行するために Java API を提供します。

- SiteMinder エージェントの作成
- ポリシー管理アプリケーションの作成
- 分散代行管理サービス（DMS）アプリケーションの作成

インストール パス

Java API、ドキュメントおよびサンプルは以下の場所にインストールされます。

- UNIX プラットフォーム： <install_path>/sdk
- Windows プラットフォーム： <install_path>¥sdk

<install_path> は、SDK ソフトウェアのインストール時に指定したインストール パスです。

コード サンプル

SiteMinder SDK には、SiteMinder クライアント アプリケーションの信頼できるサンプルが含まれます。 これらのサンプルのソース ファイルは以下の場所にあります。

- UNIX プラットフォーム：

  <install_path>/sdk/samples/<api-name>

- Windows プラットフォーム：

  <install_path>¥sdk¥samples¥<api-name>

Java サンプルの注意事項

- サンプルは、/sdk/properties に置かれる smjsdksample.properties で定義されるプロパティを使用します。 Java サンプルを実行する前に、ユーザ環境の設定でこのファイルを変更します。
- smjsdksample.properties ファイルは、ログ記録に使用されるリテラル文字列の外部化も実行します。

- サンプル smjavaagentapi および javadmsapi は、サンプル javapolicyapi によって作成されたポリシー ストアを使用します。 smjavaagentapi または javadmsapi を実行する前に smjavapolicyapi を実行します。
- サンプルはすべて同じログ記録オプションおよび出力形式を使用します。
- 64 ビット UNIX オペレーティング環境で Java サンプルを実行する場合は以下に注意してください。
  - 64 ビット JVM を使用します。 32 ビット JVM を使用して、64 ビット サンプルを実行することはサポートされていません。
  - ファイル java-run.sh （samples/smjavaagentapi フォルダ内）を使用している場合は、-d64 フラグの前のコメント インジケータを削除します。32 ビット オペレーティング環境ではこのフラグはコメント アウトしたままにします。

ポリシー サーバ前提条件

SiteMinder SDK で開発するアプリケーションとポリシー サーバのプラグインを実行するには SiteMinder ポリシー サーバが必要です。 ただし、ほとんどの場合、それらのアプリケーションとプラグインの構築にはポリシー サーバを使用しません。 アプリケーション ランタイム ファイルは、ポリシー サーバにローカルであることもリモートであることも可能です。

SiteMinder SDK の Java コンポーネント

SiteMinder SDK の Java コンポーネントは以下の表にリスト表示されます。

API 名： **パッケージ名**	**プライマリ インターフェースおよびクラス**
Java エージェント API： netegrity.siteminder.javaagent	AgentAPI
ポリシー管理 API： com.netegrity.sdk.policyapi	SmPolicyApiImpl によって実装された SmPolicyApi

API 名: パッケージ名	プライマリ インターフェースおよびクラス
認証および許可 API com.netegrity.policyserver.smapi	SmAuthScheme （認証 API） ActiveExpression （許可 API）
分散代行管理サービス API： com.netegrity.sdk.dmsapi	SmDmsApiImpl によって実装された SmDmsApi
ユーティリティ パッケージ： com.netegrity.sdk.apiutil	SmApiConnection SmApiSession

Java エージェント API

Java エージェント API は、アクセス制御の適用とユーザ セッションの管理のためのカスタム エージェントを構築するのに使用します。 アクセス制御の適用は以下から構成されます。

- ユーザ認証
- ユーザ許可
- 監査

Java エージェント API を使用するとき、エージェントのコーディングには Java を使用します。エージェント API はポリシー サーバへの接続を実行します。

SiteMinder Java エージェント API には 2 つの実装があります。

- ネイティブ C/C++ エージェント API ライブラリに依存する JNI Java エージェント API。 この実装は、SiteMinder SDK のバージョン 5.x 以降で示されるインターフェースを使用します。
- JNI Java エージェント API で使用されるネイティブ コードを純粋な Java コンポーネントで置き換える、純粋な Java エージェント API。 この API の現在のバージョンは JNI Java エージェント API と同じインターフェースを使用します。

ネイティブ モードのコンポーネントがないので、純粋な Java エージェント API は新しいオペレーティング環境への移行が非常に簡単です。純粋な Java 実装を使用して記述されたアプリケーションは、個別のオペレーティング システムではなく、実装をホストする Java 仮想マシンに対してのみ証明書を必要とします。

ポリシー管理 API

以下の SiteMinder 要素を管理するためにポリシー管理 API を使用します。

- リソース
- ポリシー
- キャッシュ
- セキュリティ ロール

ポリシー ドメイン、レルム、およびポリシーなどのポリシー オブジェクトを作成、削除、関連付け、修正することでポリシーを管理できます。

認証 API

認証 API は、標準的な SiteMinder 認証方式で提供されない認証サービスを実装するカスタム認証方式を作成するために使用します。

許可 API

許可 API は保護されたリソースへのアクセスを制御するカスタム機能を実装するために使用します。 機能は、ポリシー サーバのアクティブな式で参照されるカスタム Java クラスによって提供されます。 アクティブな式は、以下のポリシー サーバ オブジェクトに表示される、一連の変数定義です。

- アクティブ ポリシー
- アクティブ レスポンス
- アクティブ ルール

分散代行管理サービス API

分散代行管理サービス（DMS） API は以下のようなタスクを実行するために使用します。

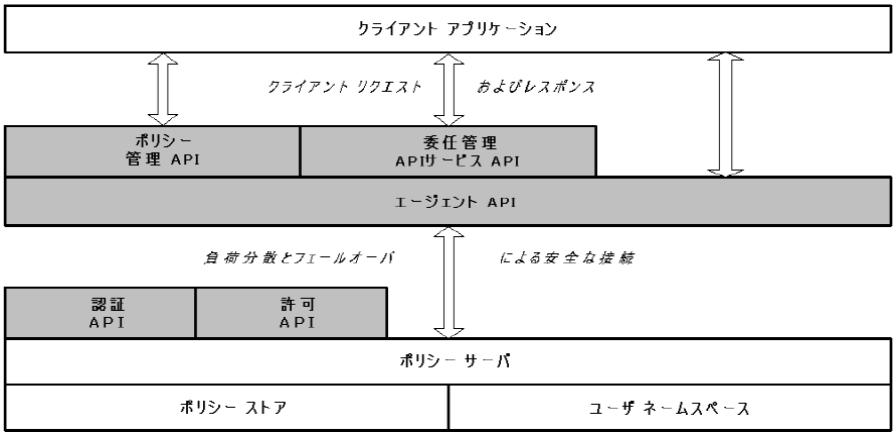
- ディレクトリ エントリの管理
- ユーザおよびグループへの権限の許可
- ユーザおよびグループへの DMS ロールの許可

ユーティリティ パッケージ

ポリシー管理 API および DMS API で API を実装するためにユーティリティ パッケージのメソッドを使用します。

Java コンポーネントが共に適合する方法

以下の図は、SiteMinder SDK の Java コンポーネントが共に適合する方法を示しています。

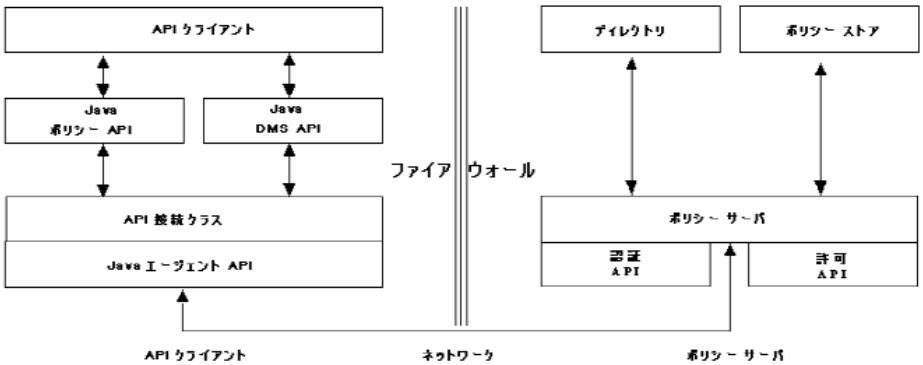


ネットワーク アーキテクチャ

Java API は、リモート SiteMinder ポリシー サーバに接続するクライアント アプリケーションを記述するのに使用できます。 これらのアプリケーションは、Java エージェント API の以下のビルトイン機能にアクセスできます。

- セキュリティ
- 負荷分散
- フェールオーバ

以下の図は、Java API のネットワーク アーキテクチャを示しています。



ポリシー管理 API および DMS API は、ポリシー サーバへのアクセスに Java エージェント API を使用します。単一の API クライアント インスタンスは、SiteMinder ポリシー サーバへの単一のセキュアなエージェント API 接続を作成します。同じプロセス空間を共有する限り、複数の API クライアントは単一のエージェント API 接続を利用できます。たとえば、接続の確立に Java エージェント API を使用して、次にその接続を使用して DMS API の呼び出しを作成できます。

Java API フロー

Policy Management API または DMS API でクライアント アプリケーションを作成する場合に必要な手順を以下に示します。

1. ポリシー サーバへの接続を確立します。
2. セッションを取得します。
3. API 要求を作成します。
4. 結果と例外を処理します。

ポリシー サーバへの接続の確立

ポリシー サーバへの接続を確立するには、ユーティリティ パッケージの SmApiConnection クラスを使用します。 このクラスは、Java API 要求が送信されるエージェント API ハンドルを保持します。

このクラスには 2 つのタイプの接続ハンドルがあります。

- *デフォルト*接続ハンドル。デフォルト接続ハンドルには以下の特徴があります。
  - エージェント API オブジェクトの単一のインスタンスを表します。
  - プロセスを通してスタティックです。
  - ポリシー管理 および DMS クライアントの両方からの エージェント API オブジェクトへの接続を許可します。

  単一のエージェント API オブジェクト インスタンスを通してポリシー サーバへの複数の接続を確立できます。
- *ユーザ定義*の接続ハンドル。複数のユーザ定義接続オブジェクトを作成できます。各オブジェクトはポリシー サーバへの複数の接続をサポートできます。

デフォルト接続の確立

ポリシー サーバへの接続をすでに確立していない場合、自動的な接続を要求できます。SiteMinder が接続を自動的に確立する場合、デフォルトの Java エージェント API オブジェクトおよびハンドルを作成します。ただし、ユーザ定義の有効なハンドルがすでに存在する場合、SiteMinder はデフォルト オブジェクトおよびハンドルを作成しません。 ユーザ定義のハンドルはデフォルトのハンドルに優先されます。

ポリシー サーバへのデフォルト接続を自動的に確立する方法

1. 以下のコンストラクタを使用して API 接続オブジェクトを作成します。

```
SmApiConnection (boolean bDefaultAgentConnection
                                boolean disableLoadBalancing)
```

2. コンストラクタで、bDefaultAgentConnection を true に設定します。たとえば、以下のようにします。

```
SmApiConnection m_defaultConnection =
                                new SmApiConnection(true,false);
```

3. bDefaultAgentConnection が false である場合、クライアント コード内で接続を明確に確立する必要があります。

自動接続には以下の要件があります。

- Web エージェントは、同じマシンのエージェント API を実行している場所にインストールされます。
- Web エージェント設定オブジェクト内のプロパティ DefaultAgentName にはエージェント名が含まれます。ポリシー サーバで Web エージェント設定オブジェクトを定義します。
- Apache の Web エージェントでは、エージェント設定ファイルへのパスは CLASSPATH にあります。 Microsoft IIS Web エージェントでは、この設定情報はレジストリにあるため、CLASSPATH 参照は必要ではありません。

ユーザ定義の接続を確立します

2 つの方法のいずれかを使用してユーザ定義の接続を確立します。

- SmApiConnection オブジェクトのコンストラクタ内の既存の Java エージェント API 接続ハンドルを参照する。
- setAgentApiConnection() メソッドを使用して新規接続を手動で確立する。

注: すでにポリシー サーバへの接続を確立している場合、後続の Policy Management API または DMS API コールを作成するためにその接続を使用できます。

既存のエージェント API 接続を使用して接続を作成する方法

1. 以下のコンストラクタを通して接続オブジェクトを作成します。

```
SmApiConnection (netegrity.siteminder.javaagent.AgentAPI
                                        agentApiConnection)
```

2. コンストラクタで、既存のエージェント API 接続のハンドルを渡すには agentApiConnection を使用します。たとえば、以下のようにします。

```
SmApiConnection myConnection =
                    new SmApiConnection (myAgentApiConnection);
```

新しい Java エージェント API ハンドルはユーザ定義のハンドルです。

デフォルト接続をまだ実行しておらず、ユーザ定義の接続オブジェクトが必要な場合、エージェント API を使用してエージェント オブジェクトを作成し、以下のように新規接続を作成できます。

1. エージェント オブジェクトを作成します。

   以下のソースのいずれかからの接続パラメータに基づくエージェント オブジェクトを作成できます。

   - たとえば以下のような、コード内で定義されたユーザ定義の接続パラメータ。

```
AgentAPI agent = new AgentAPI();
ServerDef sd = new ServerDef();
sd.serverIpAddress = POLICY_IP;
sd.connectionMin = CX_MIN;
sd.connectionMax = CX_MAX;
sd.connectionStep = CX_STEP;
sd.timeout = CX_TIMEOUT;
sd.authorizationPort = AZ_PORT;
sd.authenticationPort = AUTH_PORT;
sd.accountingPort = ACC_PORT;
InitDef init=new InitDef(AGENT_LOGIN,SHARED_SECRET,false, sd);
agent.init(init);
```

   注: SiteMinder v6.0 以降では、許可サーバ、認証サーバ、および会計サーバは単一のサーバ プロセスに統合されます。 したがって、*authorizationPort*、*authenticationPort* および *accountingPort* はすべて同じ値に設定できます。

   - 接続パラメータは、エージェント設定ファイルに格納されます。

   注: SiteMinder v4.x webagent.conf ファイルは、SM API によってサポートされなくなりました。

2. 新しい接続を作成します。

   エージェント オブジェクトを作成した後に、これらの方法のいずれかを使用して新しい接続を作成します。

   - 作成したばかりのエージェント オブジェクトを新しい SmApiConnection オブジェクトのコンストラクタに渡します。たとえば以下のようにします。

```
SmApiConnection myConnection = new SmApiConnection(agent);
```

   - setAgentApiConnection() を呼び出し、作成したばかりのエージェント オブジェクトを渡します。たとえば以下のようにします。

```
SmApiConnection myConnection=new SmApiConnection(false,false);
myConnection.setAgentApiConnection(agent);
```

   このように接続を確立する場合、Java エージェント API ハンドルはユーザ定義のハンドルです。

   setAgentApiConnection() を呼び出し、接続がない場合、Null を渡すことで自動的にスタティックな接続を確立できます。

セッションの取得

ポリシー サーバへの接続を取得した後、ユーザまたは管理者のセッションを取得します。

注: ポリシー管理 API を使用するには、SiteMinder 管理者として接続する必要があります。

セッション オブジェクトを取得した後、SmPolicyApiImpl クラスまたは SmDmsApiImpl クラスに対するコンストラクタを通してポリシー管理 API または DMS API に渡します。

セッションを取得するには、以下のアクションのいずれかを実行します。

状況	以下を実行	結果
ユーザの認証からの既存のセッションがある。	—	認証ユーザに対するセッション指定を渡します。
既存のセッションがありません。	SiteMinder 管理者として接続する必要があります。	メソッド SmApiSession.login() を使用します。

状況	以下を実行	結果
既存のセッションがありません。	管理者ではないユーザとして接続します。	ユーザのためのセッション指定を取得するために Java エージェント API を使用します。

認証されたユーザ用のセッション指定を取得済みの場合、そのセッション指定を使用できます。 新しいセッション指定を取得する必要はありません。

既存のセッションを使用するには、SmApiSession オブジェクトを作成し、そのオブジェクトとセッション指定を関連付けます。

SiteMinder 管理者としてのログイン

SiteMinder 管理者を認証するには、ユーティリティ パッケージの SmApiSession クラスの login() メソッドを使用します。このメソッドは、管理者を認証するために管理者のログイン認証情報（ユーザ名とパスワード）を使用します。 この login() メソッドを呼び出すと、セッション指定が取得され、SmApiResult オブジェクトが返されます。

login() メソッドの構文を以下に示します。

```
result=mySession.login (username,
                        password,
                        IPaddress,
                        challengeReason);
```

challengeReason パラメータに対する値を以下のように提供します。

- 管理者の初期ログインで、challengeReason を 0 （理由なし）に設定します。
- 初期ログインが失敗する場合は、次回の login() 呼び出しで challengeReason を使用して前回の認証試行の結果を指定します。

  challengeReason に割り当てる理由値を取得するには、SmApiResult オブジェクトの getReason() を呼び出します。

ユーザに対する新しいセッション指定を取得するには、Java エージェント API を使用してセッション指定を取得します。次に、SmApiSession オブジェクトを作成し、そのオブジェクトとセッション指定を関連付けます。

エージェント ディスカバリ

SiteMinder 管理者は、エージェント ディスカバリを使用して、数年にわたって展開されているエージェントを含め、さまざまなタイプのエージェントのインスタンスを追跡できます。 エージェント インスタンスは、任意のタイプのエージェント（たとえば Web エージェント、カスタム エージェント、ERP エージェント）が可能です。 エージェント ディスカバリの認識範囲に含めるためには、エージェントがアクティブで、ポリシー サーバと通信中であることが必要です。

5.x 以降のエージェントのみ追跡できます。 r12.5 以前に作成されたエージェントの場合、エージェントの識別に IP アドレスとトラステッド ホストの組み合わせが使用されます。 同じエージェントでも、この組み合わせが変わると、1 つエージェントに対して複数のエントリが生じることになります。

各 r12.5 エージェント インスタンスは固有の GUID により識別されます。 GUID は設定ファイルに保存されています。 複数のエージェント インスタンスで 1 つの設定ファイルを共有することはできません。設定ファイルの場所に加えて、AgentInstanceDef.java は、エージェント インスタンスの以下の属性を指定するパラメータを定義します。

- エージェント製品タイプ
- エージェント製品バージョン
- エージェント製品サブタイプ
- エージェント設定オブジェクト名
- ホスト設定オブジェクト名

エージェント ディスカバリの有効化

カスタム エージェントをエージェント ディスカバリの範囲内にしたい場合、以下のプロセスに従います。

1. AgentInstanceDef.java クラスをインスタンス化します。
2. getAgentIdFile メソッドを呼び出します。

   このメソッドが有効な設定パスを返す場合、エージェント インスタンスはエージェント ディスカバリ プロセスですでに処理されています。
3. getAgentIdFile が設定ファイルを返さない場合、setAgentIdFile メソッドを呼び出し、設定ファイルの場所を提供します。
4. （オプション）追加のメソッドを呼び出してエージェント インスタンスに対する属性情報を設定または取得します。
5. オブジェクトの名前を渡して AgentAPI.setAgentInstanceInfo メソッドを呼び出します。

エージェント インスタンスは、エージェント インスタンスのアクティブ状態が継続していることをポリシー サーバに伝えるハートビート メッセージをポリシー サーバに定期的に送信します。

API 要求の作成および結果の処理

セッションを確立した後、クライアント アプリケーション内でメソッドを呼び出すことができます。

*結果*はポリシー サーバから Java API 要求へのレスポンスです。 結果は SmApiResult オブジェクトで返されます。

*例外*は予期しないクライアント側エラーからスローされます。 例外には、原因および重大度などの追加情報を持つ結果が含まれます。 API 要求の結果を保存する結果オブジェクトを作成するには、ユーティリティ パッケージの SmApiResult クラスのコンストラクタを使用します。たとえば、以下のようにします。

```
SmApiResult result = new SmApiResult();
```

結果オブジェクト上でメソッド isSuccess() を呼び出すことで要求が成功したかどうかを確認できます。 メソッドは、要求が成功した場合は true、失敗した場合は false を返します。

equals() メソッドを呼び出すことで、現在の結果オブジェクトと特定の結果オブジェクトを比較することができます。

equals() メソッドを使用して、現在の結果オブジェクトを、異なる種類の結果を表す SmApiResult 定数と比較できます。 たとえば、以下のコードでは、一意の定数 SERVER_INVALID_PASSWORD によって表される結果は現在の結果オブジェクトと比較されます。

```
InetAddress address = InetAddress.getLocalHost();
SmApiResult result = apiSession.login(usr,pwd,address,0);
boolean resultStatus =
            result.equals(SmApiResult.SERVER_INVALID_PASSWORD);
```

トレース情報のログ記録

クライアント側でトレース情報をログ記録するには、Java ツールの -D オプションを使用し、システム プロパティ SMJAVASDK_LOG_INFO を true に設定します。 SiteMinder は、標準出力への情報をログ記録します。

たとえば、ユーザの Java 開発キットが Windows 上にあり、ポリシー管理 API サンプル アプリケーションをトレースしたい場合、コマンド ラインは以下のようになります。

```
java -DSMJAVASDK_LOG_INFO=true -classpath .;..¥..¥java¥smjavasdk2.jar;
..¥..¥java¥smjavaagentapi.jar PolicyApiSample
```

Javadoc リファレンス

特定のクラスまたはメソッドについて知るために Javadoc を使用します。通常、これらの詳細には構文、パラメータ、戻り値および例外情報が含まれます。

Javadoc リファレンスの各パッケージ、クラスおよびインターフェースの説明には、コンポーネントが導入されたときの SiteMinder または SDK バージョンを示す Since 見出しが含まれる場合があります。 個別のメソッドおよびフィールドには、クラスまたはインターフェースの後のバージョンに追加された場合にのみ Since 見出しが含まれます。

カスタム コードのサポート

CA は標準的な提供物の一部としてソフトウェア開発キット（SDK）をサポートします。 ただし、カスタマやパートナーによって記述されたコードはサポートされていません。 ユーザが自分の記述するコードに責任を持ちます。 SDK ベースのコードの設計や実装にサポートが必要な場合は、CA カスタマ アカウント チームにご連絡ください。

第 2 章: Java ユーティリティ パッケージ

このセクションには、以下のトピックが含まれています。

ユーティリティ パッケージの目的

ポリシー管理 API または DMS API 内の関数を呼び出す場合、Java アプリケーションの構築にユーティリティ パッケージ内の関数を使用する必要があります。

ユーティリティ パッケージ内の SmApiConnection、SmApiResult、SmApiSession、SmApiException、および SmProperty クラスは、以下のようなサービスを提供します。

- ポリシー サーバへの接続の確立
- セッションの取得
- API 要求の結果を保存する結果オブジェクトの提供
- 例外と結果の処理
- プロパティ データのカプセル化

内部使用クラス

ユーティリティ パッケージは、以下のクラスを SiteMinder 内部使用のみを目的として提供します。

- SmApiConstants
- SmApiObject
- SmApiPropertySets

- SmApiExportFileHandler
- SmApiImportFileHandler
- SmFlag

Connection クラス

API 接続オブジェクトを作成し、エージェント API およびポリシー サーバ間の接続を確立するために SmApiConnection クラスを使用します。使用するコンストラクタに応じて、デフォルト接続またはユーザ定義の接続のいずれかを確立できます。

SmApiConnection クラスの中心となるメソッドは以下のとおりです。

メソッド	説明
getAgentApiConnection()	現在の接続用の エージェント API ハンドルを取得します。後続の Java API 要求をポリシー サーバに発行する場合にこのハンドルを使用します。
isValidApiConnection()	有効な エージェント API 接続が利用可能かどうかを確認します。
setAgentApiConnection()	メソッドへ渡されたハンドルを介してユーザ定義の接続を確立します。NULL が渡される場合、スタティックな接続が確立されます。

注: クライアント アプリケーションから execute() メソッドを呼び出さないでください。 このメソッドは内部使用専用です。

Session クラス

Session クラスである SmApiSession は、ユーザに有効な API 接続、およびユーザが使用するコンストラクタによってはセッション指定を渡すことで、ユーザがセッション オブジェクトを作成できるようにします （セッション指定はセッション チケットとしても知られています）。

SmApiSession クラスの中心となるメソッドは以下のとおりです。

メソッド	説明
getApiConnection()	現在の接続用の SmApiConnection オブジェクトを取得します。
getSessionSpec()	現在のセッションに対する指定を取得します。
login()	SiteMinder 管理者をログインします。ポリシー サーバは、セッションのセッション指定を発行します。
logout()	SiteMinder 管理者をログアウトします。
setApiConnection()	有効な API 接続を設定します。
setSessionSpec()	既存のセッション指定を設定します。

注: エンド ユーザ、DMS 組織管理者、および DMS スーパー管理者のログインおよびログアウトに関しては、エージェント API パッケージの AgentAPI.login() および AgentAPI.logout() メソッドを使用します。

Result クラス

Result クラス、SmApiResult は、SiteMinder Java API 要求の結果を保存します。 SiteMinder の結果には以下の要素が含まれます。

- ファシリティ。 結果の発生元。 たとえば、結果はクライアントまたはサーバ上で発生する場合があります。
- 重大度。 情報または警告のような結果の重大度。
- ステータス。 結果のステータス コード。 ステータス コードは各ファシリティ内で一意です。ファシリティの一意のステータス コードを使用して、特定の結果を同じファシリティで発生した他の結果と識別することができます。
- メッセージ。 説明文または数値の詳細など結果に関する追加の情報。

結果には理由コードも含まれる場合があります。たとえば、パスワード ポリシー結果には、パスワードが必須最小文字数を満たしていないことを示す理由コード 1001 が含まれる場合があります。 結果用の理由コードを検索するには、getReason() を呼び出します。

サーバ側のエラーがすべて例外ではなく結果として返されます。 ただし、クライアント側の例外がスローされる場合、SmApiResult オブジェクトは例外に埋め込まれています。

結果オブジェクトの解釈

各結果オブジェクトおよびそのファシリティ/重大度/ステータスの組み合わせは一意の値によって表されます。 これらの一意の値は、たとえば SERVER_CONFIGURATION_FAILURE のように、SmApiResult クラスで定義された事前定義済み定数と関連付けられます。

結果に対するファシリティ/重大度/ステータス情報を判断するには、equals() メソッドを呼び出して SmApiResult を結果定数と比較することができます。

- 機能： FACILITY_CONNECTION
- 重大度： SEVERITY_ERROR
- ステータス： 4
- メッセージ： サーバ設定を取得できません （Unable to get server configuration）

結果オブジェクトを文字列として出力できます。たとえば、SmApiResult オブジェクト上で toString() を呼び出すことで、結果文字列を生成できます。

結果文字列にはスペースで区切られた 5 つの名前/値ペアが以下の形式で含まれています。

```
[facility=facility severity=severity reason=reason
status=statusCode message=message]
```

たとえば、ユーザがパスワードの英数字数の下限より少ない文字数でパスワードを作成しようとした場合に発生する SmApiResult オブジェクトに対して toString() を呼び出したとします。 メソッドは、以下のように見える結果文字列を返す場合があります。

```
[facility=4 severity=3 reason=1008 status=13 message=nArg=1,Arg1=3]
```

結果内のフィールドには以下の意味があります。

- facility=4。 結果はサーバで発生しました。
- severity=3。 結果はエラーです。
- reason=1008。要求されたパスワードの文字数がパスワードに必要な英数字の最小数より少ないため、エラーが発生しました。
- status=13。 このファシリティに対する一意の結果ステータス コード。
- message=nArg=1,Arg1=3 結果の追加の説明。このフィールドの 2 つの部分には以下の意味があります。
  - nArg1=1。 エラーには単に 1 つのエラー説明が含まれます。
  - Arg1=3。 エラー説明は 3 です。理由コード 1008 のコンテキストでは、Arg1 値はパスワードに少なくとも 3 つの英数字が含まれる必要があることを示しています。

Result クラスの中心的なメソッド

SmApiResult クラスには、以下のような中心的なメソッドがあります。

メソッド	説明
equals()	現在のオブジェクトがメソッドに渡されたオブジェクトと等しいかどうかを示します。
getError()	一意のエラー コードを取得します。
getFacility()	エラーと関連付けられたファシリティ コードを取得します。
getMessage()	エラーと関連付けられたメッセージを取得します。
getReason()	エラーの理由コードを取得します。
getSeverity()	エラーと関連付けられた重大度コードを取得します。
getStatus()	現在のファシリティ内のステータス コードを取得します。 このメソッドはサーバからのパラメータを結果コードとして使用することができます。
isSuccess()	要求が成功したかどうかをレポートします。
toString()	SmApiResult オブジェクトの文字列の表現を返します。

Exception クラス

Exception クラス、SmApiException には Result クラス SmApiResult が含まれます。 以下のパッケージが SmApiException を使用します。

- com.netegrity.sdk.policyapi
- com.netegrity.sdk.dmsapi
- com.netegrity.sdk.apiutil

SmApiException クラスには、以下のような中心的なメソッドがあります。

メソッド	説明
getFacility()	例外のファシリティ コードを取得します。
getReason()	例外の理由コードを取得します。
getSeverity()	例外の重大度コードを取得します。
getStatus()	例外のステータス コードを取得します。
toString()	SmApiResult オブジェクトの文字列の表現を返します。

Exception クラスは java.lang.Exception を拡張します。 継承された getMessage() メソッドを呼び出すことで、例外と関連付けられたメッセージを取得することができます。

Property クラス

Property クラス、SmProperty は、プロパティに関する以下の情報を保持します。

- 名前
- 値
- タイプ（暗号化/プレーン）

SmProperty クラスには、以下のような中心的なメソッドがあります。

メソッド	説明
getName()	プロパティの名前を取得します。

メソッド	説明
getType()	プロパティのタイプ（つまり、プレーン テキストの場合は 0、暗号化される場合は 1）を取得します。
getValue()	プロパティの値を取得します。
setName()	プロパティの名前を設定します。
setType()	プロパティのタイプを設定します。プレーン テキストの場合は 0、暗号化される場合は 1 になります。
setValue()	プロパティの値を設定します。

第 3 章: C 用のエージェント API ガイド

このセクションには、以下のトピックが含まれています。

SiteMinder エージェント

SiteMinder エージェントはエージェント API のクライアントです。 エージェントは、ポリシー サーバによって処理されるアクセス制御ポリシーを適用します。 ポリシー サーバは、リソースに対して特定の知識を持たない汎用ポリシー エンジンです。 リソースについての特定の知識は SiteMinder エージェントによって提供されます。 エージェントは、リソースの解釈を定義し、許可のないユーザからリソースを保護するゲートキーパーとして機能します。

各種のリソースを保護するためにさまざまなエージェント タイプが用意されています。一部のエージェント タイプは、SiteMinder 製品の一部として出荷される事前定義された標準エージェントです。たとえば、Web サーバに HTTP アクセス制御を提供する Web エージェントなどです。 ただし、カスタム エージェントを実装するために エージェント API を使用することもできます。

エージェント API を使用すると、さまざまなコンテキスト固有の方法でユーザを認証および許可できるカスタム エージェントを作成できます。たとえば、以下を実行するエージェントを FTP 転送に対して作成できます。

- 基本名およびパスワード認証情報の代わりに証明書ベースの認証を実装します。
- 個別のユーザの許可レベルに基づいてアップロードおよびダウンロードを許可します。

カスタム エージェントは、標準 SiteMinder Web エージェントを持つシングル サインオン環境に追加することができます。

エージェント タイプ

エージェント タイプは、エージェントの動作を定義します。カスタム エージェントを開発した後で、ポリシー サーバ ユーザ インターフェース内のエージェントに対して新規エージェント タイプを設定する必要があります。たとえば、カスタム FTP エージェントを開発した場合、ポリシー サーバ ユーザ インターフェース内の FTP エージェントに対してエージェント タイプを設定する必要があります。

注: カスタム エージェントに対するエージェント タイプの設定に関する詳細は、「SiteMinder C 言語用プログラミング ガイド」を参照してください。

エージェント API クラス階層

Java エージェント API への主要なアクセス ポイントは AgentAPI クラスです。AgentAPI クラスによって必要とされるデータを保持するための他のクラスがいくつか提供されています。

- 属性
- AttributeList
- BinaryBuffer
- InitDef
- ManagementContextDef
- RealmDef

- ResourceContextDef
- ServerDef
- SessionDef
- TokenDescriptor
- TunnelServiceRequest
- UserCredentials

JNI Java エージェント API の実装

JNI Java AgentAPI を直接または（別のエージェントを通して）間接的に 使用して構築されるアプリケーションは、以下を含めた基盤となる実装詳細から分離されます。

- LDAP ディレクトリ、SQL データベースまたは NT ドメインなどのユーザ ネームスペース
- ユーザ名/パスワードと同じくらい単純か、PKI システムと同じくらい複雑な認証方式
- グループ メンバシップまたは個別のプロファイル データに基づく許可

Java エージェント API によって提供される追加の利点には、完全なセッション管理サポート、自動暗号化キー ロールオーバー、およびリアルタイムのポリシー更新が含まれます。

JNI Java エージェント API を実装する方法

1. 付属のリリース ノートに示される必須ソフトウェアを確認します。
2. サンプル コードを確認します。
3. クライアント アプリケーション用のソース コードを記述します。

4. Java 仮想マシン（JVM）が呼び出される場合にシステムが JNI サポートライブラリを見つけることができることを以下のように確認します。

   - Windows：以下が含まれるように PATH を変更し、smjavaagentapi.dll、smerrlog.dll、および smcommonutil.dll が検出されるようにします。

   <install_path>¥sdk¥bin

   - Solaris： 以下が含まれるように LD_LIBRARY_PATH を変更し、libsmjavaagentapi.so、libsmerrlog.so、および libsmcommonutil.so が検出されるようにします。

   <install_path>/sdk/bin

   - AIX：以下が含まれるように LIBPATH を変更し、libsmjavaagentapi.so、libsmerrlog.so、および libsmcommonutil.so が検出されるようにします。

   <install_path>/sdk/bin

   - Linux： 以下が含まれるように LD_LIBRARY_PATH を変更し、libsmjavaagentapi.so、libsmerrlog.so、および libsmcommonutil.so が検出されるようにします。

   <install_path>/sdk/bin

   - HP-UX 11： 以下が含まれるように SHLIB_PATH を変更し、libsmjavaagentapi.so、libsmerrlog.so、および libsmcommontuil.so が検出されるようにします。

   <install_path>/sdk/bin

   注: Java エージェント API は HP10 では利用できません。

5. Java エージェント API を使用するエージェントをコンパイルまたは実行する場合に SiteMinder が JNI Java エージェント API JAR ファイルを検出できることを確認してください。 JAR ファイル smjavaagentapi.jar は以下の場所に保存されます。

   - Windows プラットフォーム：

   <install_path>¥sdk¥java

   - UNIX プラットフォーム：

   <install_path>/sdk/java

   CLASSPATH 設定に smjavaagentapi.jar を追加します。 コンパイルの際に、-classpath スイッチを使用できます。

6. javac を使用して Java エージェント API アプリケーションをコンパイルします。

   例については、サンプル ディレクトリ smjavaagentapi の java-build.bat または java-build.sh を参照してください。

7. Java エージェント API アプリケーションを使用するためにポリシー サーバを設定します。

8. アプリケーションを実行します。

   例については、サンプル ディレクトリ smjavaagentapi の java-run.bat または java-run.sh を参照してください。

純粋な Java エージェント API の実装

純粋な Java エージェント API を直接または（別のエージェントを通して）間接的に 使用して構築されるアプリケーションは、以下を含めた基盤となる実装詳細から分離されます。

- LDAP ディレクトリ、SQL データベースまたは NT ドメインなどのユーザ ネームスペース
- ユーザ名/パスワードと同じくらい単純か、PKI システムと同じくらい複雑な認証方式
- グループ メンバシップまたは個別のプロファイル データに基づく許可

Java エージェント API によって提供される追加の利点には、完全なセッション管理サポート、自動暗号化キー ロールオーバー、およびリアルタイムのポリシー更新が含まれます。

純粋な Java エージェント API を実装する方法

1. 付属のリリース ノートに示される必須ソフトウェアを確認します。
2. サンプル コードを確認します。
3. クライアント アプリケーション用のソース コードを記述します。

4. Java エージェント API を使用するエージェントをコンパイルまたは実行する場合に SiteMinder が純粋な Java エージェント API の .jar ファイルを検出できることを確認してください。 JAR ファイル smagentapi.jar は以下の場所に保存されます。

   - Windows プラットフォーム：

   <install_path>¥sdk¥java

   - UNIX プラットフォーム：

   <install_path>/sdk/java

   CLASSPATH 設定に smagentapi.jar、crypto.jar、cryptoFIPS.jar を追加します。 コンパイルの際に、-classpath スイッチを使用できます。

5. javac を使用して Java エージェント API アプリケーションをコンパイルします。

   例については、サンプル ディレクトリ smjavaagentapi の java-build.bat または java-build.sh を参照してください。

6. Java エージェント API アプリケーションを使用するためにポリシー サーバを設定します。
7. アプリケーションを実行します。

純粋な Java エージェント API 使用状況

下位互換性

純粋な Java エージェント API は、JNI Java エージェント API とのバイナリおよびソース互換性を維持します。 純粋な Java エージェント API は、SiteMinder ポリシー管理 API、および SiteMinder DMS API を含む SiteMinder ポリシー サーバへの接続にエージェント API に依存する他のすべての SiteMinder Java SDK インターフェースをサポートし、これらのインターフェースの移行性を拡張します。

設定制限

純粋な Java エージェント API は、SiteMinder アプリケーション サーバ エージェントまたは SiteMinder SDK で開発されたエージェントのいずれに対しても設定を変更しません。純粋な Java エージェント API の設定は、以下の例外を除いて JNI Java エージェント API の設定と同じです。

- JNI Java エージェント API から 純粋な Java エージェント API への UNIX エージェントの移行には、SiteMinder ポリシー サーバへのトラステッド ホスト エンティティの再登録が必要です。これは、JNI Java API 内の共有秘密キーが純粋な Java 実装とは異なって計算されるためです。
- Unix および Windows システム（ネイティブ コード エージェント API でのファイル ロッキングに互換性がないため）の両方で、SmHost.conf ファイルは C/C++、JNI Java エージェント API、または純粋な Java エージェント API を使用するエージェントの間で共有することはできません。 したがって、ブートストラップ設定ファイルの個別のコピーを純粋な Java エージェント API エージェント用に維持する必要があります。
- 5.x- タイプのカスタムの純粋な Java エージェントに対するホストを登録するには、smreghost.exe ではなく smreghost.bat （または UINX では smreghost.sh）を使用する必要があります。
- Unix システム上の JNI Java エージェント API からのアップグレードでは、4.x encryptkey ツールで暗号化される共有秘密キーを使用するカスタムの 4.x ベースのエージェントのユーザは、アップグレードされるエージェントのエージェント側で共有秘密キーを更新する必要があります。

純粋な Java エージェント API トレースの有効化

純粋な Java エージェント API は、コンソールに出力される詳細なトレース メッセージをサポートします。 smreghost などエージェント API を使用するコマンド ライン ツールを実行する場合にこれらのメッセージは役立ちます。

トレース メッセージを有効にするには、enableDebug という名前のシステム プロパティを true に設定します。 コマンド ラインから、-Dcom.ca.siteminder.sdk.agentapi.enableDebug="true" を追加します。 例：

```
>SM_SMREGHOST_CLASSPATH="c:¥ca¥sdk¥java¥smagentapi.jar;c:¥ca¥sdk¥java¥cryptoj.jar"
>java -Dcom.ca.siteminder.sdk.agentapi.enableDebug="true" -classpath %SM_SMREGHOST_CLASSPATH%
com.ca.siteminder.sdk.agentapi.SmRegHost -i 127.0.0.1 -hc host_conf1 -hn trustedhost3 -u siteminder
-p firewall
```

ポリシー サーバへの接続

エージェントは、ユーザに代わって作業を実行できるようになる前に、init() メソッドを発行して 1 つ以上のポリシー サーバとの接続を開始する必要があります。 InitDef パラメータを通して、フェールオーバ モードや接続プール サイズなどの接続パラメータを指定できます。この手順は TCP 接続を作成し、通常エージェント インスタンスごとに複数回実行する必要はありません。

エージェント API が初期化された後、API の呼び出しはすべて初期化された API インスタンスに対して完全にスレッド セーフです。

複数の API インスタンスを初期化できます（個別のポリシー ストアを複数使用するポリシー サーバと連係して操作する場合など）。

初期化直後に、エージェントは、ManagementContextDef オブジェクトに定数 MANAGEMENT_SET_AGENT_INFO が設定された doManagement() を呼び出すことで、バージョン情報をポリシー サーバに通知します。 実際の情報は、ビルド番号やバージョン番号などのエージェントに関する十分な情報が含まれる任意の文字列が可能です。 文字列はポリシー サーバ ログに記録されます。

エージェント API が初期化された後、エージェントは役立つ作業を実行できます。 この時点で、URL への GET 要求の受信など、ユーザからの要求の受け入れを開始できます。

リソースへのユーザ アクセス

エージェントは要求されたリソースにユーザ アクセスを付与する前に以下の手順に従う必要があります。 エージェントのパフォーマンスを向上するために、ほとんどの手順は結果をキャッシュできるようになっています。 エージェントは、キャッシュを望むだけ実行することができます。

1. ユーザの要求を受け入れます。

   リソースにアクセスするためのユーザの要求を受理します。 これはアプリケーション固有の要求です。たとえば、Web エージェントは、URL に対する GET 要求を受け入れます。

2. リソースが保護されているかどうかを確認します。

   要求されたリソースが保護されているかどうかを確認するには isProtected() を呼び出します。

   リソースが保護されている場合、ポリシー サーバはユーザの身元を検証するためにユーザから取得される必要がある必要な認証情報を返します。 リソースが保護されていない場合、要求されたリソースへのアクセスが許可されます。

   この手順の結果はキャッシュできます。

3. ユーザを認証します。

   ユーザから必要な認証情報を収集し、かつユーザを認証するには login() を呼び出します。

   認証が成功したら、ポリシー サーバはセッションを作成し、一意のセッション ID およびセッション指定を含むレスポンス属性を返します。 これらのレスポンス属性は、ポリシーによって駆動され、ユーザ プロファイル データ、スタティックまたはダイナミックの権限、多くの事前定義された認証状態属性、またはポリシー管理者によって指定された他のデータです。

   エージェントは、ユーザ セッション情報をキャッシュし、セッション有効期限を追跡することにより、セッション管理を実行できるようになります。

4. ユーザが許可されているかどうか確認します。

   authorize() を呼び出して、要求されたリソースにユーザがアクセスできるかどうか検証します。

   許可に成功したら、ポリシー サーバはリソース固有の権限を含めたレスポンス属性を返します。 これらのレスポンス属性は、ポリシーによって駆動され、ユーザ プロファイル データ、スタティックまたはダイナミックの権限、またはポリシー管理者によって指定された他のデータです。

   この時点では、要求されたリソースに関するユーザの許可情報は知られており、将来の要求の処理速度を向上するためにキャッシュできます。

5. キャッシュされた許可情報を監査します。

   認証および許可手順のいずれもユーザ、保護されたリソース、およびエージェントに関する関連情報をログ記録します。 ただし、エージェントがそのキャッシュからの許可を実行する場合、トランザクションは、今までどおり audit() メソッドを通してログ記録できます。

6. リソースへのアクセスを許可します。

   ユーザの識別情報が知られ、許可が検証され、必要な権限が取得されたため、許可されたユーザにリソースへのアクセス権を付与します。

7. 管理要求を発行します。

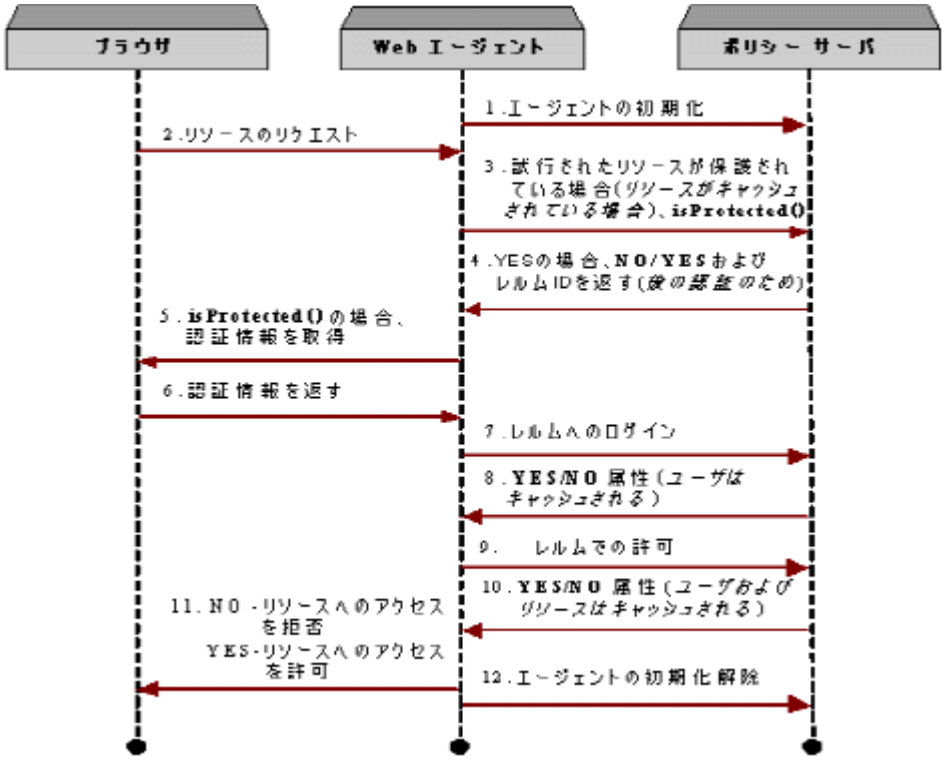
   これは、ポリシー サーバで更新コマンドをポーリングするために必要に応じて実行される手順です。 コマンドに応じて、エージェントは暗号化キーの更新またはキャッシュのクリアのいずれかまたは両方を実行します。

エージェントが必要なくなった場合に、各 API インスタンスに対して unInit() メソッドを発行します。 これにより、すべてのポリシー サーバへの TCP 接続が閉じられます。

注: エージェント API は、セッション検証を適用するような方法でのキャッシュに対してファシリティを提供しません。 ユーザ セッションとリソース固有の権限のいずれかまたは両方をキャッシュすることを選択することで、各ユーザの要求の間に自身のセッション管理を実行する必要がエージェントに発生します。 これは、エージェント上でのキャッシュによって、セッション検証およびリソース許可のいずれかまたは両方の実行のために SiteMinder ポリシー サーバにコンタクトする必要がなくなるためです。

Web エージェントがエージェント API を使用する方法

以下の図は、Web エージェントが エージェント API を使用するときに発生するプロセス フローを示しています。



Java エージェント API サービス

Java エージェント API は、高度かつ安全で、堅牢なエージェントを開発できる、充実した一連のサービスを提供します。 エージェント構築の一環で、以下のサービスを使用します。

- セッション サービス
- 許可サービス
- 監査サービスおよびトランザクション トラッキング
- 管理サービス（キーの暗号化、キャッシュの更新）
- トンネル サービス

これらのサービスは AgentAPI クラスを通してアクセスされます。

セッション サービス

セッショニングは、多層アプリケーション環境にわたって一貫したユーザ セッションを維持するために使用されます。

セッション サービスを実装する AgentAPI メソッドは以下のとおりです。

- login()
- logout()

セッション管理を実行するエージェントは、ユーザ セッションを作成、委任、検証、および終了するために Java エージェント API のセッショニング サービスを使用します。

注: SiteMinder 管理者のポリシー サーバまたは DMS セッションに対するログインおよびログアウトには、Utility パッケージの SmApiSession.login() および SmApiSession.logout() メソッドを使用します。

セッション作成およびセッション指定

セッションは、ユーザ ログインが成功した後に作成されます。 作成後は終了されるまでユーザ セッションが継続します。

ユーザが認証されるとき、ポリシー サーバは*セッション指定*を発行します。 セッション指定にはユーザについての情報が含まれます。

多層アプリケーション環境でのユーザ側セッション永続性は、セッション指定内のユーザ情報の保存および維持により実行されます。 このセッション指定は、ユーザ セッションを表します。これは SiteMinder セッション管理の鍵です。

ユーザ セッションが作成された SiteMinder 環境は、セッション指定の作成、維持および永続ストレージを実行します。 たとえば、Web エージェント（HTTP 環境）はセッション仕様を HTTP Cookie に保存します。

エージェントは login() を使用してセッションを作成します。 このメソッドは、ユーザ認証情報を認証し、セッション指定に対する情報（一意のセッション ID を含む）を取得します。 一度作成されたら、セッション指定は更新された期限切れ時間も返す後続の Java エージェント API 呼び出しで更新されます。 エージェントは、この情報を使用してカスタム セッション管理を実行し、セッション タイムアウトを追跡できます。

Web サーバのユーザ トラッキング機能がオンになっている場合、SiteMinder のポリシー サーバがセッション指定情報に加えて*識別チケット*を発行します。 識別チケットは、匿名認証方式によって保護されたリソースにユーザがアクセスしている場合のアイデンティティ ベースのパーソナライゼーションに使用できます。 識別チケットに有効期限はありません。

セッショニング メカニズムにシームレスに統合される別の重要な機能は SiteMinder の*ユニバーサル ID* です。 ユニバーサル ID は、社会保障番号または顧客のアカウント番号などの一意の識別子によって SiteMinder 環境内のアプリケーションにユーザを識別します。 ユニバーサル ID は、アプリケーションに関係なく、ユーザの識別を自動的に実行するとにより、新旧アプリケーション間のユーザ識別を簡素化します。 一度ポリシー サーバ上で設定されたら、ユーザのユニバーサル ID はセッション指定の一部になり、セッション全体の間エージェントに利用可能です。

セッション検証

エージェントは、ユーザ セッションが期限切れ、終了、または無効になっていないことを確認するためにセッション指定の検証を要求します。 これは、セッション期間中いつでも発生する可能性があります。 エージェントは、AgentAPI.login() を呼び出してセッション指定を検証します。

セッション委任

アプリケーションの論理フローがアプリケーション層と交差する場合、セッション指定を 2 つのエージェント間で渡すことによりセッションを委任することができます。 各エージェントは、セッション指定の検証を選択することができます。

セッション終了

セッションは以下のいずれかを理由として終了します。

- ユーザがログアウトし、エージェントがセッション指定を破棄した後
- セッションが期限切れになったとき
- セッションが取り消されたとき
- ユーザ アカウントが無効になってとき

セッションを終了するには、エージェントはセッション指定を破棄する必要があります。 セッションが終了すると、ユーザは新しいセッションを確立するために再度ログインする必要があります。

許可サービス

アクセス制御機能を実行するエージェントは、AgentAPI クラスの許可サービスを使用します。 これらのサービスにより、クライアントはユーザのリソースへのアクセス権の検証、特定のリソースに関するユーザの権限の取得、リソースに課される特定のアクセス制御があればその判断を実行することができます。

isProtected() メソッドを呼び出すことにより、リソースが保護されているか判断できます。 このメソッドは、要求元のエージェントが処理するリソースをパラメータとして受け入れ、ユーザの認証情報に関する情報を返します。

一度ユーザの識別情報が検証されたら、エージェントは authorize() メソッドを呼び出して要求元のユーザが要求されたリソースにアクセスできるかどうかを判断します。 エージェントは、このメソッドが取得する一連のレスポンス属性を使用することできめの細かいアクセス制御を実行できます。

監査サービスおよびトランザクション トラッキング

エージェントはセッション中にすべてのユーザ アクティビティを追跡およびログ記録できます。 ユーザのアクティビティの多くがポリシー サーバによってログ記録されますが、エージェント キャッシュ外で実行された許可をログ記録する必要が発生する場合があります。 エージェントはリソースに対するこのような要求をログ記録するために audit() メソッドを呼び出します。

エージェントは、一意のトランザクション ID を生成することによって、アクセス制御アクティビティアプリケーション アクティビティに相関させることができます。 トランザクション ID は、認可方式と監査方式の両方に与えることができるため、ポリシー サーバではトランザクションに固有の ID をアプリケーション アクティビティと関連付けて記録できます。これは非否認に使用できます。

管理サービス

エージェントと SiteMinder ポリシー サーバの間には管理プロトコルが存在します。 このプロトコルは、エージェントがキャッシュおよび暗号化キーをポリシー サーバ上の SiteMinder ポリシーと管理変更の両方の間で一貫した方法で管理するのに役立ちます。

最新のエージェント コマンドを要求するには、エージェントは ManagementContextDef オブジェクトに MANAGEMENT_GET_AGENT_COMMANDS が設定されたメソッド doManagement() を呼び出します。 通常、この呼び出しはバックグラウンドで実行されるスレッドによって *n* 秒ごとに実行されます。 取得できるエージェント コマンドのタイプはキャッシュ コマンドおよび暗号化コマンドです。

キャッシュ コマンド

キャッシュ コマンドは、ポリシー サーバの管理上の更新の結果として実行されることが必要となる可能性のあるキャッシュ変更をエージェントに通知します。

キャッシュ コマンドは以下のとおりです。

- CACHE_FLUSH_ALL
- CACHE_FLUSH_ALL_USERS
- CACHE_FLUSH_THIS_USER
- CACHE_FLUSH_ALL_REALMS
- CACHE_FLUSH_THIS_REALM

暗号化コマンド

暗号化コマンドは、管理的にまたはポリシー サーバによって自動的に生成された新規暗号化キーについてエージェントに通知します。 セキュアな状態を保存するエージェントは、このプロトコルを最新の暗号化キーを追跡するために使用できます。

暗号化コマンドは以下のとおりです。

- AFFILIATE_KEY_UPDATE
- AGENT_KEY_UPDATE_NEXT
- AGENT_KEY_UPDATE_LAST
- AGENT_KEY_UPDATE_CURRENT
- AGENT_KEY_UPDATE_PERSISTENT

トンネル サービス

トンネル サービスは、ポリシー サーバ上に置かれた呼び出し可能なサービスとのセキュアな通信を確立するためにエージェントを有効化します。これにより、暗号化キー管理などの問題を処理する必要なしに、エージェントはセキュアな VPN に似たチャネルを介してカスタム アクションを実行できます。

レスポンス属性

レスポンス属性は、エージェントに情報を配信するためにポリシー サーバを有効化します。 レスポンス属性は AgentAPI クラス内のメソッドによって管理されます。

レスポンス属性には 2 つのタイプがあります。

- 既知
- ポリシー ベース

*既知*の属性は、常に login() などの特定の呼び出しの後にポリシー サーバによって返されます。 これらの属性は、ユーザ DN やユニバーサル ID などの静的な固定データを表します。

*ポリシー ベース*の属性は、login() および authorize() メソッドによって返されます。 これらの属性はポリシーに基づいており、ポリシー サーバからエージェントへスタティックおよびダイナミック データを配信するための手段で、認証属性と許可属性を識別できます。 データの実際のソースは、各種のソースからのデータを配信するように設定できるレスポンス機能を使用してポリシー サーバに定義されます。 データには、スタティック情報、ディレクトリ プロファイルからの情報、またはカスタム ポリシー サーバ プラグインなどがあります。 一度レスポンスが適切に設定されたら、エージェントはプロファイル駆動のパーソナル化だけでなく、きめ細かいアクセス制御を実行することができます。

ポリシー定義に基づき、レスポンス属性はタイムアウトする場合や、ユーザ セッションの期間にキャッシュされる場合があります。 ポリシー サーバは、秒単位で計算された TTL（有効期間）値と共に属性を配信します。 エージェントは、ユーザ セッションと許可の両方またはいずれか一方をキャッシュしている場合、関連する属性も最新状態に保ちます。 エージェントは、古くなった属性を更新するために updateAttributes() メソッドを発行します。

シングル サインオン

シングル サインオン環境では、所定のエージェントで正しく認証されたユーザが別のエージェントによって保護されたレルムにアクセスする場合、再認証は不要です。 カスタム エージェントがシングル サインオン環境に関与する場合、 2 つのエージェントは同じ Cookie ドメイン（たとえば、xxx.domainname.com など）に配置される必要があります。

シングル サインオンは、SMSESSION というシングル サインオン Cookie によって実現されます。この Cookie は、SiteMinder またはカスタム エージェントによって作成され、ユーザのブラウザに書き込まれます。

クラス AgentAPI には、カスタム エージェントが標準 SiteMinder Web エージェントを持つシングル サインオン環境に参加するのを可能にする 2 つのメソッドが含まれます。

decodeSSOToken()

カスタム エージェントは、既存の SMSESSION Cookie からトークンと呼ばれる Cookie の内容を抽出し、トークンをこのメソッドへ渡します。メソッドはトークンを復号し、指定された情報を抽出します。 このメソッドは、トークン内の最後のアクセス タイムスタンプを更新するために使用することもできます。

createSSOToken()

ユーザがカスタム エージェントを通して正常にログインした後、カスタム エージェントはユーザに関する情報をこのメソッドに渡します。メソッドは、このユーザ情報およびログイン コールから返されたセッション情報から暗号化されたトークンを作成します。 カスタム エージェントは、トークンを SMSESSION Cookie に書き込みます。

シングル サインオン メソッドに対するパラメータの設定および結果の解析の例については、サンプル カスタム エージェント コードを参照してください。 サンプル カスタム エージェント コードは、<install_path>¥sdk¥samples の smjavaagentapi ディレクトリに置かれています。

カスタム エージェントを通したログオン

ここでは、初期ログインがカスタム エージェントを通して実行される場合のシングル サインオン環境のイベントの典型的なシーケンスを示します。

1. ユーザはカスタム エージェント経由でログインします。
2. カスタム エージェントは、ユーザを認証するために login() を呼び出します。 ユーザは認証情報を要求されます。
3. カスタム エージェントは createSSOToken() を呼び出し、ユーザに関する情報（要求元のクライアントのユーザ名、ユーザ DN、IP アドレス）を渡します。 SiteMinder は、この情報を、ログイン コールから返されたセッション情報と共にトークンに追加します。 SiteMinder はトークン内の情報の暗号化も実行します。
4. カスタム エージェントは、ユーザのブラウザに SMSESSION Cookie を作成し、この Cookie にトークンを書き込みます。
5. ユーザは標準的な SiteMinder エージェントによって保護されたリソースを要求します。
6. 標準エージェントはログイン操作を実行します。この操作により、シングル サインオン Cookie 内の情報に基づいてユーザが検証されます。ユーザは、認証情報を要求されません。

標準エージェントを通したログオン

ここでは、初期ログインが標準 SiteMinder Web エージェントを通して実行される場合にシングル サインオン環境で発生するイベントの典型的なシーケンスを示します。

1. ユーザは標準エージェント経由でログインします。
2. 標準エージェントは、ログイン コールを通じて認証情報をユーザに要求して、ユーザを認証します。
3. SiteMinder は、ユーザのブラウザに SMSESSION Cookie を作成し、セッション情報が含まれる暗号化トークンを挿入します。
4. ユーザはカスタム エージェントによって保護されたリソースを要求します。

5. カスタム エージェントは、ユーザの要求から SMSESSION Cookie を取得し、トークンを抽出します。
6. カスタム エージェントはトークンをメソッド decodeSSOToken() へ渡します。メソッドはトークンを復号し、カスタム エージェントにトークンの属性のサブセットを返します。
7. カスタム エージェントは、トークンからのセッション指定を取得し、セッション指定を login() へ渡します。ログイン コールにより、ユーザへ認証情報を要求することなく、ユーザが検証されます。
8. ユーザは標準的な SiteMinder エージェントによって保護されたリソースを要求します。
9. 標準エージェントはログイン操作を実行し、SMSESSION Cookie の内容に基づいてユーザを検証します。 ユーザは、認証情報を要求されません。

標準エージェント サポート

SiteMinder SDK v6.x で作成されたカスタム エージェントは、標準的な SiteMinder v4.x、v5.x または v6.x Web エージェントによって作成された SMSESSION Cookie を受け入れることができます。 ただし、標準的な SiteMinder v4.x または v5.x Web エージェントは、標準エージェントが適切な Siteminder Agent Quarterly Maintenance Release (QMR) でアップグレードされている場合は、カスタム エージェントによって作成された Cookie のみを受け入れることができます。 各標準エージェントのバージョンに必要な QMR バージョンの詳細については、付属の SDK リリース ノートを参照してください。

SiteMinder v4.x または v5.x エージェントを適切な QMR アップグレードで有効にし、カスタム エージェントによって作成された SMSESSION Cookie を受け入れるようにするには、標準エージェントのエージェント設定ファイル（IIS サーバの場合は LocalConfig.conf、その他のサーバの場合は WebAgent.conf）または中央設定オブジェクト（v5.x 以降）に以下のエントリが含まれる必要があります。

```
AcceptTPCookie="yes"
```

AcceptTPCookie を以下のとおりに設定します。

- 4.xQMR4 エージェント以降では、標準エージェントのエージェント設定ファイルに AcceptTPCookie="yes" を直接追加します。
- 5.xQMR1 エージェント以降では、標準エージェントのエージェント設定オブジェクトに対する AllowLocalConfig パラメータが no に設定されている場合はそのオブジェクトにエントリを追加します。
  AllowLocalConfig が yes の場合は、標準エージェントのエージェント設定ファイルに AcceptTPCookie を設定できます。

情報がセッションにバインドされる方法

セッション情報の構成には、セッション仕様以外の情報も含まれる場合があります。 セッション情報には、クライアント アプリケーションがユーザのセッションとの関連付けに必要とする情報を含めることができます。

アプリケーションに定義されるセッション情報は、*セッション変数*と呼ばれる名前/値のペアで構成されます。 たとえば、アフィリエイト操作用のビジネス ロジック、証明書情報および SAML アサーションはすべて、セッション変数として保存し、セッション ID にバインドできます。

クラス AgentAPI は、セッション変数を設定、取得および削除するための以下のメソッドを提供します。

- setSessionVariables()
- getSessionVariables()
- delSessionVariables()

セッション変数は、*セッション ストア*と呼ばれるサーバ側のデータベースに保存されます。 セッション ストアはポリシー サーバによって管理されます。

セッション変数の利点

クライアント アプリケーションがセッション変数を使用する場合、

- 各セッション変数値に対して最大 4K のデータを保存できます。
- セッション情報が複数のポリシー サーバにわたって維持されます。サーバ上のセッション情報の一元化により、異なるドメインにわたるログアウトおよびアイドル タイムアウトの適用を含むクロス ドメイン セッション管理などの機能が可能になります。

セッション変数を使用するための要件

クライアント アプリケーションでセッション変数を使用するには、以下の前提条件の両方に適合する必要があります。

- セッション ストアをポリシー サーバ管理コンソールで有効にする必要があります。
- ポリシー サーバ UI 内のレルム設定中に、セッション中に少なくとも 1 つのレルムにアクセスするには永続セッションが選択される必要があります。永続セッションに対して設定されたレルムにユーザがアクセスすると即座に、それ以降のセッション全体を通じてセッション変数が使用可能になります。

セッション終了時のクリーンアップ

ユーザがログアウトし、エージェントがセッション指定を破棄すると、セッションは終了します。永続セッションの場合、SiteMinder は、すべてのセッション変数を含めたすべてのセッション情報をセッション ストアから削除します。

サーバ クラスタ

サービス障害の防止を支援するために、SiteMinder にはフェールオーバ機能が含まれます。プライマリ ポリシー サーバが失敗し、フェールオーバが有効の場合、バックアップ ポリシー サーバがポリシー操作を引き継ぎます。SiteMinder v6.0 以降では、フェールオーバをポリシー サーバ間だけでなく、ポリシー サーバのグループまたはクラスタ間に適用することができます。

クラスタ機能は、クラスタ内のサーバ間で動的負荷分散機能を提供することで、サーバのパフォーマンスも向上します。 動的な負荷分散では、各サーバのパフォーマンス能力に応じてポリシー操作がクラスタ内の使用可能サーバ間で自動的に分散されます。

クラスタ化サーバおよび非クラスタ化サーバ

エージェント API v6.x に対して実行されるエージェントは、1 つ以上のポリシー サーバ、またはポリシー サーバ の 1 つ以上のクラスタと以下のように関連付けることができます。

- **クラスタ化サーバ**

  各クラスタ化サーバに対する ServerDef オブジェクト内で、clusterSeq() をクラスタに対するシーケンス番号に対して設定します。 クラスタ内のサーバのクラスタ シーケンス番号はすべて同じです。

  動作： 複数のクラスタが定義される場合、フェールオーバがサーバのクラス間で行われます。また、クラスタ内のサーバへの要求は、Agent API v6.0 に導入された、機能強化パフォーマンスベース負荷分散テクニックにしたがって送信されます。

- **非クラスタ化サーバ**

  それぞれの非クラスタ化サーバに対する ServerDef オブジェクトで、メソッド clusterSeq() を 0 に設定します。

  動作： 動作は v5.x インストールと同じです。つまり、エージェントと関連付けられたサーバ間のフェールオーバを有効にするか、サーバ間のラウンド ロビン動作を有効にできます。

  ラウンド ロビン動作が有効な場合、エージェント API v6.0 で導入された、改善されたパフォーマンス ベースのロード バランス技術が使用されます。

注: 同じエージェントをクラスタ化サーバおよび非クラスタ化サーバと関連付けることはできません。

クラスタの設定

エージェント API またはホスト設定オブジェクトを通してポリシー サーバ ユーザ インターフェースを使用してクラスタを設定できます。 エージェント API を通してクラスタを設定する場合、設定がホスト設定オブジェクトに定義されている可能性がある他のクラスタ設定情報と競合しないことを確認してください。

InitDef および ServerDef クラスを通して、エージェントが関連付けられる個別のサーバまたはサーバのクラスタを設定します。

クラスタ フェールオーバは、以下の設定に従って発生します。

- **フェールオーバしきい値**。ポリシー サーバ要求に対して利用可能である必要があるクラスタ内のサーバの最小の割合 (%) 。利用可能なサーバの数がしきい値以下になると、次のクラスタへのフェールオーバが行われます。

  フェールオーバしきい値の割合はエージェントと関連付けられたすべてのクラスタに適用されます。

  割合が特定のクラスタで表すサーバの数を判断するには、しきい値の割合とクラスタ内のサーバ数の積を計算し、最も近い整数値に丸めます。 たとえば、5 つのサーバのクラスタでフェールオーバしきい値が 60% の場合、次のクラスタへのフェールオーバは、現在アクティブな使用可能なサーバの数が 3 未満になる場合に発生します。

  次を通して設定： failOverThreshold パラメータが含まれる InitDef コンストラクタ。

- **クラスタ フェールオーバのシーケンス**。クラスタはそれぞれシーケンス番号を割り当てられます。クラスタ フェールオーバが発生するとき、SiteMinder は後続のポリシー サーバ要求をクラスタ シーケンス内の次のクラスタに送信します。

  次を通して設定： ServerDef.clusterSeq()。

クラスタ フェールオーバ

サイトがクラスタを使用する場合、通常 1 つのプライマリ クラスタ、そして 1 つ以上のバックアップ クラスタがあります。

*プライマリ クラスタ*は、クラスタ内の利用可能なサーバの数がフェールオーバしきい値以下に減少しない限り、SiteMinder が要求を送信するクラスタです。 プライマリ クラスタに十分な利用可能なサーバがない場合、クラスタ シーケンスの次のクラスタへのフェールオーバが発生します。 このクラスタも失敗する場合、3 番目のクラスタへのフェールオーバが発生し、同様に続いて行きます。

クラスタがすべて失敗するとき

利用可能なサーバの数がエージェントと関連付けられたすべてのクラスタ内のフェールオーバしきい値以下に減少する場合、ポリシー操作は停止しません。 要求は、少なくとも 1 つの利用可能なサーバがあるクラスタ シーケンスの最初のクラスタに送信されます。

たとえば、エージェントが 3 つのサーバが含まれる C1 および 5 つのサーバが含まれる C2 の 2 つのクラスタと関連付けられるとします。エージェントと関連付けられるクラスタのフェールオーバしきい値は 60% に設定されます。

以下の表は、各クラスタ内で利用可能である必要があるサーバの最小数を示しています。

クラスタ	クラスタ内のサーバ数	割合 フェールオーバ しきい値	数値 フェールオーバ しきい値 (最小 利用可能サーバ)
C1	3	60	1
C2	5	60	3

利用可能なサーバの数が各クラスタ内でしきい値未満になり、その結果、C1 には利用可能なサーバがなく、C2 に 2 つのみ利用可能サーバがあるという状態になった場合、着信する次の要求は最良のレスポンス時間を持つ C2 サーバに送信されます。 少なくとも 3 つの C1 サーバのうちの 2 つが修復された後は、後続の要求が利用可能な C1 サーバ間で負荷分散されます。

バージョン互換性およびフェールオーバ動作

エージェント API v6 は、エージェント API v5 と下位互換性があり、 v5/v6 エージェントおよび v5/v6 エージェント API との完全な相互操作性が実現されます。

タイムアウト

エージェントはセッション タイムアウトを適用するか、または各要求の検証をポリシー サーバに依存できます。 通常、エージェントによるユーザ セッションや権限のキャッシュには、エージェント側での何らかの形式のタイムアウト適用が必要となります。 この場合、エージェントは最後のアクセス時間の追跡およびセッションの有効期限の把握に対して責任を負います。

キャッシュしないエージェントは、ポリシー サーバのタイムアウトの適用を利用できます。以下の Java エージェント API メソッドがすべての呼び出しの後に更新されたタイムアウト情報を返します。

- login()
- authorize()
- audit()

第 4 章: ポリシー管理 API

このセクションには、以下のトピックが含まれています。

ポリシー管理について

ポリシー管理は SiteMinder ポリシー ストア内のポリシー オブジェクトの作成、削除、変更から構成されます。 ポリシー管理 API によって、実行可能なデータ操作のほとんどを、ネイティブ ポリシー サーバ ユーザ インターフェースを通して実行できます。 たとえば、管理者が以下のようなタスクを実行することを可能にするクライアント アプリケーションを記述することができます。

- ポリシー ドメインの作成
- エージェント オブジェクトの作成
- エージェント設定オブジェクトの作成
- ホスト設定オブジェクトの作成
- トラステッド ホストの登録
- SiteMinder ユーザ ディレクトリ オブジェクトの作成
- 認証方式オブジェクトの作成
- 管理者の作成
- レルムの作成
- ポリシー ドメインへのレルムの追加
- ルールの作成
- レスポンスの作成
- ポリシーの作成
- ユーザまたはグループのポリシーへの追加
- ポリシーへのルールの追加
- ポリシー内のルールに対するレスポンスの設定
- ポリシー ストア全体または個別のポリシー ドメインのリモート移行

ポリシー管理のセットアップ

ポリシー管理 API で構築されたアプリケーションを起動する方法

- ポリシー サーバ管理コンソールを使用して、アクセスしたいポリシー ストアをポイントするようにポリシー サーバを設定します。
- ポリシー サーバと同じマシン、またはポリシー サーバに対してネットワーク アクセスがあるマシン上でポリシー管理アプリケーションを実行します。

注: ポリシー管理 API で構築されたアプリケーションがポリシー サーバと同じマシン上で実行される場合、アプリケーションはポリシー サーバをインストールしたユーザと同じユーザとして実行する必要があります（たとえば、UNIX プラットフォーム上では smuser）。

必要な JAR ファイル

ポリシー管理アプリケーションの構築および起動には JAR ファイル smjavasdk2.jar が必要です。 JAR ファイルは以下の場所に保存されます。

- Windows プラットフォーム：

  <install_path>¥sdk¥java

- UNIX プラットフォーム：

  <install_path>/sdk/java

ポリシー ストア オブジェクト

インターフェース SmPolicyApi はクラス SmPolicyApiImpl によって実装されます。 ポリシー管理 API の開始点としてこのクラスを使用します。 各ポリシー ストア オブジェクトは、ポリシー管理 API 内のクラスと関連付けられます。 オブジェクトのクラス内のメソッドを通してポリシー ストア オブジェクトを作成および管理します。

ポリシー ストア オブジェクトはスコープに従って分類できます。

- *ドメイン オブジェクト*はドメイン内のみで表示されます。これらをドメイン間で共有することはできません。
- *グローバル オブジェクト*は、すべてのドメインで表示されます。

  グローバル オブジェクトはシステム オブジェクトと呼ばれる場合もあります。

グローバル オブジェクトには以下のものが含まれます。

- 管理者
- エージェント タイプ
- エージェントおよびエージェント グループ
- エージェント設定オブジェクト
- ホスト設定オブジェクト
- トラステッド ホスト
- 認証方式
- 認証/認可マップ
- 証明書マップ
- ドメイン
- ODBC クエリ方式
- パスワード ポリシー
- 登録方式
- ユーザ ディレクトリ

ドメイン オブジェクトには以下が含まれます。

- ポリシー
- レルム
- レスポンスおよびレスポンス グループ
- レスポンス属性

- ルールおよびルール グループ
- ユーザ ポリシー

ポリシー サーバのユーザ インターフェースを使用している場合、SiteMinder 管理ウィンドウの［システム］と［ドメイン］のタブに上記のオブジェクトの大部分がリスト表示されます。

注: Javadoc リファレンスの説明は、オブジェクトにグローバル スコープまたはドメイン スコープがあるかどうかを指定します。

ポリシー管理アプリケーションの記述

ポリシー管理アプリケーションを記述する方法

1. ポリシー サーバへの接続の確立
2. セッション オブジェクトの取得
3. セッション オブジェクトを渡す
4. ポリシー管理 API 要求の作成
5. 管理者セッションの終了

SiteMinder SDK には、Java ポリシー管理 API でクラスとメソッドを使用する方法のサンプルが含まれます。

ポリシー サーバへの接続の確立

ポリシー サーバへの接続を確立するには、ユーティリティ パッケージの SmApiConnection クラスを使用します。 このクラスは、Java API 要求が送信されるエージェント API ハンドルを保持します。

このクラスには 2 つのタイプの接続ハンドルがあります。

- *デフォルト*接続ハンドル。デフォルト接続ハンドルには以下の特徴があります。
  - エージェント API オブジェクトの単一のインスタンスを表します。
  - プロセスを通してスタティックです。
  - ポリシー管理 および DMS クライアントの両方からの エージェント API オブジェクトへの接続を許可します。

  単一のエージェント API オブジェクト インスタンスを通してポリシー サーバへの複数の接続を確立できます。
- *ユーザ定義*の接続ハンドル。複数のユーザ定義接続オブジェクトを作成できます。各オブジェクトはポリシー サーバへの複数の接続をサポートできます。

セッション オブジェクトの取得

ユーザまたは管理者が正常にログインするとき、セッション オブジェクトが取得されます。 この場合、管理者のみがポリシー管理を実行できるので、管理者ログインが必要です。

SiteMinder 管理者にログインし管理者セッションを確立するには、ユーティリティ パッケージの SmApiSession クラスで login() メソッドを呼び出します。

ログインが成功すると、セッション オブジェクトは有効な管理者セッション指定を保持します。

セッション オブジェクトを渡す

有効なセッションを取得した後に、セッションを SmPolicyApiImpl クラスのコンストラクタへ渡すことでポリシー管理 API オブジェクトを作成します。たとえば、以下のようにします。

```
SmPolicyApi policyApi = new SmPolicyApiImpl (apiSession);
```

この例で、policyApi は新規ポリシー管理 API オブジェクトで、*apiSession* は管理者がログインに成功した場合に取得されるセッションです。

ポリシー管理 API 要求の作成

セッション オブジェクトを取得し、ポリシー管理 API オブジェクトを作成したら、ポリシー管理を要求する準備ができています。 ポリシー管理 API 内のほとんどのメソッドは、特定のメソッドが処理する SiteMinder オブジェクトに従って、たとえばエージェント、ポリシーおよびルールのように分類されます。

キャッシュや暗号化キー管理などのサービスを実行するメソッドに対するユーティリティ カテゴリもあります。 カスタム ポリシー管理アプリケーション内で使用する特定のポリシー管理 API を見つけるためにこれらのカテゴリを使用します。

注: policyapi パッケージ内のメソッドは Siteminder 管理者セッションからのみ呼び出すことができます。

管理者セッションの終了

ポリシー管理 API 要求の作成を完了したら、ユーティリティ パッケージの SmApiSession クラス内の logout() メソッドを呼び出して管理者をログアウトします。

重要: logout() メソッドを呼び出すと、接続ハンドルが無効になります。再度参照しないでください。

管理者メソッド

別途指定がない限り、以下のメソッドはクラス SmPolicyApiImpl 内にあります。 以下のメソッドは管理者オブジェクトに対して実行されます。 管理者オブジェクトは SmAdmin のインスタンス化により作成します。

メソッド	説明
addAdmin()	ポリシー ストアへ管理者オブジェクトを追加します。
addAdminToDomain()	ドメインと管理者を関連付けます。
deleteAdmin()	管理者を削除します。
getAdmin()	管理者の内容を取得します。

メソッド	説明
getAdminUserDirs()	管理者が管理できるユーザ ディレクトリのリストを取得します。
modifyAdmin()	管理者を変更します。
removeAdminFromDomain()	ドメインから管理者を分離します。

エージェント メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはエージェント オブジェクトに対して実行されます。 エージェント オブジェクトは SmAgent のインスタンス化により作成します。

メソッド	説明
addAgent()	ポリシー ストアにエージェント オブジェクトを追加します。
deleteAgent()	エージェントを削除します。
getAgent()	エージェントの内容を取得します。
modifyAgent()	エージェントを変更します。

エージェント設定オブジェクト メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはエージェント設定オブジェクトに対して実行されます。 エージェント設定オブジェクトは SmAgentConfig のインスタンス化により定義します。

メソッド	説明
addAgentConfig()	ポリシー ストアにエージェント設定オブジェクトを追加します。
deleteAgentConfig()	エージェント設定オブジェクトを削除します。

メソッド	説明
getAgentConfig()	エージェント設定オブジェクトの内容を取得します。
modifyAgentConfig()	エージェント設定オブジェクトを変更します。

認証および許可マップ メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドは認証および許可ディレクトリ マッピング オブジェクトに対して実行されます。 認証および許可ディレクトリ マッピング オブジェクトは SmAuthAzMap のインスタンス化により作成します。

メソッド	説明
addAuthAzMap()	ポリシー ストアに認証および許可ディレクトリ マッピング オブジェクトを追加します。
deleteAuthAzMap()	認証および許可ディレクトリ マッピング オブジェクトを削除します。
getAuthAzMap()	認証および許可ディレクトリ マッピング オブジェクトの内容を取得します。
modifyAuthAzMap()	認証および許可ディレクトリ マッピング オブジェクトを変更します。

認証方式メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドは認証方式に対して実行されます。 認証方式は SmScheme のインスタンス化により作成します。

メソッド	説明
addScheme()	ポリシー ストアへ認証方式を追加します。

メソッド	説明
deleteScheme()	認証方式を削除します。
getScheme()	認証方式の内容を取得します。
modifyScheme()	認証方式を実装します。

証明書マップ メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドは証明書をマッピングするオブジェクトに対して実行されます。 証明書マッピング オブジェクトは SmCertMap のインスタンス化により作成します。

メソッド	説明
addCertMap()	ポリシー ストアに証明書マッピング オブジェクトを追加します。
deleteCertMap()	証明書マッピング オブジェクトを削除します。
getCertMap()	証明書マッピング オブジェクトの内容を取得します。
modifyCertMap()	証明書マッピング オブジェクトを変更します。

ドメイン メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはドメイン オブジェクトに対して実行されます。 ドメイン オブジェクトは SmDomain のインスタンス化により作成します。

メソッド	説明
addDomain()	ポリシー ストアにドメイン オブジェクトを追加します。
deleteDomain()	ドメインを削除します。

メソッド	説明
getDomain()	ドメインの内容を取得します。
getDomainObject()	指定されたオブジェクト名または OID に対するドメイン オブジェクトを取得します。
getDomainObjectNames()	ドメイン内のドメイン オブジェクトのリストを取得します。
isDomainObject()	オブジェクトがドメイン オブジェクトかどうかを示します。 クラス SmObjectImpl、SmDomainObjectImpl で適用されます。
modifyDomain()	ドメインを変更します。

一般的なオブジェクト メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドは複数のタイプのオブジェクトに対して実行されます。

メソッド	説明
getGlobalObjectNames()	グローバル オブジェクトのリストを取得します。
getObject()	指定されたオブジェクト名または OID に対するグローバル オブジェクトを取得します。
getOid()	オブジェクトに対するオブジェクト識別子を取得します。 クラス SmObjectImpl で適用されます。
isWriteable()	オブジェクトが書き込み可能かどうかを指定します。 クラス SmAgentType、SmDomainObjectImpl および SmObjectImpl で適用されます。
renameObject()	オブジェクトの名前を変更します。

グループ メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはグループ オブジェクトに対して実行されます。グループ オブジェクトは、SmAgentGroup（エージェント グループ）、SmResponseGroup （レスポンス グループ）または SmRuleGroup （ルール グループ）で作成されます。

メソッド	説明
addGroup()	エージェント、レスポンスまたはポリシー ストアにルール グループを追加します。
addToGroup()	指定されたグループにルール、レスポンスまたはエージェント タイプのグループ要素を追加します。
deleteGroup()	既存のグループを削除します。
getGroup()	既存のグループの内容を取得します。
getGroupMembers()	すべてのタイプのグループのリストを取得します。
modifyGroup()	グループを変更します。
removeFromGroup()	グループからグループ要素を削除します。

ホスト設定オブジェクト メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはホスト設定オブジェクトに対して実行されます。 ホスト設定オブジェクトは SmHostConfig のインスタンス化により定義します。

メソッド	説明
addHostConfig()	ポリシー ストアにホスト設定オブジェクトを追加します。
deleteHostConfig()	ホスト設定オブジェクトを削除します。

メソッド	説明
getHostConfig()	ホスト設定オブジェクトの内容を取得します。
modifyHostConfig()	ホスト設定オブジェクトを変更します。

ODBC クエリ方式メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。以下のメソッドは ODBC クエリ方式に対して実行されます。 ODBC クエリ方式は SmODBCQuery のインスタンス化により作成します。

メソッド	説明
addODBCQuery()	ポリシー ストアに ODBC クエリ オブジェクトを追加します。
deleteODBCQuery()	ODBC クエリ オブジェクトを削除します。
getODBCQuery()	ODBC クエリ オブジェクトの内容を取得します。
modifyODBCQuery()	ODBC クエリ オブジェクトを変更します。

パスワード ポリシー メソッド

別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 以下のメソッドはパスワード ポリシー オブジェクトに対して実行されます。 パスワード ポリシー オブジェクトは SmPasswordPolicy のインスタンス化により作成します。

メソッド	説明
addPasswordPolicy()	ポリシー ストアにパスワード ポリシー オブジェクトを追加します。
deletePasswordPolicy()	パスワード ポリシーを削除します。
getPasswordPolicy()	パスワード ポリシーの内容を取得します。

メソッド	説明
isEnabled()	パスワード ポリシーを有効にするかどうかを指定します。 クラス SmPasswordPolicy で実行されます。
isEntireDir()	パスワード ポリシーがユーザ ディレクトリ全体に適用されるかどうかを指定します。 クラス SmPasswordPolicy で実行されます。
modifyPasswordPolicy()	パスワード ポリシーを変更します。

ポリシー メソッド

以下のメソッドはポリシーとポリシー リンク オブジェクトに対して実行されます。 ポリシー リンクは、ポリシー、ルール、およびオプションでレスポンスの関連付けです。 別途指定がない限り、これらのメソッドはクラス SmPolicyApiImpl にあります。

ポリシー オブジェクトは SmPolicy で作成されます。ポリシー リンク オブジェクトは SmPolicyLink で作成されます。

メソッド	説明
addPolicy()	ポリシー ストアにポリシー オブジェクトを追加します。
addPolicyLink()	ポリシーにポリシー リンクを追加します。
deletePolicy()	指定されたドメインに関連付けられたポリシーを削除します。
deletePolicyLink()	ポリシーからポリシー リンクを削除します。
getPolicy()	ポリシーの内容を取得します。
getPolicyLinks()	指定されたポリシーおよびドメインに対するポリシー リンクをすべて取得します。
modifyPolicy()	指定されたドメインに関連付けられたポリシーを変更します。
modifyPolicyLink()	指定されたポリシー リンクを変更します。

レルム メソッド

以下のメソッドはレルム オブジェクトに対して実行されます。 レルム オブジェクトは SmRealm で作成されます。

メソッド	説明
addRealm(()	ポリシー ストアにレルム オブジェクトを追加します。
deleteRealm()	レルムを削除します。
getRealm()	レルムの内容を取得します。
getRealmRules()	指定されたレルムおよびドメインに対するルールをすべて取得します。
getRealmUserPolicies()	レルムにアクセスできるユーザ ポリシーのリストを取得します。
modifyRealm()	指定されたレルムを変更します。

レスポンス メソッド

以下のメソッドはレスポンスとレスポンス属性オブジェクトに作用します。 別途指定がない限り、これらのメソッドはクラス SmPolicyApiImpl にあります。 レスポンス オブジェクトは SmResponse で作成されます。 レスポンス属性オブジェクトは SmResponseAttr で作成されます。

メソッド	説明
addResponse()	ポリシー ストアにレスポンス オブジェクトを追加します。
addResponseAttr()	レスポンス属性を作成し、レスポンスと関連付けます。
deleteResponse()	レスポンスを削除します。
deleteResponseAttribute()	レスポンス属性を削除します。
getResponse()	レスポンスの内容を取得します。
getResponseAttrs()	指定されたレスポンスに対する属性のリストを取得します。

メソッド	説明
modifyResponse()	指定されたレスポンスを変更します。
setResponseInPolicyLink()	指定されたポリシー リンクに対するレスポンスを変更します。

ルート設定メソッド

以下のメソッドはルート設定オブジェクトに対して実行されます。 別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 ルート設定オブジェクトは SmRootConfig のインスタンス化により作成します。

メソッド	説明
addRootConfig()	ポリシー ストアにルート設定オブジェクトを追加します。
deleteRootConfig()	ルート設定を削除します。
getRootConfig()	ルート設定の内容を取得します。
modifyRootConfig()	ルート設定を変更します。

ルール メソッド

以下のメソッドはルール オブジェクトに対して実行されます。 別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。ルール オブジェクトは SmRule のインスタンス化により作成します。

メソッド	説明
addRule()	ポリシー ストアにルール オブジェクトを追加します。
deleteRule()	ルールを削除します。
getRule()	ルールの内容を取得します。
modifyRule()	ルールを変更します。

自己登録メソッド

以下のメソッドは自己登録オブジェクトに対して実行されます。 別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。自己登録オブジェクトは SmSelfReg のインスタンス化により作成します。

メソッド	説明
addSelfReg()	ポリシー ストアに自己登録オブジェクトを追加します。
deleteSelfReg()	自己登録オブジェクトを削除します。
getSelfReg()	自己登録オブジェクトの内容を取得します。
modifySelfReg()	自己登録オブジェクトを変更します。

トラステッド ホスト オブジェクト メソッド

以下のメソッドはトラステッド ホスト オブジェクトに対して実行されます。 別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 トラステッド ホスト オブジェクトは SmTrustedHost のインスタンス化により定義します。

メソッド	説明
addTrustedHost()	トラステッド ホストをポリシー サーバに登録します。
deleteTrustedHost()	トラステッド ホスト オブジェクトを削除します。

ユーザ ディレクトリ メソッド

ユーザ管理機能は DMS API で提供されます。ただし、ポリシー管理 API は、ユーザ属性を取得および設定するメソッドを提供します。 これらのメソッドは SmUserDirectory クラスにあります。

例：

- どのユーザ属性がユーザの無効な状態を保持するかを指定するには、SmUserDirectory 内の setDisabledAttr() を呼び出します。
- ユーザを無効および有効にするには、DMS API を使用します。

以下のメソッドはユーザ ディレクトリ オブジェクトに対して実行されます。 別途指定がない限り、このセクションにリスト表示されたメソッドは、クラス SmPolicyApiImpl にあります。 ユーザ ディレクトリ オブジェクトは SmUserDirectory のインスタンス化により作成します。

メソッド	説明
addUserDirectory()	ポリシー ストアにユーザ ディレクトリ オブジェクトを追加します。
addUserDirToDomain()	ドメインと既存のユーザ ディレクトリを関連付けます。
deleteUserDirectory()	ユーザ ディレクトリを削除します。
getDirectoryContents()	指定されたユーザ ディレクトリに対する識別名およびクラスのリストを取得します。
getUserDirectory()	ユーザ ディレクトリの内容を取得します。
getUserDirSearchOrder()	ユーザ ディレクトリ名のベクターを取得することで、ドメインに対するユーザ ディレクトリの検索順序を取得します。
lookupDirectory()	指定されたユーザ ディレクトリおよび検索パターンに対する識別名およびクラスのリストを取得します。
modifyUserDirectory()	ユーザ ディレクトリを変更します。
removeUserDirFromDomain()	ドメインから既存のユーザ ディレクトリを分離します。

メソッド	説明
setUserDirSearchOrder()	ドメインのユーザ ディレクトリの検索順序を設定します。

ユーザ ポリシー メソッド

以下のメソッドはユーザ ポリシー オブジェクトに対して実行されます。別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。 ユーザ ポリシー オブジェクトは SmUserPolicy のインスタンス化により作成します。

メソッド	説明
addUserPolicy()	ポリシー ストアにユーザ ポリシー オブジェクトを追加します。
deleteUserPolicy()	指定されたドメインに対するユーザ ポリシーを削除します。
getUserPolicies()	指定されたポリシーおよびドメインに対するすべてのユーザ ポリシーを取得します。

ユーティリティ メソッド

以下のメソッドは、キャッシュおよび暗号化キー管理を含めた各種のサービスを提供します。 別途指定がない限り、この表にリスト表示されたメソッドはクラス SmPolicyApiImpl にあります。

メソッド	説明
changeDynamicKey()	ダイナミックな暗号化キーを変更します。
changePersistentKey()	永続的な暗号化キーを変更します。
changeSessionKey()	セッション暗号化キーを変更します。
flushAll()	すべての SiteMinder キャッシュをクリアします。

メソッド	説明
flushRealm()	リソース キャッシュからレルムをクリアします。
flushRealms()	リソース キャッシュからすべてのレルムをクリアします。
flushUser()	ユーザ情報キャッシュからユーザをクリアします。
flushUsers()	情報キャッシュからすべてのユーザをクリアします。
search()	指定されたオブジェクトを検索します。
setApiSession()	API セッション オブジェクトを設定します。

オブジェクトの関連付け

オブジェクトの中には、お互いに関連付けたり分離したりできるものがあります。たとえば、AddAdminToDomain() はドメインに管理オブジェクトを追加し、RemoveAdminFromDomain() は管理オブジェクトをドメインから削除します。 追加操作では、オブジェクトの両方が呼び出しの前に存在する必要があります。 削除操作の後、オブジェクトはどちらもまだ存在しますが、お互いに関連付けられてはいません。

2 つのオブジェクトを関連付けまたは分離するメソッドを探している場合、追加または削除するメソッドのカテゴリを確認します。 たとえば、AddAdminToDomain() および RemoveAdminFromDomain() はいずれも［管理者メソッド］にあります。

ポリシーストアへのオブジェクトの追加

ポリシー管理 API オブジェクトを作成した後、ポリシー ストアに追加するオブジェクトを作成できます。

ポリシーストアにオブジェクトを追加する方法

1. ポリシー ストアに追加するオブジェクトを作成します。

   たとえば、エージェント オブジェクトを作成する場合は以下のようにします。

   ```
   SmAgent agent = new SmAgent();
   ```

2. Set the appropriate fields for the object—for example:

   ```
   agent.setName ("myAgent");
   agent.setSecret ("siteminder");
   agent.setDescription ("Sample agent");
   agent.setAgentType (SmAgentType.DefaultAgentType);
   ```

3. ポリシー ストアにオブジェクトを以下のように追加します。

   - 作成したばかりのオブジェクトに対して add... メソッドを呼び出します。たとえば、エージェント オブジェクトに対して addAgent()、またはドメイン オブジェクトに対して addDomain() を呼び出し、ポリシー ストアに追加したいオブジェクトを渡します。
   - 結果オブジェクトへ結果を返します。

   例：

   ```
   result = policyApi.addAgent(agent);
   ```

4. 結果を検討します。

呼び出しが成功した場合：

- メソッドは、isSuccess() メソッドが true を返す SmApiResult オブジェクトを返します。
- オブジェクトが SiteMinder ポリシー ストアに追加されます。
- 対応するオブジェクト構造内の［*Oid*］フィールドはオブジェクト識別子に設定されます。

ポリシーストアからのオブジェクトの取得

ポリシー ストアからオブジェクトを取得する方法

1. 返されたプロパティを保存するための関連するクラスのオブジェクトを作成します。たとえば、以下のコードはエージェント オブジェクトを作成します。

   ```
   SmAgent myAgent = new SmAgent();
   ```

2. 作成したばかりのオブジェクトに対して get... 関数を呼び出します。たとえば、エージェント オブジェクトには getAgent()、ドメイン オブジェクトには getDomain() を呼び出し、作成したばかりのオブジェクトを渡します。 たとえば、myAgent という名前のエージェントを取得する場合は以下のようになります。

   ```
   result = myPolicyApi.getAgent ("myAgent", myAgent);
   ```

メソッドが成功する場合、*myAgent* に特定のエージェント オブジェクトのプロパティが入力されます。 （get... メソッドがリストを取得する場合、リストはベクターに記述されます。）一致するオブジェクトがない場合、取得オブジェクトのプロパティは初期値を保持します。

ポリシーストアからのオブジェクトの削除

削除操作はポリシー ストアからオブジェクトを削除します。 ポリシー ストアから一度に 1 つのオブジェクトのみを削除できます。

オブジェクトを削除するには、削除するオブジェクトに対してオブジェクト削除メソッドを使用します。たとえば、エージェント オブジェクトには deleteAgent()、ドメイン オブジェクトに対しては deleteDomain() を実行します。

認証方式の設定

認証方式をプログラム的に設定する場合、通常はポリシー サーバ UI の［認証方式プロパティ］ダイアログ ボックスで提供される情報を提供します。

認証方式を設定する場合、SmScheme クラスの get... および set... メソッドを使用して以下の情報を提供します。

- 方式タイプ

  SiteMinder は数多くの標準認証方式タイプ（テンプレートとも呼ばれる）を提供します。 認証方式タイプはそれぞれ異なる方法で設定されます。 方式タイプは後のトピックで説明されます。

- 説明

  認証方式の簡単な説明。

- 保護レベル

  保護レベル値の有効範囲は 1 ～ 1000 です。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。

- ライブラリ

  認証方式ライブラリは、関連する認証方式タイプに対して認証プロセスを実行します。 事前定義済み認証方式にはそれぞれ出荷時にデフォルト ライブラリが含まれており、通常はそれを使用します。 オプションでデフォルトの代わりにカスタム ライブラリを使用することもできます。

- パラメータ

  HTML ログイン ページの URL など、認証方式が必要とする追加情報です。

  一部の認証方式では、パラメータ情報は［認証方式プロパティ］ダイアログ ボックスの［方式タイプ セットアップ］タブ内のフィールド値から構築されます。 パラメータ文字列が特定の方式タイプに対してどのように構築されるかを確認するには、このダイアログ ボックスを開き、適切な方式タイプを選択し、［方式タイプ セットアップ］タブ内のフィールドに値を入力し、構築されたパラメータを［詳細］タブで確認します。

  異なる認証方式タイプに対してパラメータ値を提供する情報については、「*Policy Design Guide*」の認証方式の章を確認してください。

- 共有秘密キー

  認証方式とポリシー サーバの両方にとって既知の情報です。認証方式ごとに、異なる種類の秘密キーを使用します。 ほとんどの方式では秘密キーを使用しません。

- テンプレートですか?

  認証方式がテンプレートかどうかを指定するフラグ。

- 管理者によって使用されますか?

  認証方式を使用して管理者を認証できるかどうかを指定するフラグです。

- 認証情報を保存しますか?

  ユーザの認証情報が保存されるかどうかを指定するフラグ。

- RADIUS ですか?

  方式を RADIUS エージェントと共に使用できるかどうかを指定するフラグ。

- パスワード チェックを無視しますか?

  方式のパスワード ポリシーが有効かどうかを指定するフラグ。 True (1) の場合、パスワード ポリシーは無効になります。

注: 情報のこれらのカテゴリは別の認証方式で別の目的に対して使用できます。 たとえば、TeleID 認証方式では、共有秘密キーは暗号化シードを提供するために使用されます。

匿名テンプレート

匿名方式タイプに基づいて認証方式を設定するときにこの表を使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeAnonymous) 匿名方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(0) 0 に設定します。 この方式タイプには適用できません。

情報タイプ	値の割り当てと意味
ライブラリ	setLibrary("smauthanon") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) ゲスト DN が含まれる文字列。 ゲスト DN と関連付けられたポリシーを匿名のユーザに適用する必要があります。
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

基本テンプレート

基本方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeBasic) 基本方式タイプです。
説明	setDescription(*description*) 認証方式の説明。

情報タイプ	値の割り当てと意味
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthdir") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter("") 空の文字列に設定します。 この方式には適用できません。
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

SSL を介した基本テンプレート

SSL を介した基本方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeBasicOverSSL) SSL を介した基本方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 10 です。
ライブラリ	setLibrary("smauthcert") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) SSL サーバのドメインまたは IP アドレス、および SSL 認証情報コレクタ（SCC）の名前が含まれる文字列。 形式 https://*server*/*SCC*?basic 以下の例ではデフォルトの SCC を使用します。 https://my.server.com/siteminderagent/nocert/smgetcred.scc?basic
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) この方式に対しては false (0) に設定します。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。

情報タイプ	値の割り当てと意味
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

カスタム テンプレート

カスタム方式タイプに基づいて認証方式を設定するときにこの表を使用します。 C 認証 API を使用して、カスタム方式を作成します。 詳細については、「*Developer's Guide for C*」を参照してください。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeCustom) カスタム方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 0 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary(*customLibName*) C 認証 API を使用して作成したカスタム共有ライブラリの名前。
パラメータ	setParameter(*param*) カスタム認証方式に必要な 1 つ以上のパラメータの任意の文字列。 SSL を使用するカスタム認証方式については、SSL ベースの認証に必要な SiteMinder Web エージェント ライブラリを参照する URL を提供する必要があります。
共有秘密キー	setSecret(*secret*) カスタム認証方式が認証情報の暗号化に使用する共有秘密キー（存在する場合）。

情報タイプ	値の割り当てと意味
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(*flag*) 管理者の認証に方式を使用できることを指定するには true (1)、または管理者の認証に方式を使用できないことを指定するには false (0) を指定します。 デフォルトは 0 です。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

HTML フォーム テンプレート

HTML フォーム方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeHTMLForm) HTML フォーム方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthhtml") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) ユーザ属性リストとフォーム認証情報コレクタ（FCC）の場所が含まれる文字列。 属性リストは AL= から始まり、リスト区切り文字としてカンマを使用する必要があります。また、セミコロンで終了する必要があります。たとえば、以下のようになります。 AL=Password,SSN,age,zipcode; 完全なパラメータ形式は次のとおりです。 *attr-list*;https:/*server*/*fcc* 以下の例ではデフォルトの FCC を使用します。 AL=PASSWORD,SSN,age,zipcode; http://my.server.com/siteminderagent/ forms/login.fcc
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(*flag*) ユーザ認証情報が保存されることを示すには true (1) に、ユーザ認証情報が保存されないことを示すには false (0) に設定します。 デフォルトは 0 です。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

インパーソネーション テンプレート

インパーソネーション方式タイプに基づいて認証方式を設定するときにこの表を使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeImpersonation) インパーソネーション方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthimpersonate") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) ユーザ属性リストとフォーム認証情報コレクタ（FCC）の場所が含まれる文字列。 属性リストは AL= から始まり、リスト区切り文字としてカンマを使用する必要があります。また、セミコロンで終了する必要があります。たとえば、以下のようになります。 AL=Password,SSN,age,zipcode; 完全なパラメータ形式は次のとおりです。 *attr-list*;https:/*server/fcc* 以下の例ではデフォルトの FCC を使用します。 AL=PASSWORD,SSN,age,zipcode; http://my.server.com/siteminderagent/ forms/imp.fcc
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(*templateFlag*) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。

情報タイプ	値の割り当てと意味
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

RADIUS CHAP/PAP テンプレート

RADIUS CHAP/PAP 方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeRadiusChapPap) RADIUS CHAP/PAP 方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthchap") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) ユーザ ディレクトリ属性の名前が含まれる文字列。この属性は認証用のクリア テキスト パスワードとして使用されます。
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。

情報タイプ	値の割り当てと意味
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

RADIUS サーバ テンプレート

RADIUS サーバ方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeRadiusServer) RADIUS サーバ方式タイプです。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthradius") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) RADIUS サーバの IP アドレスおよびポートが含まれる文字列。たとえば、 123.123.12.12:1645 デフォルト UDP ポートは 1645 です。

情報タイプ	値の割り当てと意味
共有秘密キー	setSecret(*secret*) RADIUS サーバがクリア テキスト パスワードとして使用するユーザ属性。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

SafeWord HTML フォーム テンプレート

SafeWord HTML フォーム方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeSafeWordHTMLForm) SafeWord HTML フォーム方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 10 です。

情報タイプ	値の割り当てと意味
ライブラリ	setLibrary("smauthenigmahtml") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) フォーム認証情報コレクタの名前および場所が含まれている文字列。 この例はデフォルトの認証情報コレクタを示しています。 http://my.server.com/siteminderagent/forms/safeword.fcc
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

SafeWord テンプレート

SafeWord 方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeSafeWordServer) SafeWord 方式タイプ。

情報タイプ	値の割り当てと意味
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 10 です。
ライブラリ	setLibrary("smauthenigma") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter("") 空の文字列に設定します。 この方式には適用できません。
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

SAML Artifact テンプレート

SAML Artifact バインディングに基づいて認証方式を設定するときにこの表を使用します。この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeSAMLArtifact) SAML アーチファクト方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthsaml") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) 必須パラメータは以下のとおりです。 ■ Name。 アフィリエイトの名前です。 ■ RedirectMode。 SAML 認証情報コレクタがターゲット リソースにリダイレクトする方法です。 以下の数値のいずれかになります。 0. 意味： 302 データなし。 1. 意味： 302 Cookie データ。 2. 意味： サーバ リダイレクト。 ■ SRCID。 SAML アサーションを作成するサイト用の 20 バイトのソース ID。 ID は SAML プロデューサのサイトのプロパティ ファイル AMAssertionGenerator.properties に置かれています。 ■ AssertionRetrievalURL。SAML アサーション プロデューサのサイトからアサーションを取得するための URL。 ■ Audience。ポータルとアフィリエイトの間の許諾契約について説明するドキュメントの URI。 この値は SAML アサーションで指定されたオーディエンス値と比較されます。 ■ Issuer。 アサーションで指定された SAML 発行者。 ■ AttributeXPath。 SAML アサーションに対して実行される標準の XPath クエリ。クエリはユーザを検索する検索指定で代用されるデータを取得します。 ■ attribute。指定されたタイプのユーザ ディレクトリ内でユーザを探すための検索文字列。XPath クエリから返された値を挿入する場所を示すにはパーセント記号（%）を使用します。 たとえば、属性 **LDAP:uid=%s** を指定し、クエリから **user1** が返される場合、LDAP ディレクトリに使用される検索文字列は **uid=user1** です。 少なくとも 1 つの属性を指定する必要があります。 このパラメータ文字列の形式は以下のとおりです。 セミコロン（;）で名前/値ペアを区切ります。 この形式の例には LDAP および ODBC 属性が含まれます。 Name=*name*;RedirectMode=0|1|2;SRCID=*srcid*; AssertionRetrievalURL=*url*;Audience=*audience*; Issuer=*issuer*;AttributeXpath=*XPathQuery*; attribute=LDAP:*srch*Spc;attribute=ODBC:*srchSpc*

情報タイプ	値の割り当てと意味
共有秘密キー	setSecret(*secret*) アフィリエイト サイトのパスワード。
テンプレートですか?	setIsTemplate(*templateFlag*) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

SecurID HTML フォーム テンプレート

SecurID HTML フォーム方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeACEServerHTMLForm) SecurID HTML フォーム方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 15 です。
ライブラリ	setLibrary("smauthacehtml") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) ACE ID が保存されている属性の名前、フォーム認証情報コレクタ（FCC）がインストールされている Web サーバ、およびフォーム サポートで SecurID 認証の処理に必要なターゲット実行可能ファイルが含まれている文字列。 また、SSL 接続が必要かどうかを指定します。 形式 *attr*;https://*server*/*target* **注**: 「https」の「s」はオプションで、SSL 接続を行うかどうかを示します。 以下の例では、フォーム サポートで SecurID 認証を処理するためのデフォルトを使用します。 ace_id;https://my.server.com/siteminderagent/pwcgi/smpwservicescgi.exe
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

SecurID テンプレート

SecurID 方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeACEServer) SecurID 方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。値が大きいほど、その方式によって提供される保護の程度が大きくなります。デフォルトは 15 です。
ライブラリ	setLibrary("smauthace") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) ACE サーバ ユーザ ID を含む認証ユーザ ディレクトリの属性が含まれる文字列。
共有秘密キー	setSecret("") 空の文字列に設定します。この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。

情報タイプ	値の割り当てと意味
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

smauthetsso 認証方式

smauthetsso 認証方式は、SiteMinder X.509 証明書方式に似ていますが、X.509 認証情報の代わりに eSSO Cookie を認証情報として使用します。

この方式が cookieorbasic または cookieorforms モードのいずれかに対して設定され、eSSO Cookie とログイン名およびパスワード認証情報の両方がそれに渡される場合、eSSO Cookie は無視され、ログイン名とパスワードは SiteMinder へのユーザを認証するために使用されます。

eSSO Cookie が唯一の認証情報である場合、認証方式では、Cookie を検証し、そこからユーザ識別名（DN）を抽出するために、ETWAS API を使用して、設定された eSSO ポリシー サーバに接続します。

この表は方式タイプ［カスタム］に基づく smauthetsso 認証方式を設定する場合に使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeCustom) 方式タイプ［カスタム］を使用します。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 0 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthetsso") この認証方式に使用するライブラリの名前。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) 以下のようにセミコロンによって分離され、順序付けられたトークンのセット。 <*Mode*>[; <*Target*>]; <*Admin*>; <*eTPS_Host*> 文字列を読み取りやすいようにスペースを追加できます。 <*Mode*> は、認証方式が受け入れる認証情報のタイプを指定します。以下の値が可能です。 ■ Cookie -- SSO Cookies のみが受け入れ可能です。 ■ cookieorbasic -- SSO Cookie が提供されない場合、ログイン名とパスワードは基本認証の使用により要求されます。 ■ cookieorforms -- SSO Cookie が提供されない場合、ログイン名とパスワードは基本認証の使用により要求されます。 <*Target*> は cookieorforms モードでのみ有効です。これは標準の HTML フォーム認証方式用の Target フィールドと同一です。 <*Admin*> は、ポリシー サーバの管理者用ログイン ID を指定します。この管理者のパスワードは［共有秘密キー］フィールドで指定されています。 <*eTPO_Host*> は、ポリシー サーバがインストールされているマシンの名前を指定します。 SiteMinder は、SSO Cookie の検証を要求できるように、<*eTPS_Host*> 上でポリシー サーバに対して自身を <*Admin*> として認証します。 例： "cookie; SMPS_sso; myserver.myco.com" "cookieorforms; /siteminderagent/forms/login.fcc; SMPS_sso; myserver.myco.com"
共有秘密キー	setSecret(*secret*) ［パラメータ］フィールドで指定されたポリシー サーバ管理者のパスワード。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。

情報タイプ	値の割り当てと意味
管理者によって使用されますか?	setIsUsedByAdmin(*flag*) 管理者の認証に方式を使用できることを指定するには true (1)、または管理者の認証に方式を使用できないことを指定するには false (0) を指定します。 デフォルトは 0 です。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

TeleID テンプレート

TeleID 方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeEncotone) TeleID 方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 15 です。
ライブラリ	setLibrary("smauthencotone") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter("") 空の文字列に設定します。 この方式には適用できません。

情報タイプ	値の割り当てと意味
共有秘密キー	setSecret(*seed*) 暗号化シード。 SiteMinder はハードウェア トークンを初期化するための暗号化シードとしてこの値を使用します。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(1) true (1) に設定します。管理者を認証するために方式を使用できます。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(1) true (1) に設定します。方式は RADIUS エージェントと共に使用できます。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

Windows 認証テンプレート

Windows 認証（以前は NTLM と呼ばれていた）方式タイプに基づいて統合 Windows 認証方式を設定する場合はこの表を使用します。 この方式タイプは WinNT または Active Directory ユーザ ストアに対して認証を行うために使用されます。

Active Directory は*混在モード*または*ネイティブ モード*で実行されるように設定できます。 混在モードで実行される場合、Active Directory は WinNT スタイル認証をサポートします。 ネイティブ モードでは、Active Directory は LDAP スタイル検索のみをサポートします。

この認証方式は混合モードまたはネイティブ モードのいずれかをサポートします。

この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeNTLM) Windows 認証（NTLM）方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthntlm") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) *param* の値は、以下のようにこの方式で実行する認証のスタイルを決定します。 **NTLM 認証**（混合モードで実行される WinNT または Active Directory 用） 形式 *iis-web-server-url*/*path-to-ntc-file* この形式では、*iis-web-server-url* は、リダイレクトのターゲットである IIS Web サーバの名前です。また、*path-to-ntc-file* は WinNT 認証情報を収集する .ntc ファイルの場所です。 例： http://myiiswebserver.mycompany.com/ siteminderagent/ntlm/creds.ntc 指定されたサーバに SiteMinder Web エージェントをインストールする必要があります。 デフォルトでは、Web エージェントのインストールによって、NTLM 認証情報収集用の仮想ディレクトリが作成されます。 **Windows 認証（ネイティブ モードで実行される Active Directory 用）** この認証スタイルの *param* では、リダイレクト URL の先頭に LDAP フィルタが追加されます。フィルタと URL はセミコロン（;）によって区切ります。 例： cn=%{UID},ou=Users,ou=USA,dc=%{DOMAIN}, dc=mycompany,dc=com;http:// myiiswebserver.mycompany.com/ siteminderagent/ntlm/creds.ntc SiteMinder は、LDAP フィルタを使用してブラウザ/Web エージェントから受け取った認証情報を LDAP DN または検索フィルタにマップします。
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。

情報タイプ	値の割り当てと意味
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) WinNT、および混在モードで実行される Active Directory の場合は、このプロパティを true (1) にする必要があります -- パスワード確認を無視します。 ネイティブ モードで Active Directory を実行している場合は、true (1) に設定してスワード確認を無視するか、false (0) に設定してパスワードを確認します。 デフォルトは 0 です。

X.509 クライアント証明書および基本テンプレート

X.509 クライアント証明書および基本方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeX509ClientCertAndBasic) X.509 クライアント証明書および基本方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 15 です。
ライブラリ	setLibrary("smauthcert") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) SSL サーバのドメインまたは IP アドレス、および SSL 認証情報コレクタ（SCC）の名前とパスが含まれる文字列。 サーバは、SSL 接続を介してユーザの X.509 証明書をリダイレクトします。 形式 https://*server:port*/*SCC*?cert+basic 以下の例ではデフォルトの SCC を使用します。 https://my.server.com:80/siteminderagent/ cert/smgetcred.scc?cert+basic
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

X.509 クライアント証明書およびフォーム テンプレート

X.509 クライアント証明書およびフォーム方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeX509ClientCertAndForm) X.509 クライアント証明書および HTML フォーム方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 15 です。
ライブラリ	setLibrary("smauthcert") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) SSL サーバのドメインまたは IP アドレス、およびフォーム認証情報コレクタ（FCC）の名前とパスが含まれる文字列。 サーバは、SSL 接続を介してユーザの X.509 証明書をリダイレクトします。 形式 https://*server:port*/*FCC*?cert+forms 以下の例ではデフォルトの FCC を使用します。 https://my.server.com:80/siteminderagent/certoptional/forms/login.fcc?cert+forms
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートでないことを示すには 0、方式がテンプレートの場合には 1 を設定します。 デフォルトは 0 です。
管理者によって使用されますか?	setIsUsedByAdmin(0) 0 に設定 -- 方式は管理者を認証するためには使用されません。

情報タイプ	値の割り当てと意味
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示すには 0 に設定します。
RADIUS ですか?	setIsRadius(0) 0 に設定 -- 方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには 1、パスワードを確認するには 0 に設定します。 デフォルトは 0 です。

X.509 クライアント証明書または基本テンプレート

X.509 クライアント証明書*または*基本方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeX509ClientCertOrBasic) X.509 クライアント証明書または基本方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthcert") この方式タイプ用のデフォルト ライブラリ。

情報タイプ	値の割り当てと意味
パラメータ	setParameter(*param*) 以下の情報が含まれる文字列。 SSL 接続を確立するためのサーバ。 このサーバは、SSL 接続を介してユーザの X.509 証明書をリダイレクトします。 SSL 認証情報コレクタ（SSC）の名前およびパス。 SSL を介して基本認証を使用している場合は、以下の 2 種類の情報も指定します。 基本認証用の SSL 接続を確立するために使用される SSL サーバの完全修飾名。 SSL 認証情報コレクタ（SSC）の名前およびパス。 https://*SSLserver:port*/*SCC*?certorbasic; [https://*BasicServer*/*SCC*]
	以下の例ではデフォルトの SCC 値を使用しています。 https://my.SSLserver.com:80/siteminderagent/certoptional/smgetcred.scc?certorbasic; https://my.BasicServer.com/siteminderagent/nocert/smgetcred.scc
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには true (1)、パスワードを確認するには false (0) に設定します。 デフォルトは 0 です。

X.509 クライアント証明書またはフォーム テンプレート

X.509 クライアント証明書 *または*フォーム方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeX509ClientCertOrForm) X.509 クライアント証明書または HTML フォーム方式タイプ。
説明	setDescription(*description*) 認証方式の説明。
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthcertorform") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) 以下の情報が含まれる文字列。 ■ SSL 接続を確立するためのサーバ。 このサーバは、SSL 接続を介してユーザの X.509 証明書をリダイレクトします。 ■ SSL およびフォーム認証情報コレクタ（SFCC）の名前およびパスです。 SSL を介して代替的なフォームベースの認証を使用している場合は、以下の 2 種類の情報も指定します。 ■ 認証用の SSL 接続を確立するために使用される SSL サーバの完全修飾名。 ■ フォーム認証情報コレクタ（FCC）の名前およびパスです。 https://*SSLserver:port*/*SFCC*?certorform; [https://*BasicServer*/*FCC*]

情報タイプ	値の割り当てと意味
	以下の例ではデフォルトの SCC 値を使用しています。 https://my.SSLserver.com:80/siteminderagent/certoptional/forms/login.sfcc?certorform; https://my.BasicServer.com/siteminderagent/forms/login.fcc
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートでないことを示すには 0、方式がテンプレートの場合には 1 を設定します。 デフォルトは 0 です。
管理者によって使用されますか?	setIsUsedByAdmin(0) 0 に設定 -- 方式は管理者を認証するためには使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示すには 0 に設定します。
RADIUS ですか?	setIsRadius(0) 0 に設定 -- 方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(*flag*) パスワード チェックを無視するには 1、パスワードを確認するには 0 に設定します。 デフォルトは 0 です。

X.509 クライアント証明書テンプレート

X.509 クライアント証明書方式タイプに基づいて認証方式を設定するときにこのテーブルを使用します。 この表に記載されている Java メソッドは、SmScheme クラスにあります。

情報タイプ	値の割り当てと意味
方式タイプ	setType(TypeX509ClientCert) X.509 クライアント証明書方式タイプ。
説明	setDescription(*description*) 認証方式の説明。

情報タイプ	値の割り当てと意味
保護レベル	setLevel(*value*) 1 ～ 1000 の値。 値が大きいほど、その方式によって提供される保護の程度が大きくなります。 デフォルトは 5 です。
ライブラリ	setLibrary("smauthcert") この方式タイプ用のデフォルト ライブラリ。
パラメータ	setParameter(*param*) SSL 接続の確立に関与するサーバのドメインまたは IP アドレス、および SSL 認証情報コレクタ（SCC）の名前およびパスが含まれる文字列。 サーバは、SSL 接続を介してユーザの X.509 証明書をリダイレクトします。 形式 https://*server*/*SCC*?cert 以下の例ではデフォルトの SCC 値を使用しています。 https://my.server.com/siteminderagent/ cert/smgetcred.scc?cert
共有秘密キー	setSecret("") 空の文字列に設定します。 この方式には適用できません。
テンプレートですか?	setIsTemplate(0) 方式がテンプレートではないことを示すには、false (0) に設定します。
管理者によって使用されますか?	setIsUsedByAdmin(0) false (0) に設定します。方式は管理者の認証には使用されません。
認証情報を保存しますか?	setAllowSaveCreds(0) ユーザ認証情報が保存されないことを示す false (0) に設定します。
RADIUS ですか?	setIsRadius(0) false (0) に設定します。方式は RADIUS エージェントと共には使用されません。
パスワード チェックを無視しますか?	setIgnorePwCheck(1) true (1) に設定して、パスワード チェックを無視します。

パフォーマンス考慮事項

SmRealm オブジェクトの以下のプロパティがデフォルトで true に設定されます。

- *PropProcessAuthEvents*。true の場合、認証イベント処理が発生します。
- *PropProcessAzEvents*。 true の場合、許可イベント処理が発生します。

認証および許可イベント処理はパフォーマンスに影響します。 レルム内のルールが、認証または許可イベントによってトリガされない場合、関連するプロパティを false に設定します。

第 5 章: Java 認証および許可ガイド

このセクションには、以下のトピックが含まれています。

すべてのカスタム クラスの設定

以下の設定情報は、Java 認証 API および Java 許可 API で実装されるすべてのカスタム認証方式およびアクティブな式に適用されます。

- ライブラリ名は常に smjavaapi です。
- パラメータ フィールドで、最初の項目は、認証 API または許可 API で実装されたカスタム クラスの名前である必要があります。以下のようになります。
  - 認証方式で、ベース インターフェース SmAuthScheme から実装したクラスの名前を指定します。 クラス名には、以下のように完全修飾パッケージ名が含まれる必要があります。

    com.myorg.sdk.myclass
  - アクティブなポリシー、アクティブなルールおよびアクティブなレスポンスで、ベース インターフェース ActiveExpression から実装したクラスの名前を指定します。

- パラメータがクラス名の後にパラメータ フィールドで指定される場合、クラス名はスペース文字によってパラメータ リストから分離されます。

  SiteMinder がカスタム クラスのインスタンス内のメソッドを呼び出す場合、指定されたパラメータを渡します。 クラス名は渡されません。パラメータは単一の文字列として渡されます。 文字列に複数のパラメータが含まれる場合、パラメータはカスタム クラスが要求する任意の方法で区切ることができます。
- パラメータ フィールドで指定されたクラス ファイルは、JVMOptions.txt 設定ファイルの classpath コマンドで参照される必要があります。 このファイルは、SiteMinder インストール パス内の Netegrity/SiteMinder/Config に置かれています。
- アクティブな式では、関数名（つまり、smjavaapi ライブラリ ファイルのエントリ ポイント）は常に JavaActiveExpression です。

認証と許可用のカスタム Java クラス

カスタム認証または許可クラスの実装および展開の基本手順は以下のとおりです。

1. 認証または許可 API および共通クラスを使用してカスタムの認証または許可クラスを実装します。
2. ポリシー サーバ マシン上のカスタム クラスまたは jar ファイルを展開し、JVMOptions.txt ファイルの classpath コマンドで場所を指定します。 このファイルは、SiteMinder インストール パス内の Netegrity/SiteMinder/Config に置かれています。
3. ポリシー サーバ ユーザ インターフェース内のカスタム認証または許可機能を設定します。

必要なライブラリ ファイル

カスタム認証および許可クラスはすべて同じライブラリ ファイル smjavaapi を使用します。このライブラリ ファイルはポリシー サーバに含まれています。このライブラリ ファイルは変更する必要はありません。カスタムの認証または許可クラスを設定する場合に単に参照します。

共有情報

カスタム認証および許可オブジェクトは、たとえばオブジェクト インスタンス間の状態を維持するなどの場合によっては、お互いに要求固有の情報を通信する必要があります。 これらのオブジェクトは、ApiContext を通して取得される AppSpecificContext を通して情報を共有できます。 ApiContext は、認証および許可オブジェクトの両方に渡される共通クラスの 1 つです。

AppSpecificContext によって共有された情報には要求のみのスコープがあります。 たとえば、認証要求のコンテキストで実行されるカスタム オブジェクトは、許可要求のコンテキストで実行されるオブジェクトと情報を交換できません。

共通クラス

以下のクラスは、認証 API および許可 API の両方によって使用されます。 これらのクラスが提供するサービスには以下が含まれます。

- ポリシー サーバへのログ記録、トレース、およびエラー メッセージの送信
- 情報共有のためのカスタム認証および許可オブジェクト機構の提供
- ユーザ コンテキスト情報を認証および許可オブジェクトに対して利用可能にすること

以下の表に共通クラスの概要を示します。

クラス	説明
APIContext	ログ記録、トレース、およびエラー メッセージがポリシー サーバに送信されることを可能にします。
AppSpecificContext	カスタム認証および許可オブジェクトが情報を共有することを可能にするメソッドを提供します。
SmJavaApiException	カスタム認証および許可オブジェクトに例外機能を提供します。

クラス	説明
UserContext	カスタム オブジェクトがユーザ ディレクトリのユーザに関する情報を設定し取得することを可能にします。 情報にはユーザ属性、およびユーザと関連付けられたディレクトリ属性が含まれます。 ユーザ ディレクトリ属性を設定および取得する方法は、isUserContext() が true を返す場合にのみ利用可能です。

Java を使用したカスタム認証方式の作成

認証方式は、ユーザの認証情報を収集してユーザを識別する方法を提供します。

ポリシー サーバには各種の標準的な認証方式が含まれます。その中には、ユーザ名とパスワードを使った基本認証や HTML フォームベースの認証からデジタル証明書やトークン認証まで含まれています。

ポリシー サーバに含まれる標準認証方式がサイトで必要な認証機能を提供しない場合、カスタム認証方式の作成に Java 認証 API を使用することができます。

認証 API 内のクラスおよびインターフェース

Java 認証 API 内の基本インターフェースは SmAuthScheme です。 Java 認証 API で作成されたカスタム認証方式はすべてこのインターフェースを実装する必要があります。

SmAuthScheme メソッド

SiteMinder は、基本インターフェース SmAuthScheme 内の以下のメソッドを呼び出します。

メソッド	説明
authenticate()	カスタム認証を実行し、認証結果を返します。 SiteMinder は、少なくともユーザ コンテキストの確立時とユーザの認証情報の認証時の 2 回このメソッドを呼び出します。
init()	認証方式が必要とするあらゆる初期化手順を実行します。 認証方式がロードされるときに SiteMinder はこのメソッドを各認証方式インスタンスに対して 1 度ずつ呼び出します。
query()	SiteMinder が *reason* パラメータ（オブジェクト SmAuthQueryCode）に渡す値に応じて、以下の種類の情報のいずれかを SiteMinder に提供します。 認証方式のバージョンおよび説明。 SiteMinder がユーザから収集する必要のある認証情報。オプションで、認証情報を収集する必要のあるサイトの URL。
release()	認証方式が必要とするあらゆる要約手順を実行します。SiteMinder がシャットダウンしているとき、SiteMinder は各認証方式インスタンスに対してこのメソッドを 1 度呼び出します。

認証 API 内の他のクラス

以下のクラスが SmAuthScheme 基本インターフェースと共に使用されます。

クラス	説明
SmAuthenticationContext	authenticate() に渡される以下のコンテキスト クラスが含まれます。 APIContext UserContext UserCredentialsContext このオブジェクト内の set... メソッドは SiteMinder に情報を直接渡します。

クラス	説明
SmAuthenticationResult	authenticate() の呼び出しの後に SiteMinder に返されるステータスおよび理由コードが含まれます。
SmAuthQueryCode	認証方式に SiteMinder が作成する要求のタイプ（バージョン番号および説明、または SiteMinder が収集する必要がある認証情報に関する情報）が含まれます。 SiteMinder は、このオブジェクトを query() 内の認証方式に渡します。
SmAuthQueryResponse	認証に必要な認証情報の種類を指定する定数が存在する場合はその定数が含まれます。 また、テキストメッセージがユーザへの表示のために SiteMinder に返されることを可能にします。 オプションで、認証方式は認証情報を収集する必要のある URL を指定するために setResponseBuffer() を呼び出すことができます。 このオブジェクト内の set... メソッドは SiteMinder に情報を直接渡します。
SmAuthStatus	SMAUTH_ACCEPT と SMAUTH_REJECT などの 認証 API ステータス コードが含まれます。 このオブジェクトは、SmAuthenticationResult 内で SiteMinder に対して返されることが可能です。 また、認証方式の init()、query()、および release() メソッドに対する戻り値タイプです。
UserCredentialsContext	ユーザ コンテキストが確立されたユーザ ディレクトリからの認証情報およびその他の情報が含まれています。 このオブジェクトは、authenticate() に渡された SmAuthenticationContext オブジェクトに含まれます。

SiteMinder が Java のカスタム認証方式をロードする方法

ユーザ認証が Java API で作成されたカスタム認証方式に対して発生する場合、SiteMinder は以下をロードすることでカスタム方式をロードします。

- ポリシー サーバにインストールされる標準ライブラリ ファイル smjavaapi
- SmAuthScheme から実装されたカスタム クラスのインスタンス

SiteMinder が認証処理を初期化する方法

方式がロードされた直後に、SiteMinder は SmAuthScheme から実装されたカスタム クラス内の以下のメソッドを呼び出します。

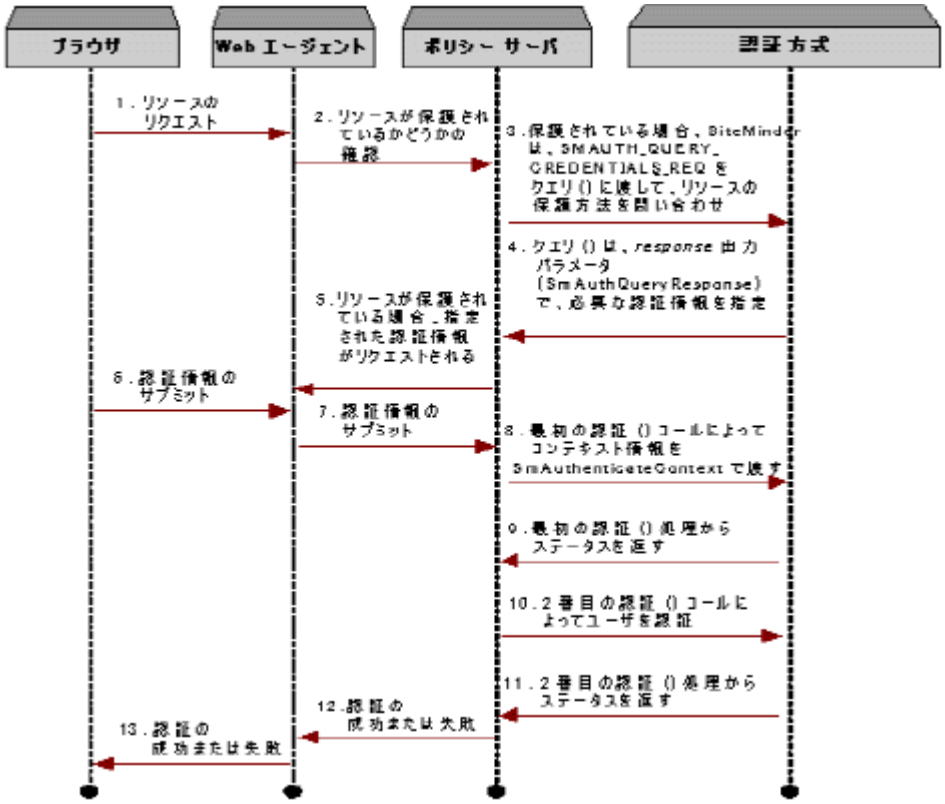
- query()。 SiteMinder は *request* パラメータ内に SMAUTH_QUERY_DESCRIPTION を渡し、認証方式がバージョン番号および Java 認証 API の説明を渡すことを要求します。

  注: SiteMinder が query() 内に SMAUTH_QUERY_CREDENTIALS_REQ を渡す場合、SiteMinder は認証方式が必要な認証情報の種類を指定することを要求しています。 その後、SiteMinder は指定された認証情報を収集します。

- init()。 SiteMinder は、認証方式がポリシー サーバに設定されたときに定義されたパラメータ文字列および共有秘密キーを渡します。 その後、方式は必要な任意の初期化手順を実行します。

ユーザ認証情報の認証

以下の図は、認証中に発生する主要なアクティビティを示しています。



サポートされた認証情報

Java 認証 API は、以下の一般的なタイプの認証情報要件に基づいた認証をサポートします。

- ユーザ名/パスワード
- X.509 証明書
- カスタム ユーザ属性

オブジェクト SmAuthQueryResponse 内の setResponseBuffer() メソッドによって必要な認証情報を指定します。 このオブジェクトには、必要とされる特定の認証情報、または認証情報が必要ないことを示す一連の定数が含まれます。

ユーザの特定および認証

認証処理には、*ユーザの特定*および*ユーザの認証*の 2 つの段階が含まれます。

ユーザを認証できる前に、ユーザのプロファイル情報をユーザ ストアから取得し、ユーザの保存された認証情報をログイン時に提供された認証情報と比較できるようにします。ユーザ ストア（LDAP ユーザ ディレクトリまたは ODBC データベースなど）内のユーザの検索は「*ユーザの特定*」と呼ばれます。SiteMinder または認証方式のいずれかがユーザの特定を実行できます。

SiteMinder は、SmAuthScheme.authenticate() を特定処理段階に少なくとも 1 回、および認証処理段階に少なくとも 1 回呼び出します。

- 特定処理中に、特定が発生したディレクトリにつき 1 回ずつ呼び出されます。
- 認証中に、ディレクトリで見つかるユーザにつき 1 回ずつ呼び出されます。

標準的な手順を以下に示します。

1. **ユーザ ログイン**。 ユーザは、認証目的のログイン ID（jsmith など）を提供します。
2. **特定段階**。データ ストア内のユーザ検索が開始する前に、提供されたログイン ID に基づいて完全な DN または検索式を構築する必要があります。 たとえば、ログイン ID が jsmith である場合、ユーザ ストアを検索するために使用される DN は以下のように作成できます。

   ```
   uid=jsmith,ou=marketing,o=myorg.org
   ```

   LDAP ユーザ ディレクトリの検索に LDAP 検索式も使用することができます。また、ODBC データベースの検索には SQL クエリが使用されます。たとえば、以下のようになります。

   ```
   (&(objectclass=inetOrgPerson)(uid=jsmith))
   ```

   ```
   select Name from SmUser where Name = 'jsmith'
   ```

   ODBC データベースに保存された LDAP DN または ID が、別のパスワードを持つ別のユーザに適用される場合があるとすれば、複数の結果が考えられます。
3. **認証段階**。 カスタム認証方式は、特定された各ユーザの既知の認証情報をログイン時に提供された認証情報と比較します。たとえば、提供されたパスワードのハッシュをユーザ ストア内のハッシュと比較します。

ユーザの特定

SiteMinder は、ユーザの特定段階の最初に authenticate() を呼び出します。

SiteMinder またはカスタム認証方式のいずれかで、ユーザが特定できます。認証方式は、以下の組み合わせによってユーザの特定を実行したかどうかを示します。

- 以下に表示されたステータス コードの 1 つ
- 値が SmAuthenticationContext の setUserText() メソッド内で SiteMinder に渡されるかどうか

ステータス コードは SmAuthStatus オブジェクトで設定されます。 このオブジェクトは SmAuthenticationResult コンストラクタの *status* パラメータで渡されます。 authenticate() から SmAuthenticationResult が返されます。

- SMAUTH_NO_USER_CONTEXT

  認証方式は、ユーザの特定を SiteMinder に依頼します。

  このステータス コードが返される場合、認証方式は setUserText() メソッドを通して空の文字列も返します。 SiteMinder は、エージェントからログイン ID を取得し、ログイン ID および [SiteMinder ユーザ ディレクトリ プロパティ] ダイアログ ボックスで定義された情報に基づいて DN または検索式を作成します。そしてユーザ ストア内のユーザを調べることにより、ユーザが特定されます。

- SMAUTH_SUCCESS

  認証方式は、ユーザの特定を SiteMinder に依頼します。

  認証方式は、setUserText() を通して SiteMinder にログイン ID を渡します。SiteMinder は、その値を使用して DN または検索式を作成し、ユーザ ストア内のユーザを特定します。 この方法により、認証方式は SiteMinder がユーザの特定前にログイン ID を変更することができます。

  注: 認証方式が setUserText() に空の文字列を渡す場合、SiteMinder はエージェントによって提供されるログイン ID を使用します（リターン コード SMAUTH_NO_USER_CONTEXT と同じ動作）。

- SMAUTH_SUCCESS_USER_DN

  認証方式は、完全な DN または検索式を構築し、ユーザ ストア内でユーザを検索することでユーザの特定を実行します。 認証方式は、ユーザの完全な DN または ODBC データベース ID を SiteMinder の setUserText() に渡します。 setUserText() には 1 つの DN またはデータベース ID のみを渡すことができます。

- SMAUTH_ATTEMPT。

  ユーザがディレクトリに見つかりません。

- SMAUTH_FAILURE

  エラー状態が存在する場合に返されます。 エラー テキストは setUserText() メソッドによって SiteMinder に返されます。

ユーザ認証

この段階中に、SiteMinder は、特定中にユーザ コンテキストが確立された後に認証方式が提供された認証情報を検証できるように authenticate() を再び呼び出します。 メソッドは以下のいずれかのステータス コードを設定します。

- SMAUTH_ACCEPT。 ユーザが認証されます。
- SMAUTH_REJECT。 ユーザは認証されません。
- SMAUTH_CHALLENGE。 ユーザは認証情報を要求されます。 方式は、setUserText() メソッドを通して SiteMinder にチャレンジ メッセージを渡します。 また、理由コードは authenticate() によって返される SmAuthenticationResult オブジェクトで提供される必要があります。
- SMAUTH_FAILURE。 エラー状態が発生しました。 エラー テキストは SiteMinder に setUserText() 内で渡されます。

リダイレクト

認証方式は、ポリシー サーバがエージェントにリダイレクトを実行するよう指示するようにできます。 リダイレクト機能を認証方式に構築する方法は以下のとおりです。

- 以下でリダイレクト URL を指定します。

  ```
  SmAuthenticationContext.setErrorText()
  ```

- authenticate() から返されるように SmAuthenticationResult オブジェクトを作成する場合、コンストラクタの *reason* パラメータ内に REASON_ERROR_MESSAGE_IS_REDIRECT を指定します。

認証イベント

認証結果は SiteMinder のイベントに結び付けられます。 認証イベントが、ユーザが認証されるレルムで有効になっている場合、SiteMinder は OnAuthAccept、OnAuthReject、OnAuthAttempt、および OnAuthChallenge ルールに結び付けられるオプションのポリシーを評価します。 これらのポリシーを、ユーザのアイデンティティに基づいてカスタム レスポンスを返すように設定するには、認証結果に基づいてユーザを別の場所にリダイレクトするか、外部データベース内でユーザ データを更新します。

SAML/WS-フェデレーション認証方式の拡張

SiteMinder SAML （1.x および 2.0）/WS-フェデレーション認証方式は、レスポンス メッセージを処理します。 ビジネス上の理由で、たとえばレスポンスをさらに処理するために追加の手順が必要となる場合があります。メッセージ コンシューマ拡張 API は、認証処理中に SAML または WS-フェデレーション レスポンスを 2 つの方法を使用して詳細に作成することを可能にするインターフェースを定義します。

- ユーザの特定中に失敗理由の詳細をレポートする方法
- ユーザ認証情報の検証をカスタマイズする方法

SiteMinder Java MessageConsumerPlugin API は、メッセージ コンシューマ拡張（MCE）インターフェースを実装します。 自分の要件に基づいてコードを書き、カスタム プラグインを SiteMinder に統合することができます

MessageConsumerPlugin には、以下の 4 つのメソッドが含まれています。

メソッド	説明
init()	プラグインが必要とするあらゆる初期化手順を実行します。 プラグインがロードされると、SiteMinder は各プラグイン インスタンスに対してこのメソッドを 1 回コールします。
release()	プラグインが必要とするあらゆる要約手順を実行します。 SiteMinder がシャットダウンしているとき、SiteMinder は各プラグイン インスタンスに対してこのメソッドを 1 度呼び出します。
postDisambiguateUser()	認証方式によって実行できない場合に、ユーザの特定を実行する処理を提供します。または、新規フェデレーション ユーザに対するデータをユーザ ストアに追加するための処理を提供します。 このメソッドが復号されたアサーションを受け取ることに注目してください。 復号されたアサーションは、キー「_DecryptedAssertion」の下の MCP に渡されるプロパティ マップに追加されます。
postAuthenticateUser()	ポリシー サーバ処理結果が成功または失敗に関係なく、アサーション処理の最終結果を決定する任意の追加のコードを提供します。

SiteMinder から、Message Consumer プラグイン クラスの以下のサンプルが提供されます。

- MessageConsumerPluginSample.java in <install-path>¥sdk¥samples¥messageconsumerplugin
- MessageConsumerSAML20.java in <install-path>¥sdk¥samples¥authextensionsaml20

MessageConsumerPlugin のロール

以下のリストは、ユーザ認証の詳細な過程における MessageConsumerPlugin について説明しています。

1. フェデレーション Web サービス（FWS）アプリケーションは、ユーザ認証に対する要求をポリシー サーバに転送します。
2. ポリシー サーバは、ユーザの特定を実行するために認証方式を起動します。
3. 認証方式は、ユーザの特定を以下のように実行します。
   a. 認証方式メタデータは、ポリシー ストアから取得されます。
   b. 認証方式は、LoginID の取得を試みます。 LoginID が見つからない場合、認証方式は手順 4 で説明される MessageConsumerPlugin を起動します。
   c. LoginID が正常に取得されたら、認証方式は現在のユーザ ディレクトリを事前定義された SearchSpec で検索します。 ユーザが見つからない場合、postDisambiguate() メソッドが手順 4 で説明されるように呼び出されます。 ユーザが見つかった場合、手順 6 以降で説明されているように、ポリシー サーバは認証情報の検証を実行します。
   d. 認証方式がユーザ ストア SearchSpec を提供しない場合、ポリシー サーバ コアはユーザ ディレクトリ オブジェクトで定義される検索文字列でユーザを検索します。MessageConsumerPlugin は呼び出されません。
4. postDisambiguateUser() メソッドは、特定の LoginID が存在するかどうか判断するためにユーザ ディレクトリを検索します。結果は認証方式に返されます。複数のユーザ ディレクトリが設定される場合、メソッドは数回呼び出される場合があります。 このメソッドは、アサーションからユーザ ストアに新規フェデレーション ユーザに対するデータを追加するために使用することもできます。

5. ポリシー サーバ、認証方式、またはプラグインによってユーザが正常に特定された場合、ポリシー サーバはユーザ DN をポリシー サーバに返し、認証情報の検証に進みます（手順 8 以降）。
6. ポリシー サーバ、認証方式、または MessageConsumerPlugin によってユーザがこのユーザ ディレクトリに対して正常に特定されない場合、FWS アプリケーションは次のユーザ ディレクトリを確認し、認証情報の検証に進む前に手順 2 から 6 を繰り返します。
7. ユーザが特定された場合、ポリシー サーバは認証方式を再度呼び出して、認証要求に対する適切な認証情報がユーザに存在するかどうかを判断します。認証方式は、レスポンス メッセージを受け入れることができるかどうかを判断します。
8. 認証方式が、レスポンス メッセージで特定されたユーザの認証を試行した場合、ポリシー サーバは MessageConsumerPlugin から postAuthenticateUser() メソッドを呼び出します。ポリシー サーバ コアがユーザの特定を実行するときでも、ユーザが特定されるとポリシー サーバは常にこのメソッドを呼び出します。
9. 実装に必要なフェデレーション認証情報の検証に対する他の手順を追加するために postAuthenticateUser() を使用できます。
10. 最終結果は、認証方式によってポリシー サーバへ再度渡されます。
11. 必要に応じて、FWS アプリケーションはどんな失敗も処理し、ユーザを適切な URL にリダイレクトすることができます。

API コマンドの使用

Java 許可 API を使用すると、保護されたリソースへのアクセスを制御するためのカスタム機能を実装できます。

機能は、ポリシー サーバのアクティブな式で参照されるカスタム Java クラスによって提供されます。*アクティブな式*は、以下のポリシー サーバ オブジェクトに表示される、一連の変数定義です。

- アクティブ ポリシー -- 外部のビジネス ロジックに基づいて動的な認可を提供するポリシー。

  たとえば、ユーザが LDAP ディレクトリ内で特定の組織ユニット（ou）に属する場合に true を返すカスタムの Java クラスを実装するとします。 ou は、アクティブな式のパラメータ（*param*）フィールド内でカスタム Java クラスに返されます。

- アクティブなレスポンス -- カスタム Java クラスから返されるレスポンス。アクティブ レスポンスの使用は、ユーザに固有の権限情報を定義できる 1 つの方法です。

  たとえば、ユーザがアクティブな式の［*param*］フィールドに渡された ou に属する場合にユーザの共通名（cn）を返すアクティブなレスポンスを定義するとします。

- アクティブ ルール -- 外部のビジネス ロジックに基づいて動的な認可を提供するルール。

  たとえば、レルムを参照する権限を持つディレクトリ管理者など、ユーザがグループのメンバである場合に true を返すカスタム Java クラスを定義するとします。グループ名がアクティブな式の［*param*］フィールド内で Java クラスに渡されます。

アクティブな式

アクティブな式は、以下の構文を使用してポリシー サーバ ユーザ インターフェースに構築されます。

```
<@ lib=<lib-spec> func=<func-spec> param=<func-params>@>
```

Java 許可 API に基づいたアクティブな式には以下の必須フィールドがあります。

- *lib* には共有ライブラリ名 smjavaapi が含まれます。このライブラリは、*param* 内のカスタム Java クラスを参照するすべてのアクティブな式と共に使用されます。
- *func* には関数名 JavaActiveExpression が含まれます。 この関数は smjavaapi ライブラリ用のエントリ ポイントです。 *param* 内のカスタム Java クラスを参照するすべてのアクティブな式と共に使用されます。
- *param* には以下の情報が含まれます。
  - カスタム Java クラスの名前
  - オプションで、クラスのインスタンスに渡す任意のパラメータ

ポリシー サーバ ユーザ インターフェースでアクティブなポリシー、ルール、またはレスポンスを設定する場合にアクティブな式を定義します。

アクティブな式の実行

SiteMinder がカスタム Java クラスを参照するアクティブな式を検出した場合、以下のタスクを実行します。

- 共有ライブラリおよびアクティブな式で指定されるカスタム Java クラスのインスタンスをロードします。
- アクティブな式で指定されたライブラリ関数を呼び出します。
- カスタム Java のインスタンスに、オプションのパラメータ文字列と以下のコンテキスト オブジェクトを渡します。
  - APIContext
  - RequestContext
  - UserContext
- Java クラスのインスタンスはカスタム機能を実行し、SiteMinder に結果を返します。 結果がカスタム Java クラスの invoke() メソッドから返されます。

アクティブな式の結果の解釈

SiteMinder は、カスタム Java クラスのインスタンスによって返された結果を、Java クラスを参照するアクティブな式のタイプに基づいて以下のように解釈します。

- アクティブなポリシー - 返される結果が空の文字列である場合または例外がスローされる場合、許可は拒否されます。

  返される結果が文字列（大文字と小文字は区別されません）FALSE、F、または 0 のいずれかに一致する場合、ポリシーは起動されません。

  ポリシーは他のどの結果によっても起動されます。

- アクティブ ルール - 返される結果が空の文字列である場合または例外がスローされる場合、以下の動作が発生します。

  - アクセス許可ルールでは、ルールが起動しません。
  - アクセス拒否ルールでは、ルールが起動します。

  これ以外の場合は、動作はアクティブ ポリシーと同じです。

- アクティブ レスポンス - 結果はレスポンス属性に対応する文字列です。SiteMinder が結果文字列を解釈する方法は、ポリシー サーバ ユーザ インターフェースによって指定されるレスポンス属性によって決定されます。 たとえば、以下のようになります。

  - WebAgent-OnReject-Redirect。 このレスポンス属性では、SiteMinder は結果文字列が URL など、リソースへのアクセスを拒否されたユーザをリダイレクトする場所を指定することを期待します

    （返される URL はカスタム Java クラスに渡される情報によって異なる場合があります。 たとえば、グループ名はアクティブな式の *param* フィールドに渡される場合があります。 この後、カスタム Java クラスはグループ名をテストして、返す URL を決定できます）。

  - WebAgent-HTTP-Cookie-Variable。 このレスポンス属性では、SiteMinder はユーザの共通名などの結果文字列が Cookie 変数に割り当てられることを期待します。 結果文字列は、フォームのカスタマイズのためのユーザの共通名の表示など、望む方法で使用できます。

    SiteMinder レスポンス属性エディタで Cookie 名を指定します。

  メソッドが失敗する場合（メソッドが -1 または 0 を返す場合）、レスポンス属性は無視されます。

ActiveExpression メソッド

Java 許可 API 内の基本インターフェースは ActiveExpression です。 カスタム許可機能を提供する Java クラスはすべてこのインターフェースを実装する必要があります。

基本インターフェースから実装するクラスの名前は、任意の関連するアクティブな式の［*param*］フィールドに表示される必要があります。

SiteMinder は、基本インターフェース ActiveExpression 内の以下のメソッドを呼び出します。

メソッド	説明
init()	カスタム Java クラスが必要とするあらゆる初期化手順を実行します。SiteMinder は、このメソッドをカスタム ActiveExpression クラスの各インスタンスにつき 1 回ずつ呼び出します。
invoke()	ActiveExpression オブジェクト内のカスタム許可機能を実行し、結果を返します。
release()	ActiveExpression オブジェクトが必要とする任意の要約手順を実行します。SiteMinder がシャットダウンしている場合、SiteMinder は ActiveExpression クラスの各インスタンスに対してこのメソッドを 1 回ずつ呼び出します。

注: ActiveExpression を実装するクラスは、ActiveExpression クラスのメンバ変数に保存されたインスタンスのステートに依存しないステートレス モデル上で実装される必要があります。

許可 API 内の他のクラス

許可 API 内の以下のクラスが ActiveExpression 基本インターフェースと共に使用されます。

クラス	説明
ActiveExpressionContext	invoke() に渡される以下のコンテキスト クラスが含まれます。 ■ APIContext ■ RequestContext ■ UserContext

クラス	説明
RequestContext	たとえば、要求のサーバまたはリソース部分など、ユーザのアクセスの要求に関する情報を提供します。

SAML アサーションまたはレスポンスの変更

SAML 仕様に基づき、アサーション（SAML 1.x）またはレスポンス（SAML 2.0）はプロデューサ サイトで生成され、検証のためにコンシューマ サイトへ送られます。通常、SiteMinder がプロデューサ サイトで生成するデフォルトの SAML アサーションまたはレスポンスを使用します。アサーションまたはレスポンスのコンテンツを変更する場合、Java アサーション ジェネレータ プラグインを実装することで実行できます。 このプラグインは、コンシューマ（SAML 1.x）およびサービス プロバイダ（SAML 2.0）の両方に適しています。

SAML アサーションまたはレスポンスを変更する方法

1. Java SAML アサーション ジェネレータ プラグインを実装します。

   実装は、SiteMinder Assertion Generator Framework 用のプラグインです。Assertion Generator Framework はカスタム プラグイン オブジェクトにデフォルト トークンを送信します。 処理の後、カスタム オブジェクトは変更されたトークンを Assertion Generator Framework に渡します。

2. プラグイン クラスの完全修飾名およびプラグインが必要とする可能性のある任意のオプションのパラメータを指定することでプラグインを設定します。

   以下の方法のいずれかでカスタムのアサーション ジェネレータ プラグインを設定します。

   - SAML 1.x サポートでは、［SiteMinder アフィリエイト プロパティ］ダイアログ ボックスの［Advanced］タブ上。
   - SAML 2.0 サポートでは、［SiteMinder サービス プロバイダ プロパティ］ダイアログ ボックスの［Advanced］タブ上。
   - C または Perl ポリシー管理 API を使用。

   注: アサーション ジェネレータ プラグインの設定は、バージョン v6.0 SP 2 以降のポリシー管理 API セッションを必要とします。

SiteMinder とアサーション ジェネレータ間の相互作用

以下の手順では、SiteMinder とカスタム アサーション ジェネレータ プラグインの相互作用の概要について説明します。 アクティビティは、許可されたユーザが SiteMinder ポリシー サーバを通して、アサーションを消費するサイトでのリソースに対する要求を作成する場合に開始します。

1. 認定ユーザは、コンシューマまたはサービス プロバイダに対して SAML アサーションまたはレスポンスを必要とします。
2. SiteMinder Assertion Generator Framework はデフォルトの SAML アサーションまたはレスポンスを生成します。
3. アサーション ジェネレータ プラグインがアサーションを消費するサイトに対して定義される場合、SiteMinder Assertion Generator Framework はプラグイン キャッシュからプラグイン オブジェクトのインスタンスを要求します。

   注: アサーションを消費するサイトに定義できる SiteMinder アサーション ジェネレータ プラグインは 1 つのみです。
4. プラグインがキャッシュにまだロードされていない場合、以下が実行されます。
   a. SiteMinder はプラグイン クラスをインスタンス化し、キャッシュにそれをロードします。
   b. SiteMinder はプラグインの init() メソッドを呼び出します。 このメソッドは、プラグインに対して実装したあらゆる初期化手順を実行します。

   正常に初期化されたプラグイン オブジェクトは、SiteMinder がシャットダウンするまでキャッシュに残ります。 これにより、プラグインが要求されるたびにオブジェクトを再ロードおよび再初期化する必要がなくなります。
5. SiteMinder Assertion Generator Framework は、手順 2 で生成されたデフォルトの XML トークンをプラグインの customizeAssertion() メソッドに渡します。
6. プラグインは、情報を必要とされたとおりに検証または修正し、処理されたアサーションを Assertion Generator Framework に返します。

7. Assertion Generator Framework は処理されたトークンをコンシューマまたはサービス プロバイダへ渡します。このサイトは、ユーザの要求に応答する方法を決定するのにアサーション内の情報を使用します。
8. ユーザがサービス プロバイダに対するアサーションを要求する場合は常に手順が繰り返されます。

SiteMinder がシャットダウンする直前に、SiteMinder はプラグインの release() メソッドを呼び出して必要な要約アクティビティを実行します。

開発と展開の注意点

カスタムの アサーション ジェネレータ プラグインを開発および展開する場合、以下の点を念頭に置いてください。

- 実装されたプラグイン クラスは、パラメータのないパブリック デフォルト コンストラクタ メソッドを提供する必要があります。
- 実装はステートレスである必要があります。つまり、単一のプラグイン インスタンスを同時に複数のスレッドに使用できる必要があります。
- 構文の要件および customizeAssertion() メソッドに渡されるパラメータ文字列の使用はカスタム オブジェクトの責任です。
- カスタム オブジェクトは、customizeAssertion() に渡されるデフォルトのアサーションを、たとえば Document Object Module (DOM) パーサまたは Simple API for XML (SAX) パーサを使用して解析する必要があります。サンプル プラグイン クラス AssertionSample.java は DOM パーサを使用します。
- 展開したプラグイン クラスを見つけるために SiteMinder を有効化するには、設定ファイル JVMOptions.txt の -Djava.class.path に展開位置を指定します。 このファイルは、SiteMinder のインストールされたディレクトリ構造 <install-path>¥siteminder¥config に置かれています。
- アサーション プラグイン クラスのサンプルについては、AssertionSample.java を参照してください。ここでは、SiteMinder がインストールされたディレクトリ構造の場所 <install-path>¥sdk¥samples¥assertiongeneratorplugin 内で SAML 1.x アサーションに対するプラグイン プロセスが示されています。同じディレクトリ内に、Assertion Generator Plugin for SAML 2.0、SAML2AppAttrPlugin.java、および WS-Federation、WSFedAppAttrPlugin.java のサンプルが置かれています。

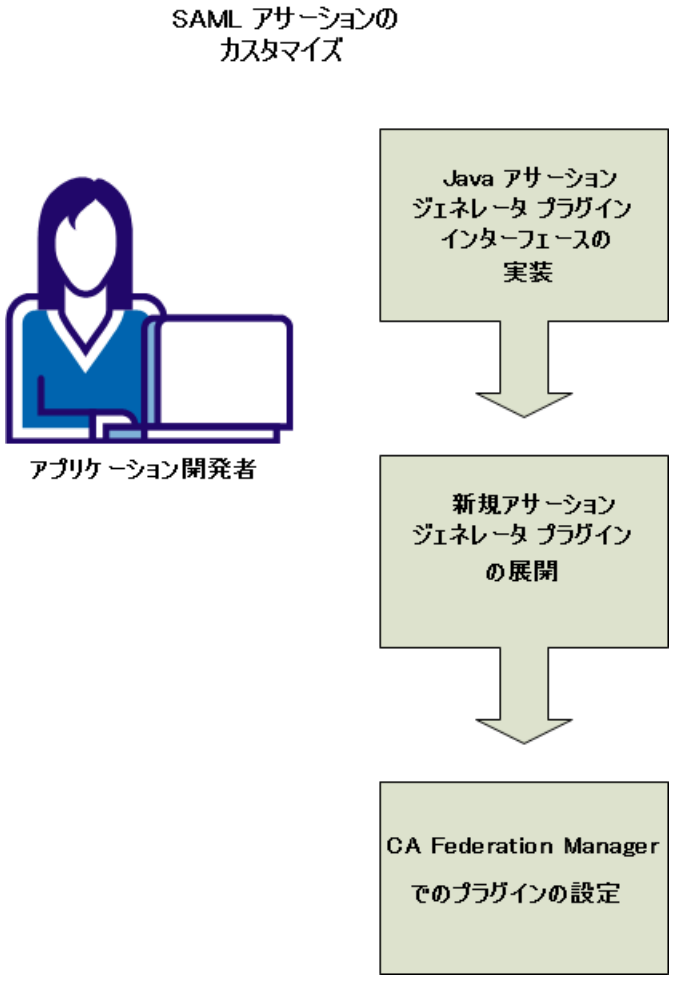
第 6 章: SAML アサーションのカスタマイズ

セキュリティ ドメインは、アサーションという名前のデータ パッケージを使用して認証と認可を交換します。 SAML（Security Assertion Markup Language）は、アサーションの形式を指定するオープン スタンダードです。連係されたパートナーシップは、ID プロバイダ（アサーションのプロデューサ）およびサービス プロバイダ（アサーションのコンシューマ）から構成されます。

企業は、連係されたパートナー間のビジネス契約に基づいてアサーションのコンテンツを変更できます。たとえば、あるパートナーは、アサーションで属性に対する使いやすい名前相当物を要求することができます。 または、あるパートナーは、アサーションに各属性の XML タイプ指定を含めるよう選ぶことができます。

CA SiteMinder® Federation は AssertionGeneratorPlugin.java インターフェースの実装で SAML アサーションを作成します。アプリケーション開発者は、既存の実装クラスを上書きすることで SAML アサーションのコンテンツを増強できます。

以下の図は、カスタム アサーション ジェネレータ プラグインを作成する過程を示します。



SAML アサーションをカスタマイズする過程にはこれらの手順が含まれます。

1. Java アサーション ジェネレータ プラグイン インターフェースを実装します (P. 145)。
2. 新しいアサーション ジェネレータ プラグインを展開します (P. 147)。
3. CA SiteMinder® Federation UI 内でアサーション ジェネレータ プラグインを設定します (P. 148)。

Java アサーション ジェネレータ プラグイン インターフェースの実装

AssertionGeneratorPlugin.java インターフェースを実装することでカスタム アサーション ジェネレータ プラグインを作成します。 実装クラスに関する最小要件を以下に記載します。

次の手順に従ってください:

1. パラメータが含まれない公のデフォルト コンストラクタ メソッドを提供します。
2. 実装がステートレスであることを確実にするのに役立つコードを提供します。その結果、多くのスレッドが単一のプラグイン クラスを使用できます。
3. customizeAssertion メソッドへのコールを含めます。

   例

   この例では、アプリケーション開発者が使いやすい名前属性を作成するために handler.updateNameID を定義していると仮定します。

```
/**
     * <p>Performs Assertion Generator callout functionality to customize the
     * SAML assertion in the <code>AssertionGeneratorPlugin</code> object and
     * returns a result.</p>
     * @param apiContext     A context object that provides methods for sending
     *                     log, trace, and error messages to the Policy Server.
APIContext.getAttrMap() メソッドを使用して、アプリケーション URL で指定されたアプリケーションによってポストされる属性を取得します。
     * @param userContext    A context object that allows a custom object to set
     *                       and retrieve information about a user in a user
     *                       directory. The information includes user
     *                       attributes and directory attributes associated
     *                       with the user.
     * @param pluginParam    The string for Assertion plug-in parameters.
     * @param inputAssertion    The current XML token representing the SAML
Assertion.
     * @param outputAssertion   The final XML token representing the SAML
Assertion.
     * @return  0 if assertion is customized successfully, or -1 if no
customization or an error occurred.
     * If the method fails, the outputAssertion is ignored.
     * @throws java.lang.Exception For cases when the customization terminates
unexpectedly.
     *
     **/
```

```
    public int customizeAssertion(APIContext apiContext, UserContext
userContext,String pluginParam,
     String inputAssertion, final StringBuffer outputAssertion) throws Exception
{
        if (inputAssertion == null || inputAssertion.equals("")) {
            // Indicates non-zero for an error.
            apiContext.trace(PLUGIN_TAG, "Received null or empty response for
customization");
            return -1;
        }
        apiContext.trace(PLUGIN_TAG, "Entering customizeAssertion");
        StringBuffer newAssertion = new StringBuffer(inputAssertion);

        try
        {
            Saml1AssertionHandler handler =
                initHandler(apiContext, userContext);
            handler.updateNameID(newAssertion);
            handler.addAttributes(pluginParam, newAssertion);
        }
        catch(Throwable th)
        {
            apiContext.error("SAML1AssertionSample: " + th.getMessage());
            StringWriter writer = new StringWriter();
            th.printStackTrace(new PrintWriter(writer));
            writer.flush();
            apiContext.trace(PLUGIN_TAG,
                             "Error customizing Assertion:¥n" +
writer.toString());

            apiContext.trace(PLUGIN_TAG, "Done customizeAssertion");
            return -1;
        }

        outputAssertion.append(newAssertion);

        apiContext.trace(PLUGIN_TAG, "Done customizeAssertion");

        // return "success"
        return 0;
    }
```

注: 構文要件および customizeAssertion メソッドへ渡されるパラメータ文字列の使用は、カスタム オブジェクトの責任です。

アサーション ジェネレータ プラグインの展開

AssertionGeneratoPlugin.java インターフェースの実装クラスをコード化した後、それをコンパイルし、CA SiteMinder® Federation が実行可能ファイルを検出できることを確認します。

次の手順に従ってください:

1. 以下のいずれかの方法でアサーション ジェネレータ プラグイン コードをコンパイルします。
   - サンプル プラグインを使用している場合は、ビルド スクリプトを使用してプラグインをコンパイルします。 ビルド スクリプトは、ディレクトリ *federation_mgr_sdk_home*¥sample にインストールされます。 ビルド スクリプトは次のとおりです。

     Windows: build_plugin.bat

     UNIX: build_plugin.sh

     コンパイルされたサンプル プラグイン、fedpluginsample.jar は、ディレクトリ *federation_mgr_sdk_home*¥jar にあります。
   - 自分のプラグインを書き込む場合は、プラグインをコンパイルするときに smapi.jar をインクルードします。
2. JVMOptions.txt ファイルで、プラグインのクラスパスをインクルードするように、-Djava.class.path 値を変更します。 ディレクトリ *federation_mgr_home*¥siteminder¥config に JVMOptions.txt ファイルを置きます。

   任意のディレクトリにプラグイン jar を配置し、JVMOptions.txt ファイルがそれを参照するよう設定できます。サンプル プラグインを使用するには、fedpluginsample.jar を参照するようクラスパスを変更しますが、smapi.jar 用のクラスパスは変更しないでください。

   注: プラグインで Apache Xerces または Xalan を使用するには、CA SiteMinder® Federation でインストールされた Xerces または Xalan のバイナリ ファイルを使用します。 バイナリは CA SiteMinder® Federation SDK でインストールされません。 互換性の理由でこれらのファイルを使用する必要があります。
3. CA SiteMinder® Federation サービスを再起動します。

   このサービスの再起動は、CA SiteMinder® Federation がアサーション ジェネレータ プラグインの最新バージョンを使用するのに役立ちます。

UI 内のアサーション ジェネレータ プラグインの設定

アサーション ジェネレータ プラグインを設定するには、CA SiteMinder® Federation UI で設定に対して値を指定します。

注: プラグインを展開するまで、プラグイン設定を行わないでください。

次の手順に従ってください:

1. CA SiteMinder® Federation UI にログオンします。
2. 変更するパートナーシップのパートナーシップ ウィザードのアサーション設定手順に移動します。
3. 次のフィールドに値を入力します。

   プラグイン クラス

   プラグインの Java クラス名を指定します。 名前を入力します。 このプラグインはランタイムで呼び出されます。

   例： com.mycompany.assertiongenerator.AssertionSample

   このプラグイン クラスはアサーションを解析および変更してから、最終処理のために CA SiteMinder® Federation に結果を返すことができます。各依存パーティのアサーション ジェネレータ プラグインを指定します。 コンパイルされたサンプル プラグインは、ディレクトリ *federation_mgr_sdk_home*/jar に含まれています。

   プラグイン パラメータ

   （オプション）。 CA SiteMinder® Federation が実行時にパラメータとしてプラグインへ渡す文字列を指定します。 文字列にはあらゆる値を含めることができ、従う特定の構文はありません。

   プラグインは、受信するパラメータを解釈します。 たとえば、パラメータは属性の名前です。または、文字列には、何かを実行するようにプラグインに命令する整数を含めることができます。

アサーション ジェネレータ プラグインは、コード化およびコンパイルされ、配置済みです。 CA SiteMinder® Federation アサーション ジェネレータは、フェデレーション パートナーによって定義される、機能強化されたアサーションを作成します。

第 7 章: 分散代行管理サービス API

このセクションには、以下のトピックが含まれています。

DMS API について

ディレクトリ管理は SiteMinder ユーザ ディレクトリ内のオブジェクトを管理することから構成されます。たとえば、ディレクトリ管理アプリケーションのユーザは組織を作成し、組織にグループを追加し、グループにエンド ユーザを追加できます。アプリケーションは、DMS API でディレクトリ管理操作を実行します。

分散代行管理サービス（DMS） API を使用すると、LDAP および ODBC ディレクトリ上でディレクトリ管理操作を実行できます。

LDAP ディレクトリでは、特定の権限を持つユーザが以下のタスク（これだけに制限されない）を実行するのを許可するクライアント アプリケーションを記述するのに DMS API を使用できます。

- 組織の作成
- グループの作成
- 組織へのグループの追加

- グループへのユーザの追加
- ユーザのプロファイルの変更

ODBC ディレクトリでは、多くの DMS API 操作を実行できますが、一部の操作は実行できません。

注: DMS API（Java のみで利用可能）の機能は、DMS Workflow API（C/C++ のみで利用可能）とは異なっています。 DMS API を使用すると、SiteMinder DMS 製品と同様の操作を実行するディレクトリ管理アプリケーションを開発できます。 DMS Workflow API は、DMS と共に動作し、特定のプリプロセスおよびポストプロセス DMS イベントが発生した場合に起動します。これにより、これらのイベントの前または後に追加の機能を実行するアプリケーションを開発できます。

必要な JAR ファイル

JAR ファイル smjavasdk2.jar は委任管理アプリケーションの構築および実行に必要です。 JAR ファイルは以下の場所に保存されます。

- Windows プラットフォーム：

  <install_path>¥sdk¥java

- UNIX プラットフォーム：

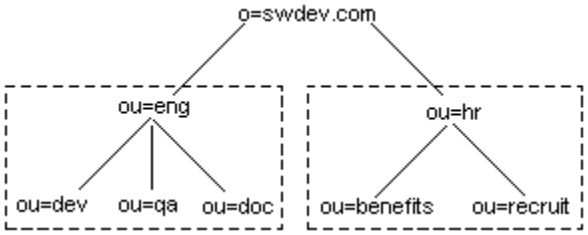
  <install_path>/sdk/java

SiteMinder ユーザディレクトリ

SiteMinder ユーザ ディレクトリは、大規模のエンティティ（企業など）内の単一の組織単位（エンジニアリングまたは人事など）の概念的なビューです。 SiteMinder ユーザ ディレクトリは、ディレクトリをより小さく、管理しやすい、論理的に関連するセグメントに分けることで、ディレクトリ構造全体の管理を簡素化します。

カスタム DMS アプリケーション内のメソッドは、一意の組織 DN を指定することで、特定の SiteMinder ユーザ ディレクトリを参照します。 組織 DN は、SiteMinder ユーザ ディレクトリの反転されたツリー構造のルートまたはトップレベル、またはサブレベルのいずれかをポイントします。

DMS 要求はすべて組織 DN を参照します。以下の図では、2 つの SiteMinder ユーザ ディレクトリが破線のボックスで囲まれています。 ディレクトリは、組織 DN の ou=eng、o=swdev.com（エンジニアリング組織単位を表す）、および ou=hr、o=swdev.com （人事組織単位を表す） で識別されます。



SiteMinder ユーザ ディレクトリは他の SiteMinder ユーザ ディレクトリ内に存在できます。 前の図で、エンジニアリング組織単位内には 3 つの SiteMinder ユーザ ディレクトリがあります。これらには属性および組織名 ou=dev、ou=qa および ou=doc があります。 人事組織単位内には、ou=benefits および ou=recruit の 2 つの SiteMinder ユーザ ディレクトリがあります。

SiteMinder ユーザ ディレクトリ コンテナ

通常、SiteMinder ユーザ ディレクトリ内の組織 DN には 1 つ以上のサブ DN があります。 サブ DN には情報のリストが含まれるため、「コンテナ」とも呼ばれます。 これらのコンテナのデフォルトの名前、およびそれらに含まれる情報は次のとおりです。

- people --Siteminder ユーザ ディレクトリ内のエンド ユーザ。
- roles -- SiteMinder ユーザ ディレクトリで使用されるロール。

- groups -- SiteMinder ユーザ ディレクトリで使用されるグループ。
- orgadmin-- 組織単位を管理できる管理者に対するグループ。

サブ DN はクラス SmDmsConfig によって管理されます。SmDmsConfig オブジェクトを作成する場合、デフォルトの サブ DN 名を維持するか、または新しい名前を割り当てることができます。

組織管理者は orgadmin コンテナでリスト表示されます。 階層組織では、特定の orgadmin コンテナにリストされる組織管理者は、そのコンテナと関連付けられる組織単位およびその下のあらゆる組織単位を管理することができます。

属性ベースの委任

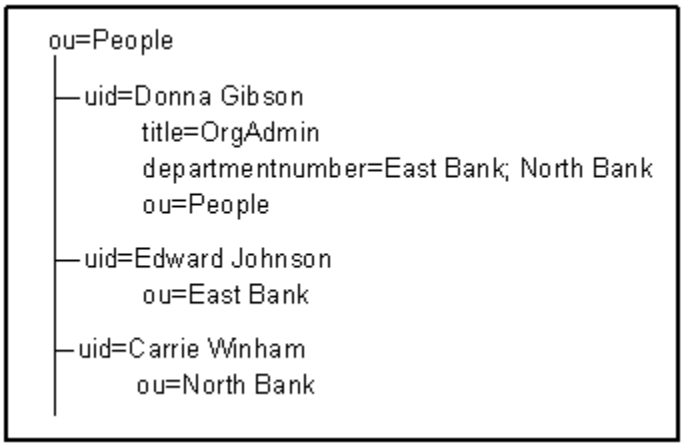
階層組織に加えて、DMS はまた、フラットなディレクトリ構造を実装したサイトに対して管理モデルを提供します。 このモデルでは、委任は階層レベルの代わりにユーザ プロファイル内の属性に基づいています。

フラットなディレクトリで、DMS は属性/値のペアをユーザ プロファイルに追加し、ユーザをグループ化します。 一度ユーザがグループ化されたら、別の属性/値のペアがどのユーザがグループを管理できるかを決定します。

DMS はユーザ プロファイルに属性/値ペアを追加することで、ユーザを組織にグループ化します。 たとえば、組織 East Bank に属するユーザがプロファイルに *ou*=East Bank という属性/値ペアを持つとします。ここで *ou* はユーザが所属する組織を示す属性です。

組織管理者は、組織管理者のプロファイルにリストされる組織のみを管理できます。組織のリストは、SmDmsConfig コンストラクタで指定するプロファイル属性に割り当てられます。 たとえば、組織管理者が管理できる組織を含む属性として departmentnumber を指定した場合、属性/値のペア *departmentnumber*=East Bank は、組織の管理者が East Bank 組織のみを管理できることを意味しています。

以下の図は、属性ベースの委任が実装される方法について説明しています。



この例では、Donna Gibson は East Bank および North Bank の組織管理者です。Edward Johnson および Carrie Winham は Donna のユーザ プロファイル内の departmentnumber 属性にリストされる組織に属するため、Donna による管理が可能です。

属性ベースの委任の設定

SmDmsConfig コンストラクタ内の属性ベースの委任を有効にする属性を指定します。 以下の情報を識別するのに 3 つの属性が必要です。

- 組織管理者としてのユーザ。 この属性は、LDAP ディレクトリ内の、他の情報の保存に使用していない任意の属性、たとえば o とすることが可能です。

  以下の例のように、コンストラクタの *OrgAdminSubDn* パラメータを通してユーザを組織管理者として指定できます。

  OrgAdminSubDn="(title=OrgAdmin)";

- 組織管理者が管理できる組織。 デフォルトで、DMS は管理された組織の保存に departmentnumber 属性を使用します。

  コンストラクタの *OrgAdminOrgs* パラメータによってこの属性を指定します。 たとえば、以下のようにします。

  OrgAdminOrgs="departmentnumber";

- ユーザが属する組織。 デフォルトで、DMS は ou 属性を使用します。

  コンストラクタの *DnOrgs* パラメータによってこの属性を指定します。 たとえば、以下のようにします。

  DnOrgs="ou";

DMS ユーザ

DMS ユーザには、ディレクトリ管理権限の以下のカテゴリのいずれかが割り当てられます。 カテゴリは、低いものから高いもの順番で以下にリストされています。

- エンド ユーザ -- ユーザのパスワードの変更およびユーザがメンバーであるロールの表示（追加はできません）などのエンド ユーザ自身のアカウントに関する特定の情報を管理できます。
- 組織管理者 -- 全組織およびその下の任意の組織を管理できます。

- SiteMinder 管理者 --1 つ以上のドメイン内のディレクトリを管理できます。

  さまざまな SiteMinder 管理者権限が存在します。DMS で、SiteMinder 管理者はシステム レベルのユーザ管理権限が必要で、少なくとも 1 つのドメインの中に存在する必要があります。
- スーパー管理者 -- すべてのドメインのディレクトリをすべて管理できます。

DMS ユーザの異なるカテゴリにログインするには異なる login() メソッドを使用します。

実装クラス

インターフェース SmDmsApi はクラス SmDmsApiImpl によって実装されます。 DMS API の出発点としてこのクラスを使用します。

このクラスを使用して、SmDmsDirectory オブジェクト内の情報にどのようにアクセスしたいかを決定できます。以下の 2 つの種類の情報のいずれかを提供することで決定できます。

- ターゲット ユーザ ディレクトリの名前または OID。この情報を提供するには、getDirectoryContext() を呼び出します。
- ユーザがアクセスを試行する保護レルムの OID。 この情報を提供するには、getDmsContext() を呼び出します。

これらのメソッドは、渡されるコンテキスト オブジェクトに入力します。

Context クラス

SmDmsApiImpl クラス内の getDirectoryContext() および getDmsContext() メソッドが、SmDmsDirectoryContext または SmDmsContext のいずれかのコンテキスト オブジェクトを作成します。 コンテキスト オブジェクトには、ユーザ ディレクトリ、セッション、および接続情報などの情報が含まれます。 コンテキスト オブジェクトは、提供されたレルム OID またはユーザ ディレクトリ名または OID 内のコンテキストから派生するためにこのように呼ばれています。 コンテキスト オブジェクトがある場合、getDmsDirectory() メソッドを呼び出して SmDmsDirectory オブジェクトを取得します。 このオブジェクトは LDAP またはその他のネームスペースを表し、ネームスペース内の組織またはその他の要素へのアクセスを提供します。

Object クラス

Object クラス、SmDmsObject およびそのサブクラスは、ディレクトリ オブジェクトを作成し管理する方法を提供します。SmDmsObject には以下のサブクラスがあります。

- SmDmsDirectory は、LDAP などユーザのネームスペースを表します。これはディレクトリ全体の情報へのアクセスを提供します。
- SmDmsOrganization は、ディレクトリ内のエンジニアリングまたは人事などの組織を表します。 SiteMinder ユーザ ディレクトリは組織の概念的なビューです。これは組織管理者によって管理され、組織 DN によって一意に識別されます。
- SmDmsGroup は組織内のグループを表します。 グループは、たとえば企業の在籍期間が 1 年未満の従業員のグループなど、共通事項を持つ一連のオブジェクトです。 グループ オブジェクトでは、権限をユーザ個別にではなく集合的に割り当てることができます。
- SmDmsRole は、組織内のロールを表します。 ロールは、組織内のユーザの機能を説明します。 これは、ユーザが同じ権限を持つ他のユーザと共に管理されることを可能にします。 たとえば、オンラインで商品を注文でき、インベントリ リストを参照できるユーザは buyer のロールを持つ可能性があります。
- SmDMSUser は、組織内のユーザを表します。 ユーザはエンド ユーザまたは管理者が可能です。

オブジェクト モデル

ディレクトリ、組織、グループ、ロール、またはユーザ オブジェクトで操作を実行する場合、一般的な SmDmsObject またはそのサブクラスのいずれかを使用することを選択できます。 ただし、オブジェクト固有の操作（たとえばユーザの認証、ユーザのパスワードの変更、またはユーザの権限の取得）に対しては、オブジェクト固有のサブクラスを使用する必要があります。

サブクラスに対応するオブジェクトは、ユーザ オブジェクトに対する DMSOBJECT_CLASS_USER などの *クラス識別子*で識別されます。これらの識別子は SmDmsObject で定義されています。 サブクラスを使用してオブジェクトを作成する場合、たとえば SmDmsUser でユーザを作成する場合で addObject() を呼び出すと、クラス識別子が自動的に設定されます。 ただし、SmDmsObject で一般的なディレクトリ、組織、グループ、ロール、またはユーザ オブジェクトを作成する場合、addObject() を呼び出す前にクラス識別子を設定する必要があります。

Search クラス

Search クラス SmDmsSearch は、検索操作に対する設定オブジェクトを表します。 これは検索ベースおよびフィルタを保持します。 フィルタは、オブジェクト クラスに対する文字列ベースの検索式を予期します。

Search クラスは、対応するクラス識別子とペアにされた識別名のリストを返します。オプションで、検索で取得された項目に対する選択された属性情報も返します。

Cursor クラス

SmDmsCursor クラスを使用して、結果セット操作に対する並べ替えおよびページング動作を定義できます。たとえば、以下のようにします。

- SmDmsCursor コンストラクタの *sort* パラメータを設定して、行の並べ替えに使用する列を指定します。
- setBlockSize() を呼び出して、一度に結果セットから呼び出すことのできる行の最大数、つまりページ内の最大行数を定義します。

- setOffset() を呼び出して、結果セットから返されたブロックの開始オフセット（行番号）を指定します。
- isSortingCritical() を呼び出して、結果セットを並べ替える必要があるかどうかを指定します。
  - true を指定すると、並べ替えが可能な場合にのみ結果セットが取得されます。
  - false を指定すると、並べ替えられなかった結果セットが、並べ替えの実行が不可能な場合に送信されます。 同じ SmDmsCursor オブジェクト上の isSorted() 呼び出しは false を返します。
- isPagingCritical() を呼び出して、結果セットにページングを実行する必要があるかどうかを指定します。
  - true を指定すると、ページングが可能な場合にのみ結果セットが取得されます。
  - false を指定すると、並べ替えが実行できない結果セット全体が送信されます。 同じ SmDmsCursor オブジェクト上の isPaginated() 呼び出しは false を返します。

カーソル操作をサポートする検索

定義された SmDmsCursor オブジェクトを以下のメソッドのいずれかに渡すことで、並び替えおよびページング操作を実行できます。

- SmDmsOrganization 内の検索メソッドのいずれか
  - search()
  - searchForward()
  - searchBack()
  - searchRefresh()
- SmDmsObject.getGroups()
- SmDmsOrganization.getGroups()
- SmDmsGroups.getMembers()

注: getGroups() および getMembers() は、ODBC ディレクトリの検索ではサポートされません。

Microsoft LDAP ディレクトリの検索

並べ替えおよびページング操作は AD ネームスペースを通した Active Directory に対してサポートされていません。 並べ替えおよびページング操作は LDAP ネームスペースを通した Active Directory に対してサポートされています。

AD ネームスペースを通して Active Directory と通信する場合、SiteMinder は並べ替えおよびページング要求に以下のように応答します。

- isPagingCritical() および isSortingCritical() の両方が SmDmsCursor オブジェクト内で false を返す場合、結果セットが返されます。並べ替えおよびページング操作は実行されません。
- isPagingCritical() または isSortingCritical() のいずれかが SmDmsCursor オブジェクト内で true を返す場合、エラーが発生します。 結果セットは返されません。

並べ替えおよびページング操作が SmDmsCursor コンストラクタに重要かどうかを指定します。

ディレクトリ管理アプリケーションの記述

ディレクトリ管理アプリケーションを記述する方法

1. ポリシー サーバへの接続の確立
2. セッション オブジェクトの取得

   ユーザまたは管理者が正常にログインするとき、セッション オブジェクトが取得されます。

   - SiteMinder 管理者にログインし SiteMinder 管理者セッションを確立するには、ユーティリティ パッケージの SmApiSession クラスで login() メソッドを呼び出します。

     ログインが成功する場合、セッション オブジェクトにはセッション指定が含まれます。

   - エンドユーザ、DMS 組織管理者、または DMS スーパー管理者としてログインするには、エージェント API パッケージのエージェント API クラス内の login() メソッドを呼び出します。

     ログインが成功する場合、セッション指定は SessionDef オブジェクトの［*spec*］フィールドに入力されます。SmApiSession オブジェクトで［*spec*］値を設定します。

3. セッション オブジェクトを渡す

   有効なセッションを取得した後に、セッションを SmDmsApiImpl クラスのコンストラクタへ渡すことで DMS API オブジェクトを作成します。たとえば、以下のようにします。

   ```
   SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);
   ```

   この例で、dmsApi は新規 DMS API オブジェクトで、*apiSession* は管理者がログインに成功した場合に取得されるセッションです。

   注: DMS API オブジェクトを作成する場合はいつでも、セッションおよび接続情報をオブジェクトに渡します。

4. ディレクトリ管理コンテキストの作成

   ユーザ ディレクトリにアクセスするために DMS API を使用するには、以下のいずれかの情報が必要です。

   - 自己登録方式が設定されたレルムの OID。

     この情報を渡すには、SmDmsApiImpl.getDmsContext() を呼び出します。

   - 作業している SiteMinder ユーザ ディレクトリ。

     この情報を渡すには、SmDmsApiImpl.getDirectoryContext() を呼び出します。

   知っているまたは提供を選択する情報のタイプが、ユーザ ディレクトリへのアクセスに対するディレクトリ管理コンテキストを決定します。以下のようになります。

知っている情報	以下を実行	使用項目
自己登録方式が含まれるレルムの OID	ユーザが SiteMinder 管理者	分散代行管理サービス（DMS）コンテキスト
SiteMinder ユーザ ディレクトリ名または OID	ー	ディレクトリ コンテキスト

   DMS コンテキストおよびディレクトリ コンテキストは、ディレクトリ情報のアクセスおよび操作が可能な SmDmsDirectory オブジェクトを取得するという同じ結果を達成するために 2 つの異なる手段を提供します。

5. オブジェクトの作成および操作

   コンテキストの作成後、DMS オブジェクト モデルを使用してディレクトリ オブジェクトを作成および操作できます。 ディレクトリ オブジェクトで作業する場合、以下の情報が必要です。

   - オブジェクトの識別名。
   - 以下のようなオブジェクトのタイプ。
     - トップレベルの組織
     - 組織単位
     - グループ
     - ユーザ
     - ロール

DMS コンテキスト

DMS コンテキストを使用すると、提供するレルム OID のコンテキスト内で SmDmsDirectory オブジェクトにアクセスできます。 DMS コンテキスト クラスは SmDmsContext です。

以下のように DMS コンテキスト オブジェクトを作成できます。

```
SmDmsContext dmsContext = new SmDmsContext();
```

クラス SmDmsApiImpl でメソッド getDmsContext() を使用して DMS コンテキスト オブジェクトを取得できます。

注: getDmsContext() の呼び出しには SiteMinder 管理者権限が必要です。

DMS コンテキスト オブジェクト情報を取得する前に、getDmsContext() 呼び出しに渡すレルム オブジェクトを作成する必要があります。 レルム オブジェクトには以下が必要です。

- AgentAPI.isProtected() へのエージェント呼び出しから取得された、有効なオブジェクト識別子（OID）がある。
- 登録方式と設定されている。

以下のように SmRealm オブジェクトを作成します。

```
SmRealm realm = new SmRealm();
```

次に、setOid() を呼び出すことによりレルム OID を設定します。このメソッドは、ポリシー管理 API の SmObjectImpl クラスを拡張するオブジェクトを通して呼び出すことができます。

レルム オブジェクトに対して OID を設定した後で getDmsContext() を呼び出し、レルム オブジェクトを渡します。

例：

エージェントは isProtected() を呼び出して、ユーザがアクセスを試みるリソースが保護されているかどうかを判断します。 ポリシー サーバは、リソースにアクセスするために必要な認証情報を返すことで、リソースが保護されていることを示します。 返される情報には、保護されたレルムの OID が含まれています。 以下の例に示されるように、返されたレルム OID（例では m_REALM_OID）を使用して、作成して getDmsContext() に渡すレルム オブジェクトに対する OID を設定します。

```
// Create a DMS API object from a valid session.
SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);

// The realm below should contain a registration scheme.
// You can get a directory OID from the registration scheme.
SmRealm realm = new SmRealm ();
realm.setOid (m_REALM_OID);
// Create the DMS context object.
SmDmsContext dmsContext = new SmDmsContext ();

// This call returns the realm, self registration,
// and user directory information through dmsContext.
result = dmsApi.getDmsContext (realm,
                               new SmDmsConfig(),
                               dmsContext);
```

dmsContext オブジェクトから完全なディレクトリ情報を取得するには、dmsContext.getDmsDirectory() を呼び出します。

ディレクトリ OID のみを取得するには、dmsContext.getSelfReg() を呼び出し、次に SmSelfReg.getUserDir() を呼び出します。

ディレクトリ コンテキスト

ディレクトリ コンテキストを使用すると、ユーザ ディレクトリ名または提供する OID のコンテキスト内で SmDmsDirectory オブジェクトにアクセスできます。ディレクトリ コンテキスト クラスは SmDmsDirectoryContext です。ディレクトリ コンテキストを取得するには、クラス SmDmsApiImpl でメソッド getDirectoryContext() を使用します。

以下の例では、SmDmsDirectoryContext オブジェクトが *dirContext* で返されます。 ディレクトリ オブジェクトに関する情報を取得するには getDmsDirectory() を呼び出します。

// Create a DMS API object from a valid session.

```
SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);

// Create the directory context object.
SmDmsDirectoryContext dirContext=new SmDmsDirectoryContext();

// Directory object to pass in to getDirectoryContext().
SmUserDirectory userDir = new SmUserDirectory ();

// setOid() method can take the name of the user directory.
userDir.setOid ("smdev");

// This call returns directory information through dirContext.
result=dmsApi.getDirectoryContext(userDir,
                                  new SmDmsConfig(),
                                  dirContext);
```

DMS コンテキスト内のユーザ タイプの変更

ディレクトリ コンテキストで、スーパー管理者、SiteMinder 管理者、組織管理者、またはエンド ユーザなど、任意のユーザ タイプに代わって操作を実行できます。 ただし、DMS コンテキスト オブジェクトを作成するにはメソッド getDmsContext() を呼び出す必要があり、このメソッドを呼び出すには SiteMinder 管理者権限が必要です。

getDmsContext() が呼び出され、DMS コンテキストがセッションに対して確立された後、セッション内の後続の操作に対してユーザ タイプを変更することが可能です。 たとえば、SiteMinder 管理者が DMS コンテキスト内のセッションを開始した後、同じセッション内で後ほどエンド ユーザが自分のプロファイルを変更することが必要になる場合があります。 SiteMinder 管理者ではなくエンド ユーザの代わりにプロファイル要求を作成するには、ユーザ タイプを変更する必要があります。

DMS コンテキスト オブジェクトを作成するには、SmDmsApiImpl.getDmsContext() を呼び出します。 これを実行する場合、接続情報および SiteMinder 管理者のセッション指定が DMS コンテキスト オブジェクトに含められます。

一連の後続オブジェクト（たとえば、SmDmsDirectory/SmDmsOrganization/SmDmsUser）がセッション内で作成されるにあたり、接続およびセッション情報がオブジェクトからオブジェクトに渡されます。 指定されたオブジェクトに対してユーザ タイプを変更するには、そのオブジェクトの SiteMinder 管理者のセッション指定を、代わって後続の呼び出しが実行される新規ユーザのタイプのセッション指定で置き換えます。 どんなオブジェクト レベルでもセッション指定を変更できます。

DMS コンテキストで作成されたオブジェクト用のユーザ タイプの変更

1. 新規ユーザ タイプによる要求のターゲットになるオブジェクトを作成します。

   たとえば、組織 dmsOrg 内の新規ユーザ オブジェクト dmsUser に対して、識別名 USER_DN を持つエンド ユーザの代わりに要求を作成する場合、以下のようにします。

   ```
   SmDmsUser dmsUser = dmsOrg.newUser(USER_DN);
   ```

   例では、dmsOrg オブジェクト内の SiteMinder 管理者セッション仕様が dmsUser オブジェクトに渡されます。

2. 以下のいずれかの方法で、新規ユーザに対するセッション仕様を取得します。

   - 標準的な SiteMinder エージェントで、デフォルトの HTTP ヘッダ HTTP_SM_SERVERSESSIONSPEC を使用します。
   - カスタム エージェントで、新規ユーザにログインするためにエージェント API を使用します。

3. 新規ユーザおよび DMS オブジェクトのためのセッション指定を渡します。 たとえば、*sessionSpec* がセッション指定である場合、以下のようになります。

   ```
   dmsUser.getApiSession().setSessionSpec(sessionSpec);
   ```

詳細情報:

Context クラス (P. 156)

オブジェクトの作成

組織オブジェクト、グループ オブジェクト、ユーザ オブジェクトまたはロール オブジェクトなどのオブジェクトを作成するには、以下を実行します。

1. DMS コンテキストまたはディレクトリ コンテキスト上で getDmsDirectory() を呼び出して、ディレクトリ オブジェクトを取得するためにコンテキストを使用します。 たとえば、DMS コンテキストを使用するには以下のようにします。

   ```
   SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
   ```

2. クラス SmDmsDirectory 内で newOrganization() を呼び出すことにより組織オブジェクトを作成するには、ディレクトリ オブジェクトを使用します。 o=swdev.com など組織の識別名を渡します。 例：

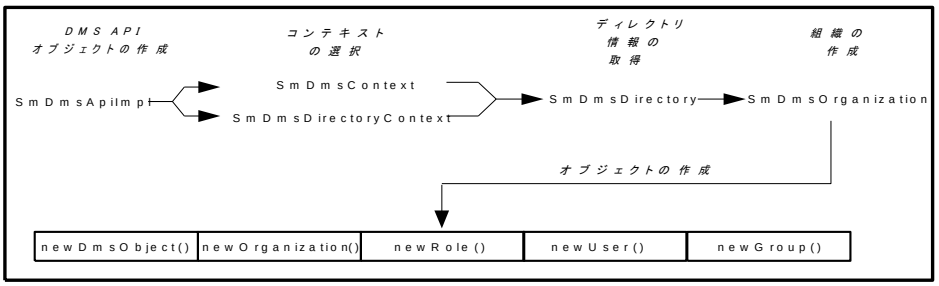
   ```
   SmDmsOrganization org=dmsDir.newOrganization("o=swdev.com");
   ```

3. グループ オブジェクトまたは組織単位オブジェクトなど他のオブジェクトを作成するには組織オブジェクトを使用します。 以下の例は、識別名 ou=UI、ou=eng、o=swdev.com を持つ grp という名前のグループ オブジェクトを作成します。

   ```
   SmDmsGroup grp=org.newGroup("ou=UI,ou=eng,o=swdev.com");
   ```

注: このコードは、ディレクトリにグループを追加しません。

以下の図は、ディレクトリ オブジェクトを作成するための DMS API フローを示しています。



ディレクトリ エントリ属性の取得

特定の属性の値を取得するには、クラス SmDmsObject 内で getAttribute() を呼び出し、属性名を文字列として渡します。 getObject() で属性を取得した後、属性値を利用することができます。 メソッド getAttribute() は、java.lang.Object クラスのメンバを返します。 属性が複数の値を持つ場合、返されたオブジェクトはキャレット（^）で区切られた複数の値を持ちます。

ディレクトリへのオブジェクトの追加

ディレクトリにオブジェクトを追加するには以下を実行します。

1. クラス SmDmsObject 内で setAttribute() を呼び出し、属性名とその値に渡すことで、オブジェクトに対する属性を設定します。 属性名はユーザのディレクトリシステムで定義されています。

   setAttribute() はオブジェクトの定義に必要なだけ何回でも呼び出します。

2. クラス SmDmsObject 内でメソッド addObject() を呼び出します。 例：

   ```
   result = grp.addObject();
   ```

   例で、*result* は SmApiResult オブジェクトです。

   注：（一般的な） SmDmsObject オブジェクトで addObject() を呼び出したい場合、最初に setClassId() を呼び出してクラス識別子を設定する必要があります。

オブジェクトを追加する場合、*objectclass* 属性に対して複数の値を設定することができますが、他の属性に対しては実行できません。modifyObject() メソッドでオブジェクトを変更する場合、どの属性に対しても複数の値を設定できます。

属性に対して複数の値を設定するには、以下を実行できます。

- 値の区切りにキャレット（^）を使用した文字列を渡します。
- 値のベクターを渡し、SiteMinder がその値を文字列に変換するようにします。

たとえば、top および organizationalunit の値を含む文字列を渡すには、以下のコードを使用できます。

```
group.setAttribute("objectclass","top^organizationalunit");
```

同じ値に対するベクターを渡すには、以下のコードを使用できます。

```
Vector objectclass = new Vector();
objectclass.add("top");
objectclass.add("organizationalunit");
group.setAttribute("objectclass", objectclass);
```

注: 既存のオブジェクトに対して、オブジェクト クラスは modifyObjectClass() メソッドによって変更できます。 このメソッドを使用して、オブジェクト クラスに対して複数の値を設定できます。

グループへのユーザの追加

グループへユーザを追加するには、addToGroup() メソッドをクラス SmDmsObject 内で呼び出します。 以下の例では、ユーザ *user1* はグループ *devGroup* に追加されます。

```
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmDmsGroup devGroup = org.newGroup(GROUP_DN);
SmDmsUser user1 = org.newUser(USER_DN1);
result = devGroup.addToGroup(user1);
```

ロールへのユーザの追加

ユーザをロールに追加するには、addToRole() メソッド（クラス SmDmsUser）を呼び出します。 以下の例で、ユーザ *user1* はロール *role* に追加されます:

```
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmDmsRole role = org.newRole(ROLE_DN);
SmDmsUser user1 = org.newUser(USER_DN1);
result = user1.addToRole(role);
```

オブジェクトの取得、変更または削除

オブジェクトの属性を取得または変更、またはオブジェクトを削除するには getObject()、modifyObject()、または deleteObject() を呼び出します。 これらのメソッドはクラス SmDmsObject で定義されます。

たとえば、DN がディレクトリ ネームスペース dmsDir 内の ORG_ROOT によって参照される組織 org の属性を取得するには以下を実行します。

```
ORG_ROOT="o=swdev.com";
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmApiResult result = org.getObject();
```

オブジェクトの属性を変更するには、まず getObject() で既存の属性を取得します。その後、オブジェクトを追加する場合と同じように、setAttribute()（クラス SmDmsObject 内）を呼び出すことで新規属性（複数可）を設定します。 たとえば、上記の組織 org 内のユーザ USER_DN1 を、値 Boston に属性 *l* を設定することで修正するには、以下を実行します。

```
SmDmsUser user = org.newUser(USER_DN1);
result = user.getObject();
user.setAttribute("l", "Boston");
result = user.modifyObject();
```

objectclass 属性だけでなく、すべての属性に対して複数の値を修正できます。

前の例でユーザを削除するには、以下を実行します。

```
SmDmsUser user = org.newUser(USER_DN1);
result = user.deleteObject();
```

検索

LDAP ディレクトリおよび ODBC ディレクトリを検索できます。 クラス SmDmsOrganization 内の search... メソッドのいずれかを使用して組織を検索します。

以下のオブジェクトを使用して検索を定義します。

- 検索開始点、取得するレコードの最大数、および検索フィルタなどの検索パラメータを設定する SmDmsSearch。 以下のセクションでは、このオブジェクトの使用について説明します。
- オプションの並べ替えおよびページングの設定を定義できる SmDmsCursor。

ディレクトリを検索する場合に使用する検索パラメータを指定できます。 検索パラメータは以下の 2 回指定できます。

- 検索オブジェクトを作成するとき
- 検索オブジェクトを作成した後

一方または両方のオプションを使用できます。 これらは相互に排他的ではありません。

検索オブジェクトを作成する場合に検索パラメータを設定

検索オブジェクトを作成する場合に検索パラメータを設定するには、1 つ以上の検索パラメータ名を SmDmsSearch クラスのコンストラクタに渡します。

たとえば scope など、検索オブジェクトの作成中に指定できない検索パラメータがいくつかあります。 SmDmsSearch クラスのコンストラクタは以下の検索パラメータのみを受け入れます。

- フィルタ
- root
- propertyNames
- maxItems

コンストラクタに検索パラメータを渡すことなしに SmDmsSearch オブジェクトを作成できます。

検索オブジェクトの作成後に検索パラメータを設定

検索オブジェクトが作成された後、SmDmsSearch クラス内で set... メソッドを使用して以下を実行できます。

- 追加の検索パラメータの設定。
- 検索オブジェクトが作成された場合に設定する検索パラメータのリセット。

set... メソッドを使用することで、以下の表に示されるパラメータのいずれかをセットまたはリセットできます。

パラメータ	デフォルト	set メソッド	定義
classId	不明 (まだ設定されない)	setClassId()	クラス識別子。
フィルタ	" "	setFilter()	検索フィルタ、または検索する文字列。 検索オブジェクトが作成されるときにも設定できます。

パラメータ	デフォルト	set メソッド	定義
maxItems	50	setMaxItems()	一度に表示する結果セット項目の最大数。 検索オブジェクトが作成されるときにも設定できます。
nMaxResults	-1	setMaxResults()	結果セット用の項目の最大数。 たとえば、nMaxResults が 500 であるが、検索条件に 750 項目が一致する場合、最初の 500 の一致のみが検索から返されます。
nextItem	-1	setNextItem()	次回の前方向の検索で開始する項目。たとえば、以下のようになります。 nextItem += maxItems
previousItem	-1	setPreviousItem()	次回の後方向の検索で開始する項目。たとえば、以下のようになります。 previousItem-=maxItems
propertyNames	Null	setPropertyNames()	検索から返す必要のあるプロパティ。 検索オブジェクトが作成されるときにも設定できます。
root	" "	setRoot()	検索が開始するディレクトリエントリ。 検索オブジェクトが作成されるときにも設定できます。 LDAP 検索に対してのみ有効です。
スコープ	なし	setScope()	検索されるレベル。 LDAP 検索のみが対象です。
タイムアウト	-1	setTimeout()	秒単位の検索の最大の期間。

検索フィルタの設定

検索フィルタは、検索で取得するアイテムを定義します。SmDmsSearch コンストラクタ、または SmDmsSearch メソッド setFilter() を使用して検索フィルタを設定できます。

検索フィルタは、LDAP および ODBC ディレクトリに対して異なった方法で説明されます。

LDAP ディレクトリに対する検索フィルタの設定

LDAP ディレクトリで、SmDmsSearch コンストラクタまたは setFilter() メソッドの *filter* パラメータ内の完全な LDAP 検索フィルタを提供します。たとえば、グループに対する組織 swdev.com を検索するために SmDmsSearch コンストラクタに *filter* および *root* を渡す場合、以下を指定します。

```
SmDmsSearch search = new SmDmsSearch (
              "(&(objectclass=organizationalUnit) (ou=groups))",
               "o=swdev.com");
```

ODBC ディレクトリに対する検索フィルタの設定

ODBC ディレクトリの検索は SQL クエリによって実行されます。 DMS API は SQL SELECT ステートメントをサポートします。

検索フィルタで提供する情報は、検索が並べ替えおよびページング操作を提供するのに SmDmsCursor オブジェクトを使用するかどうかに依存しています。

- SmDmsCursor オブジェクトを検索メソッドに渡さない ODBC 検索では、検索フィルタ内に完全定義の SQL SELECT ステートメントを使用します。
- SmDmsCursor オブジェクトを検索メソッドに渡す ODBC 検索では、検索フィルタに、FROM および WHERE 節のみから構成される部分的な SQL SELECT ステートメントを使用します。

SmDmsCursor オブジェクトを検索メソッドへ渡す ODBC データベース検索では、DMS API はさまざまなソースからの完全な SQL SELECT ステートメントを以下のように組み立てます。

- FROM および WHERE 節は、SmDmsSearch コンストラクタまたは setFilter() メソッドの *filter* パラメータから取得されます。
- クエリの SELECT *列*の部分は、以下のパラメータのいずれかで指定された属性から取得されます。
  - setPropertyNames() の *propertyNames* パラメータ。これらの属性は、SmDmsSearch オブジェクトが SmDmsOrganization 内の検索メソッドのいずれかに渡される場合に使用されます。
  - getGroups() または getMembers() メソッドの *attrNames* パラメータ。
- ORDER BY *キーワード*部分は、SmDmsCursor コンストラクタの *sort* パラメータに指定する属性の順序から取得されます。

以下のコード フラグメントを考慮します。

```
String DIR_ROOT = "root";
String SRCH_FILTER ="from SmGroup";
SmDmsSearch search = new SmDmsSearch(SRCH_FILTER);
String[] prop = {"Name", "'Group' as Class"};
search.setPropertyNames(prop);
Vector SortOrder = new Vector();
SortOrder.add("uid");
SmDmsCursor cursor = new SmDmsCursor(SortOrder,blockSize,false,true);
```

DMS API は、以下の SQL ステートメントを構築するために前の例の情報を使用します。

```
SELECT Name, 'Group' AS Class FROM SmGroup ORDER BY uid ASC
```

コード ソース	SQL ステートメントの部分
SRCH_FILTER パラメータ SmDmsSearch コンストラクタ	from SmGroup
SortOrder パラメータ SmDmsCursor コンストラクタ	order by uid asc
prop パラメータ setPropertyNames()	select Name、'Group' as Class

組織の検索

DMS API で、検索は組織オブジェクト上で実行されます。

組織を検索するには以下を実行します。

1. 検索オブジェクトを作成します。 この検索オブジェクトは検索パラメータを保持します。

   たとえば、以下の SmDmsSearch コンストラクタの呼び出しは、グループを検索するための検索オブジェクトを作成します。 *root* パラメータは、o=swdev.org の開始点を指定します。

   ```
   SmDmsSearch mySearch = new SmDmsSearch (
              "(&(objectclass=organizationalUnit) (ou=groups))",
               "o=swdev.org");
   ```

   注: ルートは検索する SiteMinder ユーザ ディレクトリのトップ レベルです。必ずしもディレクトリ構造全体のトップ レベルではありません。

   SmDmsSearch クラスで set... メソッドを使用して、その他の検索パラメータを設定します。以下のようにします。

   ```
   mySearch.setScope(2);
   ```

2. オプションで、SmDmsCursor オブジェクト内で並べ替えおよびページングの設定を定義します。

3. 検索したい組織上のクラス SmDmsOrganization で search() メソッドを呼び出します。たとえば、以下のようにします。

   ```
   result = targetOrg.search (mySearch, 1);
   ```

   以下の表で示されるように、search() メソッドの 2 番目のパラメータは検索する方向を示しています。

方向	整数値
リセット	0
前方	1
後方	2
リフレッシュ	3

4. 検索から返される項目を取得するには、検索オブジェクト上で getResults() を呼び出します。たとえば、以下のようにします。

   ```
   Vector mySearchResults = search.getResults();
   ```

   結果ベクターの最初の要素には SmDmsSearchResultParams オブジェクト内の検索パラメータが含まれます。残りの要素は SmDmsObject オブジェクトです。オブジェクト タイプを区別するには、各オブジェクトの *classId* 属性が setClassId() メソッドによって設定されます。たとえば、*classId* が DMSOBJECT_CLASS_USER である場合、オブジェクトはユーザです。 classId が DMSOBJECT_CLASS_GROUP である場合、オブジェクトはグループです。

検索の例

以下の例は、search.set... メソッドを通して検索パラメータ セットを使用して、組織を検索します。 前方向検索の結果はベクター *vsearch* に割り当てられ、検索パラメータと共に出力されます。

```
SmDmsContext dmsContext = new SmDmsContext();
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmApiResult result = new SmApiResult();
SmDmsOrganization org = dmsDir.newOrganization (DIR_ROOT);

// Search
SmDmsOrganization test = org.newOrganization("");
SmDmsSearch search = new SmDmsSearch (
             "(&(objectclass=organizationalUnit) (ou=groups))",
              "o=swdev.com");
// Define search parameters
search.setScope(2);           // Number of levels to search.
search.setNextItem(0);        // Initialize forward search start
search.setMaxItems(20);       // Max number of items to display
search.setPreviousItem(0);    // Initialize back search start
search.setMaxResults(500);    // Max items in the result set
result = test.search(search, 1);
Vector vsearch = search.getResults();
System.out.println("Search object vector size " + vsearch.size());
SmDmsSearchResultParams searchParams =
                 (SmDmsSearchResultParams)vsearch.firstElement();
System.out.println("***Search Parameters***");
System.out.println(searchParams.toString());
System.out.println("removed element at 0");
vsearch.removeElementAt(0);
System.out.println("Search object vector size " + vsearch.size());
```

```
for (int i=0; i<vsearch.size(); i++)
{
   SmDmsObject dmsObj = (SmDmsObject)vsearch.elementAt(i);
   System.out.println("***Search**** " + dmsObj);
   printObject (dmsObj, result);
}

Hashtable attrs = dmsObj.getAttributes();
Enumeration keys = attrs.keys();
Enumeration values = attrs.elements();
while(values.hasMoreElements() )
```

以下のコード断片は、SmDmsCursor オブジェクトを通して並べ替えおよびページング機能を設定し、検索を実行します。 SmDmsSearch オブジェクト検索用のパラメータは前の例で同様の方法で定義されます。

```
Vector SortOrder = new Vector();
SortOrder.add("uid");
int blockSize = 20;
SmDmsCursor cursor=new SmDmsCursor(SortOrder,blockSize,false,true);
cursor.setOffset(15);
result = org.search(search, cursor, 1);     //Forward search
System.out.println(keys.nextElement() + " = " +
                                          values.nextElement() );
```

ユーザ パスワード状態

パスワード状態は、たとえばパスワードが最後に変更されたとき、およびパスワードが最後にユーザをログインするために使用されたときなど、特定のユーザのパスワードに関連するアクティビティを示しています。
ユーザに対する既存の SmDmsUserPWState を取得する、または属性が変更された新規パスワード状態オブジェクトを設定するには、SmDmsUser 内で getUserPWState() または setUserPWState() を呼び出します。

以下の表は、特定のユーザに対してアクセスできるパスワード状態属性、および属性値の設定または取得に使用されるメソッドをリストしています。 特に明記されないかぎり、すべてのメソッドはクラス SmDmsUserPWState にあります。

パスワード状態属性	メソッド	説明
ログインの失敗	setLoginFailures() getLoginFailures()	最後の正常なログインの後、ユーザがログインに失敗した回数を設定または取得します。
最終ログイン時間	setLastLoginTime() getLastLoginTime()	ユーザが最後に正常にログインした時間を設定または取得します。
前のログイン時間	setPrevLoginTime() getPrevLoginTime()	ユーザの最後から 2 番目の正常なログインの時間を設定または取得します。
無効化時間	setDisabledTime() getDisabledTime()	ユーザ オブジェクトが無効化された時間を設定または取得します。
パスワード履歴	SmDmsUser。 setUserPWState()	オプションで、ユーザのパスワード状態オブジェクトを設定するときに、ユーザのパスワード履歴をクリアします。 パスワード履歴を取得したり、パスワード履歴エントリを設定することはできません。
前回のパスワード変更時間	setLastPWChangeTime() getLastPWChangeTime()	ユーザ パスワードが最後に変更された時間を設定または取得します。

パスワード状態属性を変更する場合、変更は現在のパスワード状態オブジェクトのみに適用されます。 続けて取得される可能性のあるパスワード状態オブジェクトに変更を適用するには、現在のパスワード状態オブジェクトを SmDmsUser.setUserPWState() への呼び出しで受け渡します。このメソッドは、メソッドに渡される属性値を含む新規パスワード状態オブジェクトを設定します。

ODBC のサポート

ODBC ベースのユーザ ディレクトリに対して作業を行う場合、以下の DMS API メソッドを使用できます。

- SmDmsApiImpl.getDirectoryContext(SmUserDirectory, SmDmsConfig, SmDmsDirectoryContext)
- SmDmsDirectory.getCapabilities(Vector)
- SmDmsDirectory.getUserChallengeText(String)
- SmDmsDirectory.getUserTempPassword(String, String)
- SmDmsObject.getGroups(Vector, boolean)
- SmDmsObject.getObject()
- SmDmsObject.getObject(Vector)
- SmDmsOrganization のすべての search... メソッド
- SmDmsUser.authenticate(String)
- SmDmsUser.changePassword(String, String, boolean)

DMS ロールはサポートされていません。 ユーザおよびグループの追加および削除、グループへのユーザの追加、およびグループからのユーザの削除などの操作もサポートされていません。

制限されたメソッド

DMS API 内のメソッドの一部は、ユーザ権限階層の少なくともレベル以上で確立されたセッション内でのみ呼び出すことができます。 たとえば、ロールへのエンド ユーザの追加には、組織管理者セッション、Siteminder 管理者セッション、またはスーパー管理者セッションが必要です。

以下の表は、セキュリティ制限を持つ DMS メソッド（加えて apiutil パッケージ内の login() および logout() メソッド）、メソッドの呼び出しに必要な少なくとも権限レベル、およびメソッドの呼び出し元のクラスを示しています。

メソッド	最小権限レベルおよびクラス
addObject()	組織管理者セッション SmDmsObject およびサブクラス

メソッド	最小権限レベルおよびクラス
addToGroup()	組織管理者セッション SmDmsObject およびサブクラス
addToRole()	組織管理者セッション SmDmsUser クラス
authenticate()	エンドユーザ セッション SmDmsUser クラス
changePassword()	エンドユーザ セッション SmDmsUser クラス
deleteObject()	組織管理者セッション SmDmsObject およびサブクラス
getCapabilities()	エンドユーザ セッション SmDmsDirectory クラス
getDirectoryContext()	エンドユーザ セッション SmDmsApiImpl クラス
getDisabledState()	エンドユーザ セッション SmDmsUser クラス
getDmsContext()	SiteMinder 管理者セッション SmDmsApiImpl クラス
getDmsRoles()	組織管理者セッション SmDmsDirectory クラス
getGroups()	エンドユーザ セッション SmDmsObject およびサブクラス
getGroups()	組織管理者セッション SmDmsOrganization クラス
getMembers()	組織管理者セッション SmDmsGroup クラス
getMembers()	組織管理者セッション SmDmsRole クラス
getObject()	エンドユーザ セッション SmDmsObject およびサブクラス
getOrganizations()	組織管理者セッション SmDmsOrganization クラス

メソッド	最小権限レベルおよびクラス
getRoles()	エンド ユーザ セッション SmDmsUser クラス
getRoles()	組織管理者セッション SmDmsOrganization クラス
getUserChallengeText()	スーパー管理者セッション SmDmsDirectory クラス
getUserPWState()	エンド ユーザ セッション SmDmsUser クラス
getUserTempPassword()	スーパー管理者セッション SmDmsDirectory クラス
login()	セッションなし SmApiSession クラス
logout()	SiteMinder 管理者セッション SmApiSession クラス
modifyObject()	エンド ユーザ セッション SmDmsObject およびサブクラス
removeFromGroup()	組織管理者セッション SmDmsObject およびサブクラス
search()	組織管理者セッション SmDmsOrganization クラス
searchBack()	組織管理者セッション SmDmsOrganization クラス
searchForward()	組織管理者セッション SmDmsOrganization クラス
searchRefresh()	組織管理者セッション SmDmsOrganization クラス
setDisable()	組織管理者セッション SmDmsUser クラス
setDisabledState()	組織管理者セッション SmDmsUser クラス
setEnable()	組織管理者セッション SmDmsUser クラス

メソッド	最小権限レベルおよびクラス
modifyObjectClass()	組織管理者セッション SmDmsObject およびサブクラス
setPasswordMustChange()	エンド ユーザ セッション SmDmsUser クラス
setUserPWState()	エンド ユーザ セッション SmDmsUser クラス