Apache Directory ServerへのJavaオブジェクトの保管 第1回

Apache Directory Serverの内部

Bilal Siddiqui
Freelance consultant
WaxSys

2006年 5月 02日

この2部構成の記事では、ApacheDS (Apache Directory Server) にJava (TM) オブジェクトを保管する手順を詳しく説明します。この第1回目の記事では、Bilal SiddiquiがApacheDSを紹介し、コア・アーキテクチャーの概要を説明します。ApacheDSは主にJavaオブジェクトを保管するLDAPサーバーとして使用されるため、LDAPの概念と用語の概要について説明します。また、JXplorerを使用して属性タイプ、オブジェクト・クラスなどのLDAPスキーマ・コンポーネントを表示する方法、およびデータ・オブジェクトをApacheDSに格納する方法も示します。記事の最後に、第2回目の記事で説明する実践的な方法の準備として、ApacheDSへのJavaオブジェクトの保管時に適用されるJavaオブジェクトの直列化とRMI (Remote Method Invocation) の概要を説明します。

Apache Directory Serverはオープン・ソースで、多数のインターネット・プロトコルをJavaベースで実装したものです。ApacheDSの核となっているのは、アプリケーション・データを保管し、さまざまなタイプのデータに対して検索を実行できるディレクトリー・サービスです。プロトコル実装はディレクトリー・サービス上で動作し、データの保管、検索、および取得に関連するインターネット・サービスを提供します。

ApacheDSで最も重要な機能は、さまざまなプロトコルを使用してディレクトリー・サービスを公開できる機能でしょう。つまり、ApacheDSにアプリケーション・データ (ランタイムJavaオブジェクトを含む) を保管でき、さまざまなクライアントがさまざまなプロトコルを使用してそのデータを利用できるということです。ApacheDSによって実装されたプロトコルの中で最も重要なものは、LDAP (Lightweight Directory Access Protocol)です。ApacheDSはLDAPサーバーとして機能し、要求をlistenして内部のコア・ディレクトリー・サービスと連携してLDAP要求に応答します。

この2部構成の記事ではApacheDSのコア・アーキテクチャーを紹介し、ランタイムJavaオブジェクトをApacheDSに保管するすべての手順を説明します。ここからはLDAPサーバーの実装としてのApacheDSに重点を置いて説明するため、この第1回目の記事ではLDAPの機能と用語の説明が中心になります。ただしその前に、ApacheDSのモジュラー形式の拡張可能なアーキテクチャーを紹介し、新しいプロトコル実装とインターネット・サービスをApacheDSに組み込む方法を紹介しま

す。ApacheDSのコア・ディレクトリー・サービスの動作を理解すると、ApacheDSによってLDAP機能がどのように提供されているかを後で理解するのに役立ちます。

この記事の説明を理解するには、ApacheDSとJXplorerをダウンロードしてインストールする必要があります。この記事で紹介している完全なソース・コードは、いつでもダウンロードすることができます。

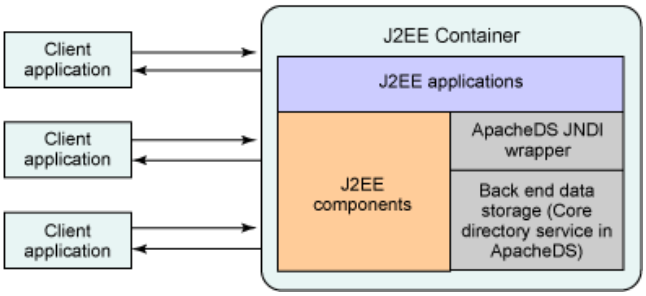
ApacheDSのディレクトリー・サービス

「ディレクトリー・サービス」とは、データを保管し整理するアプリケーションのことです。ディレクトリー・サービスでは、システムのユーザーの個人データ (名前、住所、電話番号など)、生産施設の製造能力 (数量、タイプ、設置された機械の生産能力) など、頻繁に更新する必要がないデータを処理します。後で、この2種類のデータを組み込むサンプル・アプリケーションを紹介します。ここでは、ApacheDSがどのようにディレクトリー・サービスを提供するかについて説明します。

ApacheDSによるJNDIの実装

図1は、コア・ディレクトリー・サービス用のJNDI (Java Naming and Directory Interface) ラッパーを実装するApacheDSを示しています。JNDIは、ディレクトリー操作 (ディレクトリーへのデータ保管や保管データの検索など) を実行するメソッドを定義するJavaインターフェースです。JNDIは、J2EE (Java 2 Enterprise Edition) とJ2SE (Java 2 Standard Edition) の一部です。J2SEではクライアント側のJNDIしかサポートされていませんが、通常の場合、J2EEコンテナーにはサーバー側のJNDI実装が含まれています。図1のように、J2EEコンテナーではJNDIラッパーを介してApacheDSのディレクトリー・サービスを使用することができます。

図1. J2EEコンテナー内で動作するApacheDS



JNDIのインターフェース・セットは、ディレクトリー・サービスを抽象化します。JNDI実装には、ディレクトリー・サービスとやり取りする実際のロジックが用意されています (たとえば、JavaプラットフォームにはLDAP用のJNDI実装が付属しています)。JNDIを使用すると、あらゆるタイプのディレクトリー・サービスとのやり取りが可能になります (そのディレクトリー・サービス用のJNDIがある場合に限ります)。Javaベースのクライアント・アプリケーションでJNDIを使用する場合、クライアント側のJNDI実装が必要になります。クライアント側のJNDI実装には、ディレクトリー操作の要求を作成するJNDIインターフェースを実装するクラスが用意されています。

ApacheDSは、「サーバー側のJNDI」を実装します。つまり、ApacheDSには、ディレクトリー操作の要求に応答するJNDIインターフェースを実装するクラスが含まれているということです。上

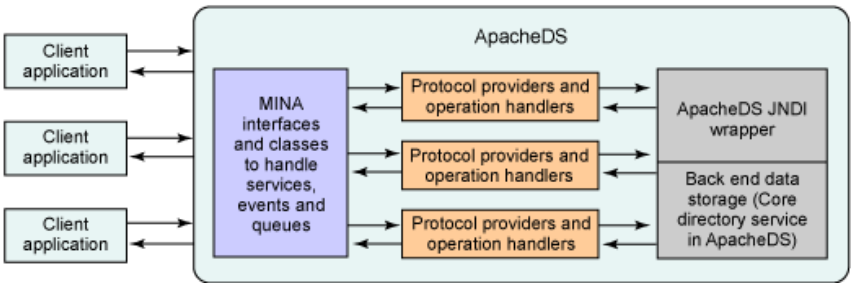
で説明したように (図1を参照)、J2EEコンテナーではJNDIラッパーを介してApacheDSのディレクトリー・サービスを使用することができます。

組み込み可能なプロトコルのサポート

図1に示したのは、ApacheDSの使用モデルの1つにすぎません。ApacheDSの用途は、J2EEコンテナー内に組み込まれたディレクトリー・サービスだけとは限りません。ApacheDSを使用し、バックエンド・ディレクトリー・サービスが必要なプロトコルを実装することができます。さまざまなタイプのプロトコルを同時に機能させることも可能です。たとえば、現在のApacheDS実装では、LDAPとKerberosが実装されています。さらに、ApacheDSでサポートされるプロトコルのリストは、現在も増え続けています。

ApacheDSは、柔軟性の高い拡張可能なアーキテクチャーを備えています。このアーキテクチャーは、新しいプロトコルの実装を可能にするものです。図2は、ApacheDSアーキテクチャーのモデルを示しています。このモデルは、図1に示したJNDIラッパー上で動作します。

図2. ApacheDSの柔軟性の高い、拡張可能なアーキテクチャー



この図を見てわかるように、ApacheDSではMINA (Multipurpose Interfaces for Networked Applications) というインターフェースの集合が使用されています。MINAを使用すると、新しいプロトコル実装をApacheDSに組み込むことができます。ここで、MINAがどのように動作するかを説明します。

MINAの動作

MINAのインターフェースには、プロトコル固有のファクトリー・オブジェクトを生成するメソッドが含まれています。これらのファクトリー・オブジェクトには、新しいプロトコル実装をApacheDSに組み込む方法が用意されています。プロトコル実装では、MINAインターフェースが実装されます。ApacheDSフレームワークは、MINAに設定されているメソッドを利用してプロトコル実装とやり取りを行います。

たとえば、MINAにはgetCodecFactory() メソッドがあるProtocolProviderという名前のインターフェースがあります。ProtocolProvider.getCodecFactory() メソッドは、ProtocolCodecFactoryという名前の別のMINAインターフェースを公開するオブジェクトを返します。

ApacheDSのプロトコル実装では、ProtocolProviderインターフェースが実装されます。
たとえば、ApacheDSのLDAP実装には、ProtocolProviderインターフェースを実装するLDAPProtocolProviderという名前のクラスがあります。

LDAPProtocolProviderのgetCodecFactory()メソッドは、ProtocolCodecFactoryインターフェースを公開するオブジェクトを返します。このProtocolCodecFactoryは、ApacheDS フレームワークで

LDAP固有のエンコード・オブジェクトとデコード・オブジェクトの作成に使用されるファクトリー・オブジェクトです。

ProtocolCodecFactoryには、newEncoder()メソッドとnewDecoder()メソッドがあります。これらのメソッドは、MINAのProtocolEncoderインターフェースとProtocolDecoderインターフェースを公開するオブジェクトを返します。プロトコル固有のエンコード・オブジェクトはProtocolEncoderインターフェースを公開し、デコード・オブジェクトはProtocolDecoderインターフェースを公開します。

MINAのエンコードとデコードのフレームワーク

ApacheDSフレームワークは、プロトコル固有のProtocolDecoderインスタンスを使用してプロトコル要求をデコードし、要求を処理する前にその意味を理解できるようにします。デコード後、ApacheDSによって要求が処理されます。たとえば、要求がLDAP検索要求である場合、ApacheDSはバックエンド・ディレクトリー・サービスで要求されたデータを検索し、検索結果を取得します。

ApacheDSフレームワークは、要求された検索結果を取得した後、プロトコル固有のProtocolEncoderオブジェクトを使用して検索結果をエンコードします。LDAP検索要求の場合、ApacheDSはLDAP固有のProtocolEncoderオブジェクトを使用して検索結果をエンコードした後、要求元クライアントに応答を送信します。

MINAのサービス・フレームワーク

MINAには、サービスを処理するクラスもあります。すべてのサービス・プロバイダーは、自分自身をサービス・レジストリーに登録することができます。プロトコル・プロバイダーは、サービスを提供するプロバイダー・クラスと共にサービス・レジストリーに登録されます。次に、プロトコル・プロバイダーはプロトコル要求をJNDI操作にマップします。簡単な例として、JNDIの検索操作にマップされたLDAP検索要求があります。プロトコル要求の処理中にどのJNDI操作を呼び出せばいいのか、ということをApacheDSフレームワークがひとたび理解すると、次からはApacheDSフレームワークによるイベントの生成が可能になります。

MINAのイベント処理フレームワークは、該当するハンドラーにイベントを送信します。たとえば、JNDI検索操作の呼び出しが要求された場合、検索ハンドラーが呼び出されます。また、MINAはスレッドのプールも保持します。ハンドラーが前の操作の処理でビジー状態になっている場合、イベントは処理されるまでの間、一時的にスレッド・プールに保管されます。

図2に、JNDIで使用されるプロトコル・プロバイダー、MINAインターフェースとクラス、操作ハンドラーを示します。

ApacheDSアーキテクチャーの最大の利点は、さまざまなプロトコル・プロバイダーに対して共通のディレクトリー・サービス (JNDI) を使用することにあると言えるでしょう。つまり、ApacheDSを使用した場合、さまざまなプロトコルを使用するクライアントにデータを公開できるということです。ApacheDSでサポートされているプロトコルの中で最も重要なものの1つはLDAPであるため (また、ApacheDSは主にJavaオブジェクトを保管するLDAPサーバーとして使用されるため)、ここからはLDAPの詳しい説明に入ります。ApacheDSアーキテクチャーの詳細については、「参考文献」を参照してください。

LDAPの概要

LDAPプロトコルは、ディレクトリー操作の要求メッセージと応答メッセージを定義します。ディレクトリー操作には、ディレクトリーへの新しいデータの保管、保管データの検索と取得、不要なデータの削除、古いデータの更新などがあります (参考文献」からRFC 2251を参照してください。RFC 2251は、LDAPディレクトリー内のデータの保管、検索、更新、削除を行う要求/応答メッセージを定義する正式なLDAP仕様です)。

ApacheDSに新しいデータ (ランタイムJavaオブジェクトなど) を保管する際に使用するLDAPメッセージは、「バインド」メッセージと呼ばれます。バインド・メッセージはユーザー・データをApacheDSなどのLDAPディレクトリー・サービスに転送し、ディレクトリーにデータを格納します。

LDAPでは、データを保管する物理的な場所は問題となりません。その代わり、ApacheDSに保管されるすべてのデータ・エントリーに識別名 (DN) が指定されます。すべてのDNは、ディレクトリー・サービス内で一意の値でなければなりません。2つのエントリーが同じDNを持つことはできません。各DNを一意にするためのLDAPの仕組みについては、後で説明します。

さらに、LDAPの検索機能でもDNが使用されます。次のセクションでは、LDAPの用語に慣れるため、アプリケーション・シナリオの例を紹介します。LDAP検索の仕組みについても説明します。このサンプル・アプリケーションは、次回の記事でも使用します。

学習するアプリケーション

このアプリケーション・シナリオをわかりやすくするため、ある製造会社向けにデータ管理システムを設計することにしましょう。この会社には、従業員、顧客、パートナー、ベンダーがいます。全員が、データ管理システムのユーザーです。データ管理システムでは、ユーザーに関するデータをApacheDSに保管する必要があります。

このシステムでは、すべてのユーザーがシステムの使用方法を独自に設定することができます。たとえば、データをカスタマイズしてシステムの使用時に個人的なデフォルト・ビューでデータを表示でき、異なるデータ要素には異なる表示スタイルを適用できます。また、ユーザーの「タイプ」に従って、特殊な設定を使用することもできます。たとえば、会社の従業員 (社内ユーザー) はメッセージに関する設定、顧客は出荷に関する設定、ベンダーは送り状に関する設定をそれぞれ行うことができます。

ApacheDSで個々のユーザー設定を定義する簡単な方法は、Javaオブジェクトの形で設定を保管することです。このアプリケーション・シナリオでは、最初にすべてのタイプのユーザー用にPreferencesクラスを設計します。Preferencesクラスには、すべてのタイプのユーザー (この場合、社内ユーザー、顧客、ベンダー) に共通する設定をユーザーが指定できるようにするメソッドが含まれます。たとえば、Preferencesクラスには、スタイルシートの場所を指定するsetStyles() メソッドが含まれます。スタイルシートは、さまざまなデータ要素に表示スタイルを適用するときに使用します。

また、Preferencesクラスを拡張してMessagingPreferencesクラスを形成することもできます。このクラスには、社内ユーザーのメッセージング設定が含まれます。同様に、顧客用のShippingPreferencesクラスとベンダー用のInvoicingPreferencesを設計することができます。

リスト1は、Preferencesクラス、MessagingPreferencesクラス、ShippingPreferencesクラス、InvoicingPreferencesクラスのスケルトンです。内容を簡単にするため、リスト1にはメソッドが含まれていません (setStyles() メソッドを除きます)。ここでは、ApacheDSのクラス・インスタンスの保管を示します。

リスト1. さまざまなタイプのユーザー設定を表すJavaクラス

```
public class Preferences implements java.io.Serializable {
    String styleSheetURL = null;

    public void setStyles(String styleSheetURL){
        this.styleSheetURL = styleSheetURL;
    }
    //Other methods of the Preferences class

}

public class MessagingPreferences extends Preferences {
    //Methods of the MessagingPreferences class
}

public class ShippingPreferences extends Preferences {
    //Methods of the ShippingPreferences class
}

public class InvoicingPreferences extends Preferences {
    //Methods of the InvoicingPreferences class
}
```

ApacheDSを使用したデータ管理

用語に関する注意

DNは、ディレクトリー・サービスの「名前付きコンテキスト」と考えることもできます。Aliceなどのユーザーのデータ・エントリーは、ユーザーのDNによって定義された名前付きコンテキストに書き込まれます。実際に、「DN」と「名前付きコンテキスト」という用語が区別なく使用されることがよくあります。通常、JNDIのドキュメントでは名前付きコンテキスト (または命名コンテキスト) が使用され、LDAPのドキュメントではDNが使用されます。LDAPとJNDIの両方を使用する場合、両方の用語は同じ意味を持つと考えることができます。

Aliceというユーザーが、ある製造会社の営業部門に勤務しているとします。Aliceはデータ管理システムを使用し、自分のすべてのデータ (名前、部門、電子メール・アドレス、電話番号など) や設定 (MessagingPreferencesオブジェクト形式) をApacheDSに保管します。データはすべて、一意のDNが設定された状態でApacheDSに保管されています。

ここで、Aliceはデータ管理システムを使用してメッセージング設定を変更するとします。データ管理システムでは、最初にLDAPを使用してAliceの名前付きコンテキストを検索し、そのDNを調べます。DNを認識すると、DNからAliceのMessagingPreferencesオブジェクトを取得してAliceの最新データでオブジェクトを更新し、オブジェクトをApacheDSに保管し直します。

アプリケーション・シナリオについて理解したら、次はどのようにApacheDSを使用してこうしたサーバー側の作業を行うかを見てみましょう。

ApacheDSを開始する

ApacheDSに任意のタイプのデータ (Javaオブジェクトを含む) を保管する方法を理解するには、LDAPスキーマを理解する必要があります。さらに、LDAPサーバーに保管されたデータをツリー形式のグラフィカル表示にすると便利です。そのため、JXplorerの使用方法を説明します。JXplorerは、Javaベースのオープン・ソースのクライアント側LDAP実装です。LDAPサーバーに保管されたデータのブラウザ表示を行います。JXplorerを使用してLDAPスキーマを紹介するため、最初にJXplorerをダウンロードしてインストールする必要があります。

ApacheDSをダウンロードしていない場合は、ここでApacheDSもダウンロードする必要があります。インストールは簡単です。取得したzipファイルをunzipしてApacheDSのJARファイルを抽出したら、以下のようにコマンド行からapacheds-main-0.9.jarを実行してApacheDSのLDAPサーバーを起動してください。

```
<JAVA_HOME>/java -jar apacheds-main-0.9.jar
```

これで、クライアント・アプリケーションからのLDAP 要求に応えるため、ApacheDSのLDAPサーバーがlocalhost:389 (デフォルト設定) でlisten状態になります。

ApacheDSに接続する

ApacheDSの起動後、JXplorerを実行すると図3のようなブラウザが表示されます。この図からわかるように、JXplorerはまだLDAPサーバーに接続されていません。

図3. JXplorer起動時の最初の表示

次に、ApacheDSに接続する必要があります。この場合、JXplorerの「File」メニューにある「Connect」コマンドを使用します。図4に示すように、接続ダイアログ・ボックス (「Open LDAP/DSML Connection」) に値を入力します。

図4. 接続ダイアログ・ボックス

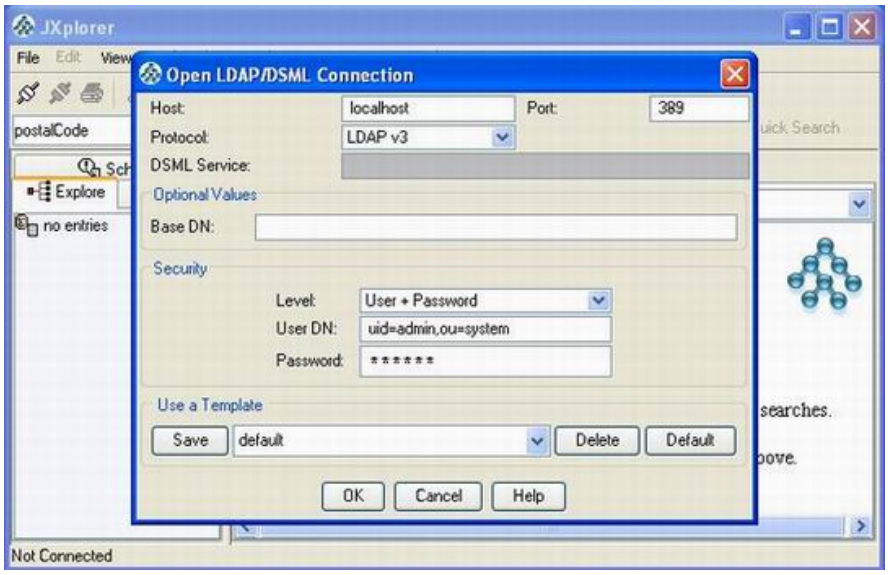
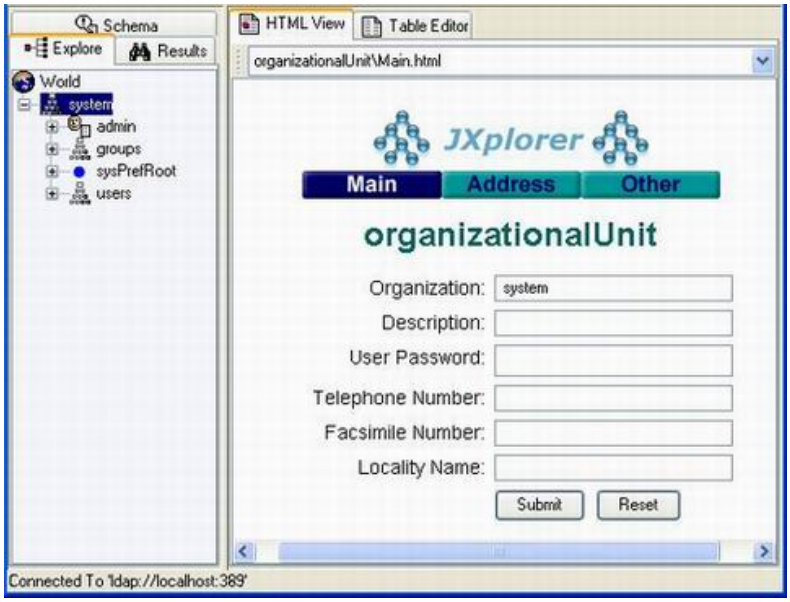


図4で、ホストとポートのフィールドには、ApacheDSがlistenしているアドレスを指定することに注意してください。また、ApacheDSに接続するには、ユーザー名とパスワードも必要です。ApacheDSにはデフォルトの管理ユーザー名 (uid=admin,ou=system) とパスワード(「secret」)のペアが用意されているので、この値を使用してApacheDSに接続することができます。ここでは、「User DN」フィールドにデフォルト・ユーザー名を、「Password」フィールドにデフォルト・パスワードを入力しています。

接続ダイアログ・ボックスに値を入力した後、「OK」をクリックします。JXplorerがApacheDSに接続要求を送信した少し後に、図5のような画面が表示されます。これでApacheDSに接続されました。

図5. ApacheDSへの接続後、最初のJXplorer画面



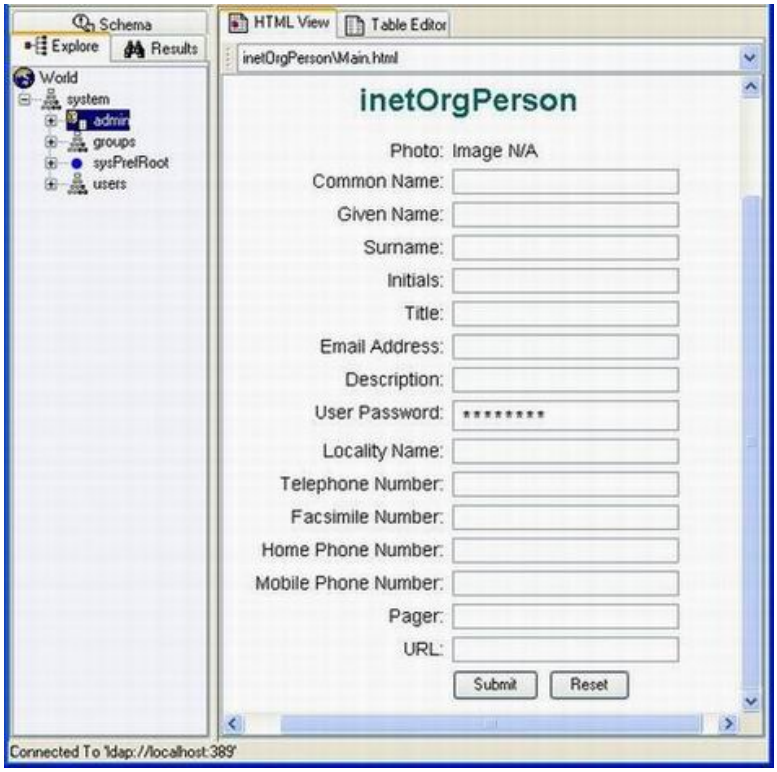
ApacheDSのデフォルト・データ

図5の画面は、Windows Explorer画面のように2つの部分に分かれています。左側はツリー表示になっています。右側には、ユーザーがこのツリー表示で選択した項目の詳細が表示されます。

左側には、「Explore」、「Results」、「Schema」という3つのタブがあります。「Explore」タブでは、ApacheDSにあるデータを調べることができます。「Explore」タブは、ApacheDSのデータ・エントリーを表示する場合に使用します。「Results」タブには検索結果が表示され、「Schema」タブにはApacheDSでサポートされているスキーマの詳細が表示されます。この説明では、主に「Explore」タブと「Schema」タブを使用します。

図5では「Explore」タブが選択され、ApacheDSに含まれているデータの詳細情報が表示されています。ApacheDSにはまだデータを保管していないため、デフォルト・データが表示されています。たとえば、図5の「admin」エントリーをクリックすると、図6のような画面が表示されます。ここには、ApacheDSと共にロードされたデフォルトの管理エントリーが表示されます。

図6. adminエントリーの詳細情報



adminエントリーはApacheDSの管理者を表します。この管理者のユーザー名とパスワードは、図4で入力しました。図5の右側部分はadminエントリーのHTML表示を示しています。「User Password」フィールドなど、いくつかのフィールドがあります。管理者パスワードを変更するには、「User Password」フィールドに新しいパスワードを入力して「Submit」ボタンをクリックします。

属性タイプについての注意

図6の右側の各エントリーは、実際にはLDAP属性です。LDAPには多くのタイプの属性が定義されています。属性のタイプはLDAP用語ではattributeTypeと呼ばれます。

属性にはデータ値が入ります(たとえば、「Common Name」フィールドには値としてApacheDS管理者の名前が入ります)。保管するデータのさまざまな側面を、属性値として定義する必要があります。たとえば、属性にテキスト・データを入れるか未加工のバイナリー・データを入れるかといった、データ・エンコードを指定する必要があります。

データの使用でもう1つ重要な点は、データ検索ができることです。たとえば、名前または電子メール・アドレスがわかっているユーザーに関連するすべてのデータを検索する場合、検索中に適用されるマッチング規則を指定する必要があります。電子メール・アドレスの場合、大文字と小文字を区別せずに検索した方がいいでしょう。

JXplorerでは、デフォルト管理エントリーの定義に使用される属性タイプを表示できます。図6の画面の右側にある「Table Editor」タブをクリックすると、図7のようなTable Editorが表示されます。

図7. adminエントリーの「Table Editor」ビュー

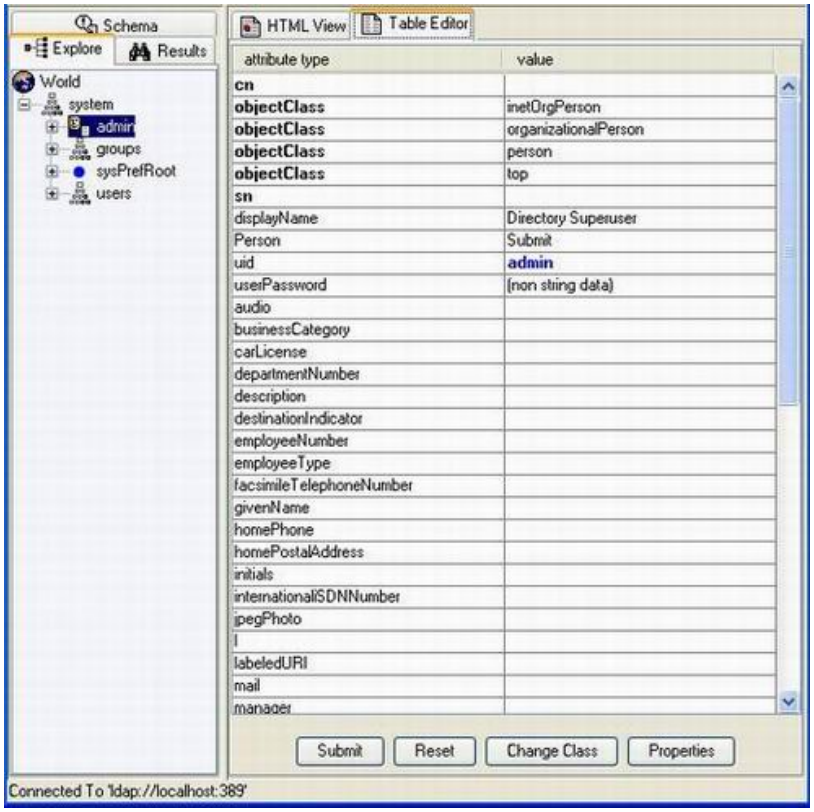


図7の表には、「attribute type」列と「value」列があります。表示されたいくつかの属性タイプは、図6のフィールドにマップすることができます。たとえば、図7の属性タイプ「cn」と「sn」は、図6のフィールド「Common Name」と「Surname」にそれぞれ対応します。

つまり、データ・エントリーの属性には属性タイプが関連付けられているということです。ApacheDSでは、広範囲の目的に対応した、さまざまなインターネット標準で指定された数十の属性タイプがサポートされています。この記事では、ApachDSでサポートされている属性タイプのうち、Javaオブジェクトの保管に使用されるものを説明します。特に、cnを多く使用します。これは、エンティティーの共通名の指定に使用される多目的な属性タイプです。cn属性タイプを

使用すると、企業の営業部門に所属する管理職の名前を指定することができ、Javaオブジェクトも指定することができます。図7の画面を見るとわかりますが、sn属性タイプは汎用ではなく、人の姓を指定する場合にのみ使用します。

ApacheDSのオブジェクト・クラス

属性タイプについて少し見たところで、次にobjectClass属性タイプを調べてみましょう。図7にはinetOrgPerson、organizationalPerson、person、topの4つの値があります。つまり、adminエントリーには4つの「オブジェクト・クラス」があります。

実際には、ApacheDSのすべてのデータ・エントリーではオブジェクト・クラスが使用されます。オブジェクト・クラスは属性タイプの集合です。データ・エントリー (図6と図7のadminエントリーなど) がオブジェクト・クラスに関連付けられている場合、エントリーにはそのオブジェクト・クラスの属性タイプがすべて含まれます。

adminエントリーで使用されている4つのオブジェクト・クラスに含まれる属性タイプを実際に調べてみましょう。図7の画面の左側にある「Schema」タブをクリックすると、図8に示す画面が表示されます。「Schema」タブには、ApacheDSでサポートされている属性タイプ、オブジェクト・クラス、データ・フォーマットに関する情報が表示されます。

図8. 「Schema」タブ

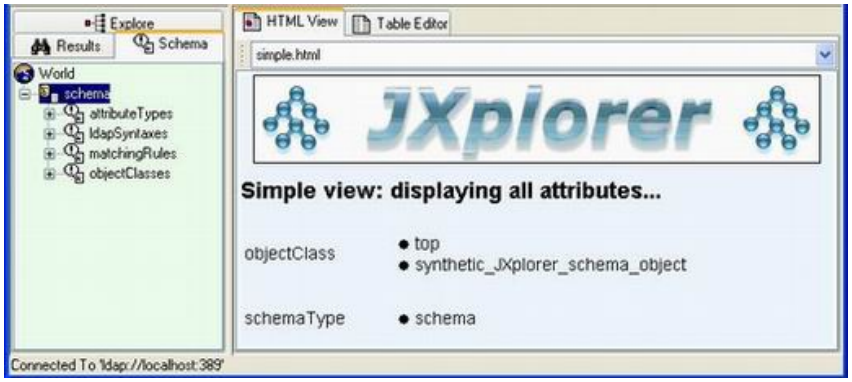
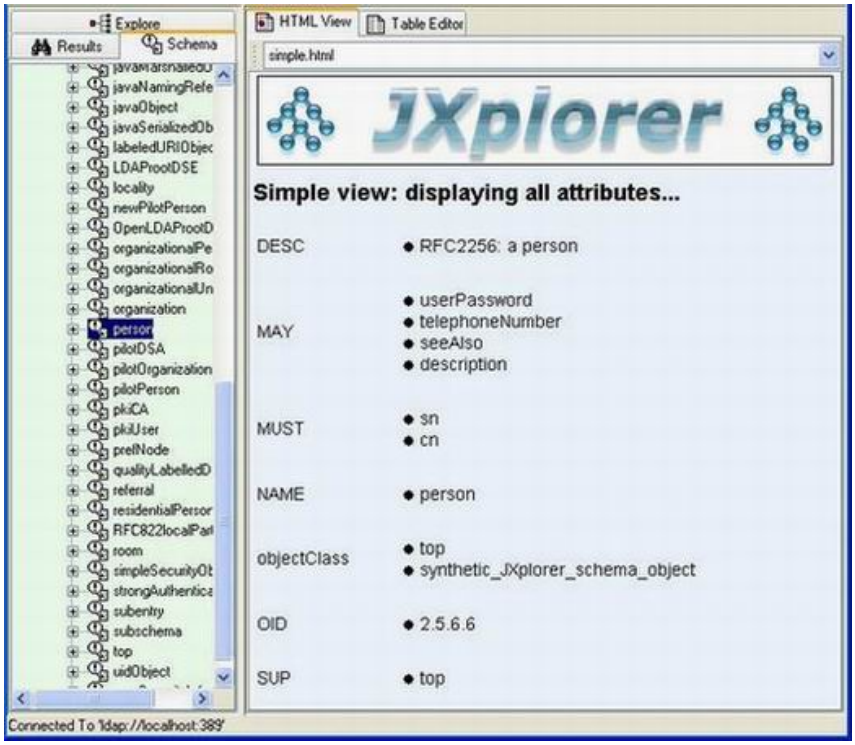


図8に示した画面の左側の「objectClasses」エントリーを展開します。エントリーは、オブジェクト・タイプのアルファベット順に長いリストとして展開されます。これは、ApacheDSでサポートされているリスト形式です。このリストから「person」オブジェクト・クラスを探してクリックすると、図9に示す画面が表示されます。

図9. personオブジェクト・クラス



オブジェクト・クラスを定義する

図9には、さまざまなフィールドの形式でpersonオブジェクト・クラスの詳細情報が表示されています。オブジェクト・クラスを定義するフィールドは以下のとおりです。

表1. オブジェクト・クラスを定義するフィールド

DESC
: personオブジェクト・クラスの説明です。personオブジェクト・クラスはRFC 2256 (「参考文献」を参照) に定義されています。RFC 2256には、LDAPディレクトリー・サービスのクライアントやユーザーによって使用されるさまざまなオブジェクト・クラスと属性タイプが記述されています。たとえば、誰でもLDAPサービスのユーザーになることができるため、personオブジェクト・クラスでは、すべての人に対して設定可能な属性 (氏名など) が定義されます。

MAY
: personオブジェクト・クラスに含まれるオプションの属性タイプのリストを指定します。たとえば、personオブジェクト・クラスには、個人のパスワードと電話番号の格納に使用される属性タイプuserPasswordとtelephoneNumberが含まれることがあります。後で、属性タイプの詳細情報を指定する方法を説明します。

MUST
: personオブジェクト・クラスの必須属性タイプを指定します。personオブジェクト・クラスには、すでに説明した2つの必須属性タイプsnとcnしかありません。

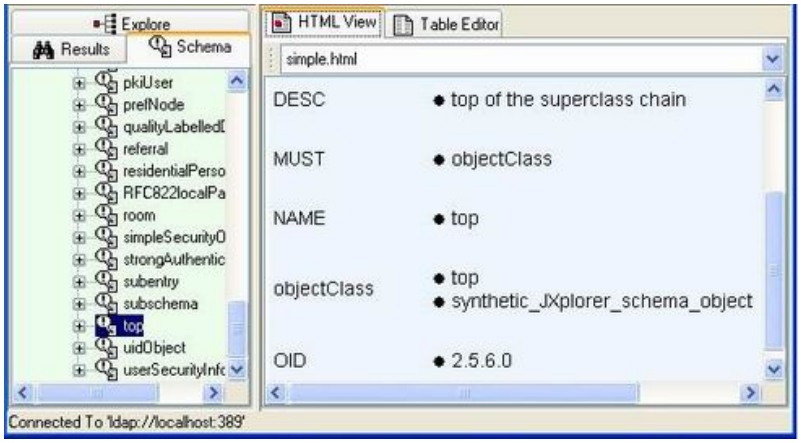
NAME
: オブジェクト・クラスの名前を指定します。

OID
: オブジェクト・クラスのオブジェクト識別子を指定します。すべてのLDAPオブジェクト・クラスと属性タイプには、一意のオブジェクト識別子が必要です。RFC 2256では、personオブジェクト・クラスのオブジェクト識別子が指定されます。オブジェクト識別子を一意にするため、Internet Assigned Numbers Authority (「参考文献」を参照) では、オブジェクト識別子を無料で発行しています。この記事では新しい属性タイプを定義しないため、オブジェクト識別子を取得する必要はありません。

SUP
: オブジェクト・クラスの親を指定します。オブジェクト・クラスの継承の概念 (親の機能を子に拡張する) は、オブジェクト指向プログラミング言語の継承に似ています。図9のSUP属性値からわかるように、personオブジェクト・クラスではすべてのオブジェクト・クラスのスーパー・クラスであるtopクラスが拡張されています。つまり、LDAPのすべてのオブジェクト・クラスは、直接的または間接的にtopオブジェクト・クラスを拡張していることになります。

図10に示したtopオブジェクト・クラスを見てください。1つの属性タイプobjectClassしか含まれていません。すべてのオブジェクト・クラスはtopクラスから拡張されているため、データ・エントリーでどのオブジェクト・クラスが使用されているかに関係なく、objectClass属性タイプはすべてのデータ・エントリーに常に存在します。簡単に言うと、すべてのデータ・エントリーで、使用するオブジェクト・クラスを定義する必要があるということです。

図10. topオブジェクト・クラス



オブジェクト・クラスのタイプ

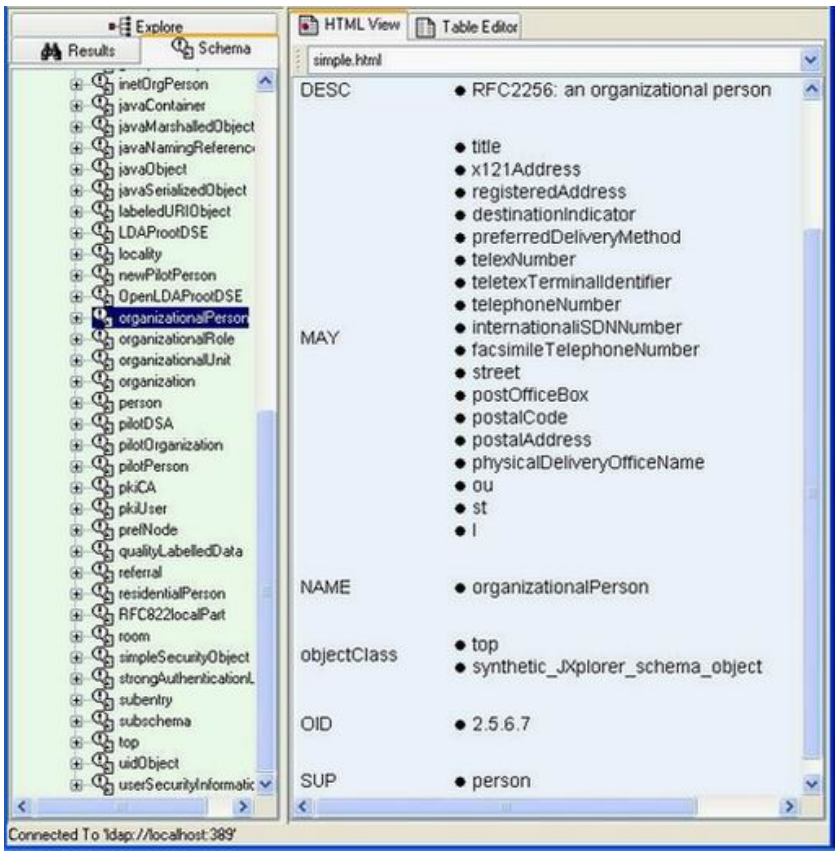
オブジェクト・クラスには抽象、構造、補助の3つのタイプがあります。この場合、topオブジェクト・クラスは抽象クラスになります。このクラスは、ほかのクラスによる拡張を可能にするためのみ存在します。データ・エントリーで直接抽象クラスを使用することはできません。

一方、personオブジェクト・クラスは構造クラスです。すべてのデータ・エントリーは構造クラスを使用します。構造クラスは、ほかの構造クラスと抽象クラスを拡張します。たとえば、organizationalPersonという構造クラス (RFC 2256で定義済み) はpersonオブジェクト・クラスを拡張します。一方、personオブジェクト・クラスはtopオブジェクト・クラスを拡張します。

organizationalPersonオブジェクト・クラスは、ある組織に所属する特定のタイプの人間を表します。したがって、組織に所属する人間に適用できる属性を定義します (たとえば、従業員の職務

名など)。図11に、JXplorer画面に表示されたorganizationalPersonオブジェクト・クラスを示します。

図11. organizationalPersonオブジェクト・クラス



補助オブジェクト・クラスは、特殊な目的に使用されます。補助オブジェクト・クラスには、ほとんどすべてのデータ・エントリーに必要な汎用属性タイプ (上で説明した属性タイプcnとobjectClassなど) は含まれません。

データ・エントリーは、補助オブジェクト・クラスに全面的に依存することはできません。したがって、補助オブジェクト・クラスを使用するデータ・エントリーでは、少なくとも1つの構造クラスを使用する必要があります。後でApacheDSにJavaクラスを保管する方法を説明しますが、その際に補助クラスの例を示します。

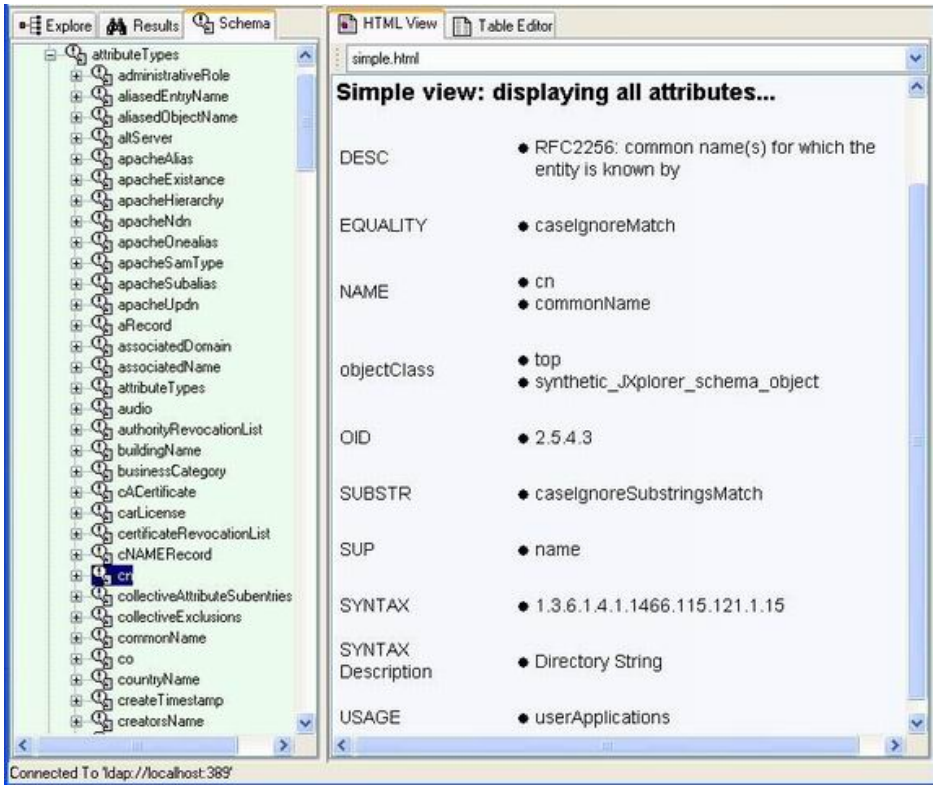
ApacheDSの属性タイプ

次に、ApacheDSの属性タイプの使用方法をさらに詳しく見てみましょう。図8に示したJXplorer画面のattributeTypesエントリーを展開すると、さまざまな属性タイプが図12の画面に表示されます。これらの属性タイプは、ApacheDSでサポートされているいくつかの仕様で定義されています。たとえばRFC 2713では、LDAPディレクトリーにJavaオブジェクトを保管するための属性タイプとオブジェクト・クラスが定義されています。「参考文献」を参照してください。

図12. ApacheDSでサポートされている属性タイプ

図12に示された「cn」属性エントリーをクリックすると、図13のような画面が表示されます。図13には、1つの属性タイプを定義するフィールドの集合が表示されます。

図13. cn属性タイプ



personオブジェクト・クラスに関する説明で、上に表示されたフィールドのいくつかについてはすでに説明しました。ここでは、まだ説明していないフィールドだけを紹介します。

表2. その他のフィールド

EQUALITY
: 特定の属性値を持つデータ・エントリーを検索するときに適用されるマッチング規則を指定します。cn属性タイプに対するこのフィールドの値はcaseIgnoreMatchです。この属性タイプと値を使用すると、特定の名前を持つ人を検索する際に、大文字と小文字を区別せずに検索されます。

SUBSTR
: 「EQUALITY」フィールドに似ていますが、SUBSTRフィールドでは、属性値全体ではなく、指定したサブストリングを探すための検索マッチング規則を指定します。SUBSTRフィールドの効果については、この記事の第2部で説明します。

また、OID値が入った図13の「SYNTAX」フィールドにも注意してください。OID値は、属性値の構文 (またはデータ・フォーマット) を識別します。すべての属性タイプは、属性値として保管されるデータの構文を定義します。図13のOID値は、文字列をLDAPディレクトリーへ保管する際に使用されるLDAP構文を指しています。次のセクションでは、LDAP構文について説明します。

ApacheDSのLDAP構文

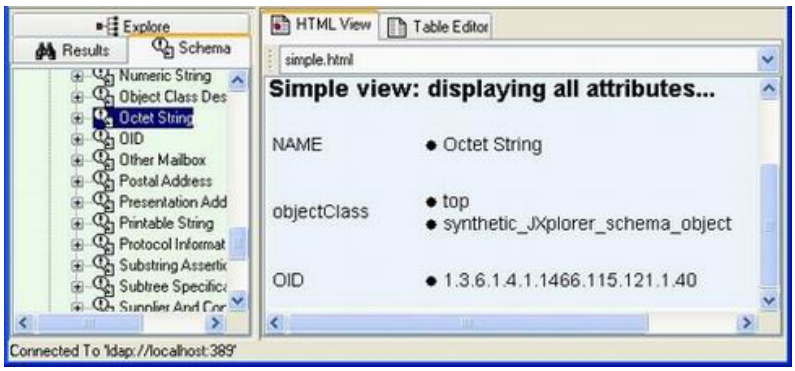
JXplorerの「Schema」タブをクリックし、そのタブの「ldapSyntaxes」エントリーをクリックすると、ApacheDSでサポートされているLDAP構文のアルファベット順リストが表示されます。このリストから「Directory String」というエントリーを探してクリックすると図14に示す画面が表示され、LDAP構文を定義するフィールドが表示されます。

図14. Directory String構文

Directory String構文は、LDAPディレクトリーに文字列値を保管します。この構文のフィールドはいずれも、これまでに説明したフィールドと同じような機能を持っているため、その意味は簡単に理解できると思います。ただし、図14のOIDフィールドは、図13のcn属性のSYNTAXフィールドと正確に一致することに注意してください。これは、cn属性タイプがDirectory String LDAP構文に従うためです。つまり、LDAPは名前を文字列として扱うということです。

また、図15に示す別のLDAP構文Octet Stringは、オクテット文字列を表しています。Javaオブジェクトはオクテット文字列として保管されるため、このシリーズ記事ではOctet String LDAP構文を使用します。

図15. Octet String構文



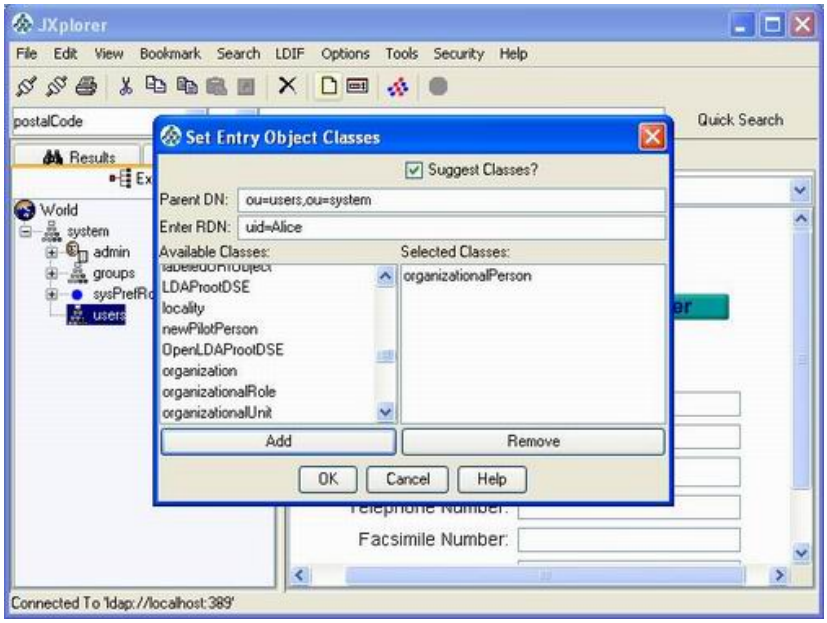
Directory String構文とOctet String構文を見たところで、次はApacheDSを使用したデータ・エントリーの作成手順について見ていきましょう。

ApacheDSにデータを入力する

ここまでは、LDAPコンポーネントについて見てきました。また、新しいデータ・エントリーをApacheDSに書き込む場合、どのようにLDAPコンポーネントをApacheDSに実装するかについても見てきました。ここからは、オブジェクト・クラス、属性タイプ、構文などのスキーマ・コンポーネントによるLDAPディレクトリーへのデータ・エントリーの保管方法や表現方法について説明します。

最初に「JXplorer Explore」タブをクリックし、そのタブの「Users」エントリーをクリックします。ApacheDSにはデフォルト・ユーザーが設定されていないため、「Users」エントリーは空になっています。エントリーを右クリックすると、ポップアップ・メニューが表示されます。ポップアップ・メニューから「New command」を選択すると、図16に示すように「Set Entry Object Classes」というダイアログ・ボックスが表示されます。

図16. 「Set Entry Object Classes」ダイアログ・ボックス



最初に、ダイアログ・ボックスに情報を指定する必要があります。その方法については、次のセクションで説明します。

識別名構文を使用する

LDAPスキーマの説明の最初で述べたように、ApacheDSのすべてのデータはツリー形式で保持されています。つまり、ルート・エントリーを除き、すべてのデータ・エントリーには親のエントリーが存在するということです。図16を見てわかるように、ApacheDSのルート・エントリーはsystemです。

すべてのデータ・エントリーには、ディレクトリー内で一意なDNが設定されています。RFC 2253 (「参考文献」を参照) には、文字列形式でDNを指定する構文が用意されています。RFC 2253では、DNはコンマで区切られたコンポーネントのリストとして指定され、各コンポーネントにはそれぞれの値が設定されます。いくつかのコンポーネントを使用してDNを定義することができますが、この記事ではou、uid、cnの3つのみを使用します。各コンポーネントには異なる目的があります。

- ouコンポーネントは、組織単位の名前を指定します。この記事では、データ組織の単位の名前と考えることができます。この属性を使用して、ApacheDSの異なるタイプのデータを構造化することができます。
- uidコンポーネントは、エントリーのユーザー識別子を指定します。通常、この属性を使用してApacheDSのユーザー (Aliceなど) を識別します。
- cnコンポーネントは、Javaオブジェクト名を指定します。

図16に示したsystemルート・エントリーのDNは、ou=systemです。「users」エントリーはsystemの直接の子エントリーであるため、usersのDNはou=users,ou=systemになります。この最初の部分のDN (ou=users) は、「相対識別名」(RDN)と呼ばれることに注意してください。

RDNエントリーを作成する

子エントリーのRDN (ou=usersなど) の後にコンマが置かれ、その後に親のDN (ou=systemなど) が続きます。これらの一連の値が、子エントリーのDNになります (ou=users,ou=systemなど)。1つの親エントリーに直接つながる2つの子エントリー (つまり兄弟エントリー) に同じRDNを設定することはできません。これにより、LDAPディレクトリー内におけるDNの一意性が確保されます。

これで、図16に示した「Parent DN」フィールドと「Enter RDN」フィールドの意味は簡単に理解できます。新しいエントリーの作成中に「Users」エントリーを右クリックしたため、「Parent DN」フィールドには「Users」エントリーのDNがすでに入力された状態で表示されています。

次に、作成するエントリーのRDNを指定します。このサンプル・アプリケーションではAliceという新しいユーザーを作成するので、図16の「Enter RDN」フィールドにuid=Aliceと入力します。uid=alice,ou=users,ou=systemがAliceのエントリーのDNになります。

識別名に対するデータ照会または検索操作 (この記事の第2部で示します) では、大文字小文字が区別されないことに注意してください。つまり、RDNフィールドの値としてuid=Aliceまたはuid=aliceのどちらを指定しても実際上の違いはありません。

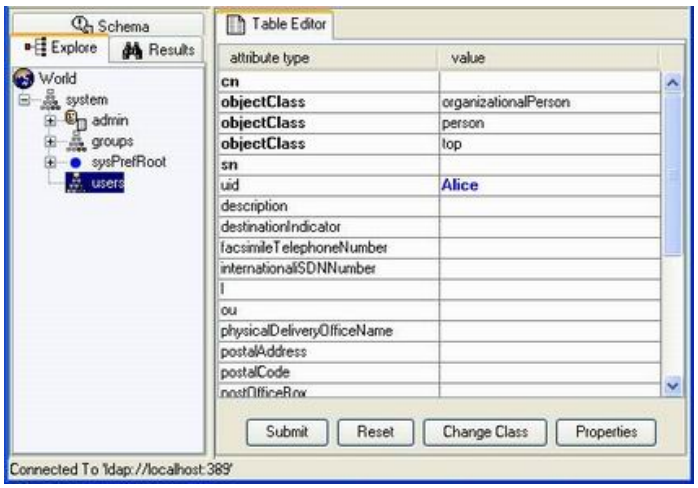
オブジェクト・クラスを使用する

ここまでは問題ないと思います。次に、新しいエントリーに指定するオブジェクト・クラスを選択します。エントリーには、任意の数のオブジェクト・クラスを選択することができます。オブジェクト・クラスを選択した場合、そのオブジェクト・クラスに含まれる必須属性をすべて指定する必要があります。さらに、オブジェクト・クラスのオプション属性も任意に指定することができます。

Aliceはデータ管理システムの作成対象である会社の営業部門に所属しているため、図11で紹介したorganizationalPersonオブジェクト・クラスに属する属性しか必要ありません。したがって、図16に示した「Selected Classes」リストにorganizationalPersonオブジェクト・クラスを追加します。

図17の画面は、「Set Entry Object Classes」ダイアログ・ボックスを設定した直後のAliceのエントリーを示しています。図17を見てわかるように、Aliceのuid値 (作成したばかりの値) は青色で表示されています。

図17. 「Set Entry Object Classes」ダイアログ・ボックス設定直後のAliceのエントリー

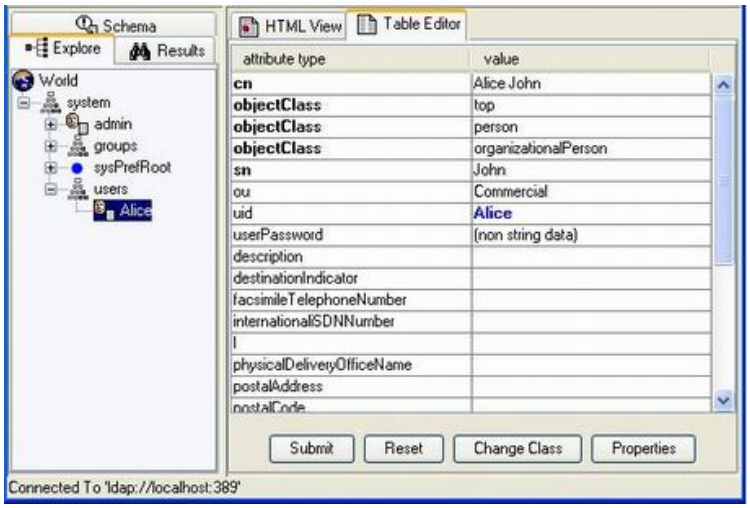


属性値を適用する

次に、必須属性 (図15に太字で示したcnとsn) の値を指定し、「Submit」をクリックする必要があります。JXplorerによって新しいエントリーを作成するLDAP要求が生成され、ApacheDSに要求が送信されます。

ApacheDSは確認応答メッセージをJXplorerに返し、JXplorerは図16に示したようにデータ・ツリーを更新します。図18の右側に、作成されたエントリーAliceが表示されています。

図18. Aliceのエントリー



これで作業は終了です。ApacheDSを使用した最初のデータ・エントリーが完成しました。

Java直列化とRMI

ディレクトリーにJavaオブジェクトを保管する場合は、常に最初にJavaオブジェクトをバイト表現に変換する必要があります。この変換プロセスを「直列化」と呼びます。直列化処理によってバイト・ストリームが出力されます。バイト・ストリームはネットワーク経由でApacheDSに送信することができます。送信されたバイト・ストリームは、Javaオブジェクトのバイト表現として保管することができます。

RMI (Remote Method Invocation) では、特殊な方法で直列化プロセスが使用されます。ApacheDSにJavaオブジェクトを保管する詳細を説明する前に、基本的な考え方について説明します。

Javaオブジェクトを直列化する

Java言語には、『Java Object Serialization Specification version 1.5』という仕様書があります。そこでは、Javaオブジェクトをバイト・ストリームに変換するプロセスが定義されています(「参考文献」を参照)。『Java Object Serialization Specification』によると、Javaオブジェクトを直列化するプロセスは、Javaオブジェクトの直列化と非直列化を処理する直列化ランタイムの形で、すでにJREに実装されています。

カスタム直列化

アプリケーション・アーキテクチャーの制限のためにデータ・メンバーを直列化できない場合 (たとえば、データ・メンバーの1つが既存の直列化不能クラスに属している場合)、直列化するJavaクラスに、独自にカスタマイズした直列化と非直列化のロジックを実装する必要があります。カスタム直列化については、この記事では説明しません。カスタム直列化の詳細については、「参考文献」を参照してください。

直列化ランタイムには、writeObject() とreadObject() という2つのメソッドが含まれています。このメソッドを使用し、『Java Object Serialization Specification』に従ってJavaの直列化と非直列化を行います。直列化するJavaオブジェクトでは、java.io.Serializableというインターフェースのみが実装対象となります。Serializableインターフェースにはメソッドはありません。Serializableインターフェースの役割は、オブジェクトが直列化可能であることを直列化ランタイムに知らせることだけです。

直列化ランタイムは、直列化の処理中にJavaオブジェクトのデータ・メンバーを直列化することに注意してください。データ・メンバー自体も別のJavaオブジェクトである場合、直列化するメイン・オブジェクトと共にそのメンバーも直列化する必要があります。したがって、そのようなメンバーもSerializableインターフェースを実装する必要があります。

第2回目の記事では、Javaのデフォルトの直列化サポートを使用してApacheDSにJavaオブジェクトを保管する方法を説明します。

RMI

オブジェクトを直列化してネットワーク経由で送信する前に、バイト表現に特定のエンコードの適用が必要になる場合があります。このエンコード・プロセスは「マーシャル」と呼ばれ、Java RMI仕様の重要な要素の1つです。RMIフレームワークを使用すると、Javaアプリケーションでリモート・オブジェクトを使用することが可能になります。リモート・オブジェクトは、Javaアプリケーションの範囲外 (インターネット上など) に存在するアプリケーションです。

RMIを使用すると、ローカル・オブジェクトのメソッドを呼び出す場合と同じようにリモート・オブジェクトのメソッドを呼び出すことができます。この場合、リモート・オブジェクトのスタブ・クラスが必要です。スタブ・クラスには、リモート・オブジェクトのメソッド・シグニチャーが含まれています。アプリケーションはスタブ・クラスをインスタンス化し、メソッドをローカルで呼び出します。その後、RMIフレームワークによってリモート・オブジェクトとの通信が管理され、要求されたメソッドのリモート呼び出しが行われます。そのためアプリケーションは、オブジェクトが実際にローカルかリモートかを識別する必要がなくなります。

RMIフレームワークの詳細については、この記事では説明しません。「参考文献」で紹介している資料を参照してください。ここで重要なことは、RMIフレームワークを使用することにより、Javaオブジェクトをネットワーク経由で送信し、リモート・オブジェクトのメソッドの呼び出しが可能になるということです。したがって、RMIでJavaオブジェクトを直列化する必要があります。RMI仕様ではマーシャル・アルゴリズムが指定されています。このアルゴリズムは直列化された形のJavaオブジェクトに適用され、直列化されたオブジェクトに特殊フラグを挿入します。

RMI仕様はすでにJ2SEに実装されているため、オブジェクト・マーシャルの下層レベルの詳細について心配する必要はありません。RMIクラスを使用するだけで、Javaオブジェクトをマーシャルすることができます。LDAPを使用すると、LDAPディレクトリーにマーシャルされたJavaオブジェクトの保管が可能になります。次回の記事では、Javaオブジェクトのマーシャルとアンマーシャルの方法、およびApacheDSへの保管とApacheDSからの取得の方法について説明します。

ApacheDSにJavaオブジェクトを保管する

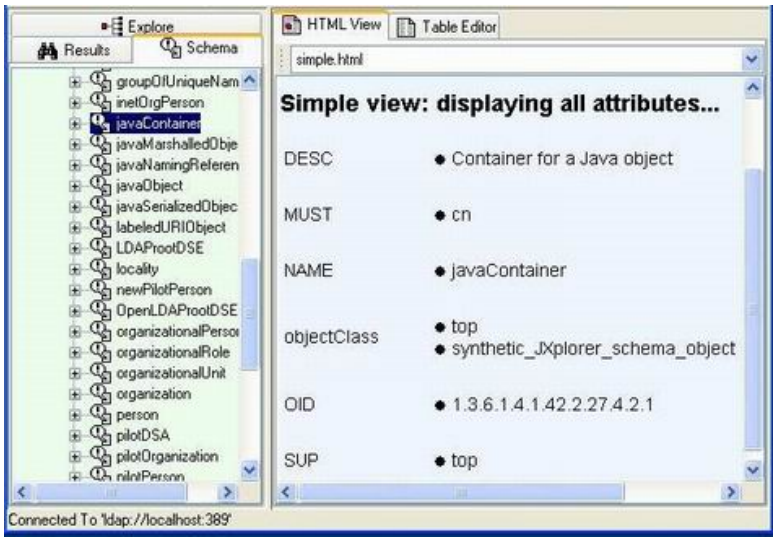
ここからは、これまでに見てきた内容を示す簡単なアプリケーションを説明します。直列化されたJavaオブジェクトを表す方法、さまざまなオブジェクト・クラスを使用する方法、マーシャルされたJavaオブジェクトを表す方法、ApacheDSにJavaオブジェクトの参照を保管する方法について、それぞれ説明します。次回の記事ではさらに詳しく見ていきますが、ここでは基本手順を中心に説明します。

最初に、使用するオブジェクト・クラスの集合が必要です。RFC 2713 (「参考文献」を参照) では、LDAPディレクトリーのJavaオブジェクトを表すオブジェクト・クラスと属性タ

イプが定義されています。ApacheDSは、これらの属性タイプとオブジェクト・クラスを完全にサポートしています。この演習と次回の記事の演習では、4つのオブジェクト・クラス (javaContainer、javaObject、javaSerializedObject、javaNamingReference) を使用します。最初の2つのオブジェクト・クラスとその属性を見てみましょう。

javaContainer (図19に表示) は、1つの属性 (cn) のみが設定された構造クラスです。このオブジェクト・クラスの目的は、Javaオブジェクトを含むデータ・エントリーの名前を指定することです。LDAPでは、ほかの構造オブジェクト・クラスを使用してJavaオブジェクト名を付けることができます。たとえば、personオブジェクト・クラスを使用してJavaオブジェクトを保管することもできます。

図19. javaContainerクラス



javaObjectは抽象クラスです。このオブジェクト・クラスの目的は、Javaオブジェクトのデータ保管には直接使用されず、検索操作でディレクトリーのオブジェクトを検索して取得する場合に役立つ属性を定義することです。図20にjavaObjectクラスを示します。

図20. javaObjectクラス

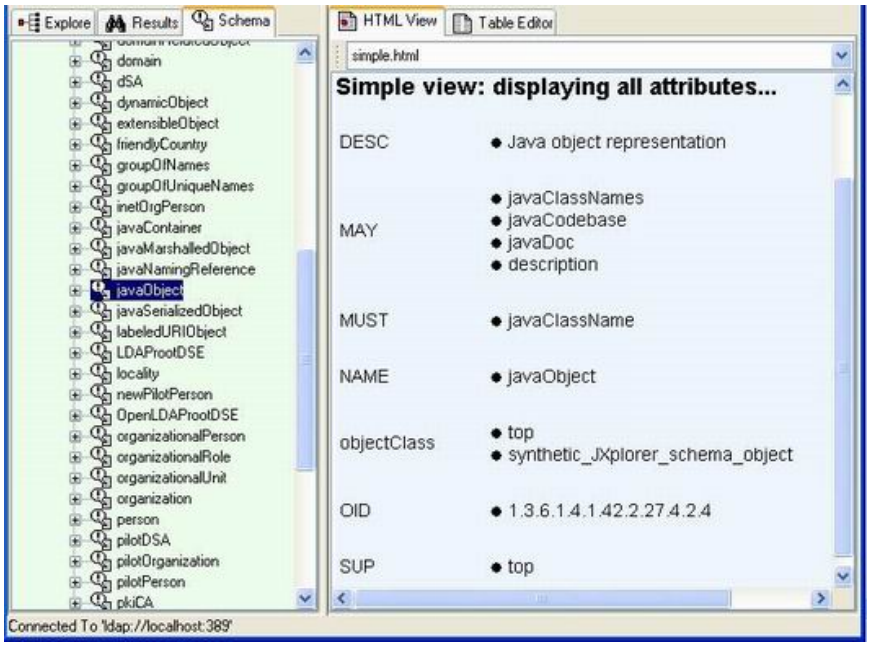


図20に示したjavaObjectクラスの属性タイプは以下のとおりです。

- 必須属性タイプjavaClassNameには、インスタンスがApacheDSに保管されているクラスの名前が格納されます。通常、ApacheDSに保管された特定のクラスのインスタンスを検索する場合にこの属性タイプを使用します。
- もう1つの属性タイプjavaClassNamesには、Javaオブジェクトのすべてのスーパー・クラスの名前と、オブジェクトが実装するすべてのインターフェースの名前が格納されます。この属性は複数の値を持つことを前提としているため、多値属性タイプになっています。LDAPでは、多値属性は複数の値を保持することができます。特定のインターフェースを実装したオブジェクトまたは特定のクラスを拡張しているオブジェクトを検索する場合、javaClassNames属性タイプを使用します。
- 図20に示したjavaCodebase属性タイプには、保管されたJavaオブジェクトのクラス定義の場所が格納されます。これはオプション属性であり、ApacheDSから場所を読み込んで、クラス定義のロードが必要な場合に使用します。
- javaDoc属性タイプもオプションです。この属性タイプには、インスタンスがApacheDSに保管されたクラスのJavaドキュメントを指すURLが格納されます。

これまでの説明からわかるとおり、description属性タイプにはクラスの説明が格納されます。

直列化されたJavaオブジェクトを表す

javaObjectの属性タイプには、保管されるオブジェクトを表すオクテットの実際の文字列は格納されないことに注意してください。そのため、javaObjectクラスは抽象クラスのままで、javaObjectクラスを使用してデータ・エントリーを直接作成することはできません。LDAPでこのようにjavaObjectクラスが定義されているのは、Javaオブジェクトをさまざまな形式 (たとえば、直列化またはマーシャルされた形式) で保管できるようにするためです。オブジェクト表現の各形式には、javaObjectクラスを拡張する独自のオブジェクト・クラスがあります。

javaSerializedObjectというオブジェクト・クラス (図21に表示) は、javaSerializedDataという属性を1つだけ追加してjavaObjectクラスを拡張しています。javaSerializedData属性には、実際に直列化された形式のJavaオブジェクトが入ります。

図21に示すjavaSerializedObjectクラスは、補助オブジェクト・クラスです。理由については、この後で説明します。

図21. javaSerializedObjectクラス

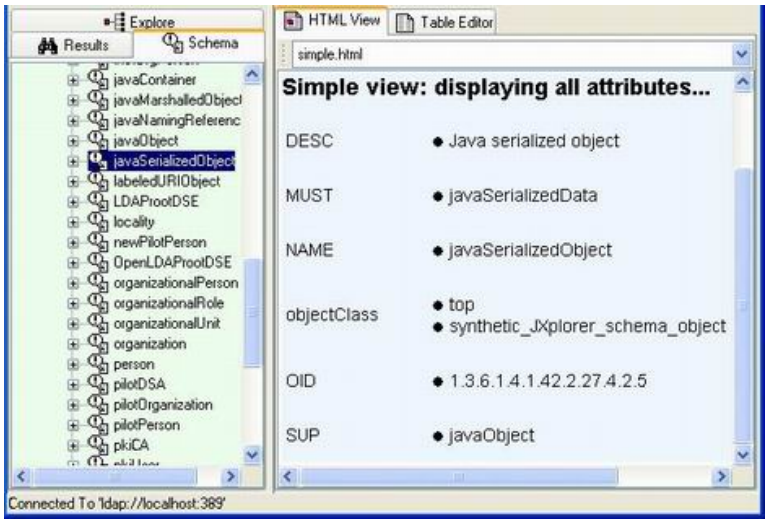


図22には、ApacheDSに保管されたMessagingPreferencesというJavaの直列化オブジェクトがJXplorer画面に表示されています。

図22. ApacheDSに保管されたAliceのMessagingPreferencesオブジェクト

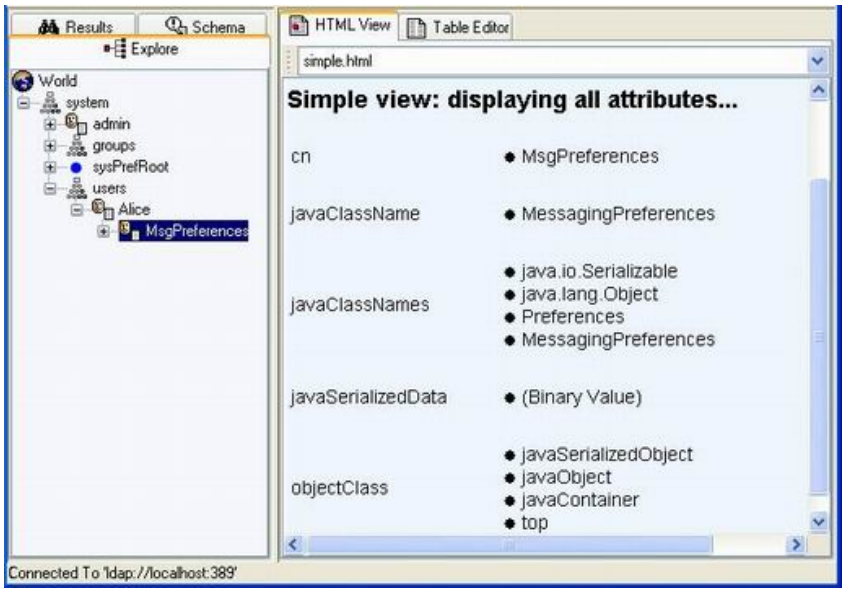


図22のMessagingPreferencesオブジェクトは、Aliceが使用する設定を表しています。そのため、MessagingPreferencesオブジェクトは、前に作成したAliceのデータ・エントリー・オブジェクトの子オブジェクトとして表示されています。

異なるオブジェクト・クラスを使用する

図22に示したように、MessagingPreferencesデータ・エントリー・オブジェクトでは4つのオブジェクト・クラスが使用されます。javaSerializedObject、javaObject、javaContainer、topです。なぜ、MessagingPreferencesオブジェクトにこれら4つのクラスが必要になるのでしょうか。

MessagingPreferencesデータ・エントリー・オブジェクトにjavaSerializedObjectクラスが必要な理由は、直列化された形のMessagingPreferencesオブジェクトを保管するためです。

javaObjectクラスが必要な理由は、javaSerializedObjectクラスがjavaObjectクラスを拡張するためです。つまり、データ・エントリー・オブジェクトがjavaSerializedObjectクラスを使用する場合、javaObjectクラスも常に使用するということになります。

javaContainerクラスが必要な理由は、ほかの3つのオブジェクト・クラス (javaSerializedObject、javaObject、top) は構造クラスではないためです。ApacheDSのすべてのデータ・エントリー・オブジェクトでは、構造クラスを使用する必要があります。Javaオブジェクトでは、javaContainerや、personオブジェクト・クラスなどのほかの構造クラスを使用することができます。Javaオブジェクト・エントリーに構造クラス (javaContainerやpersonオブジェクト・クラスなど) を組み合わせるため、LDAP仕様ではjavaSerializedObjectクラスが補助クラスとして定義されています。

最後に、MessagingPreferencesデータ・エントリー・オブジェクトがtopクラスを必要とする理由は、すでに説明したように、すべてのオブジェクト・クラスがtopクラスから拡張されるためです。

マーシャルされたJavaオブジェクトを表す

図23に、ApacheDSに保管されたマーシャル済みJavaオブジェクトを示します。

図23. ApacheDSに保管されたマーシャル済みオブジェクト

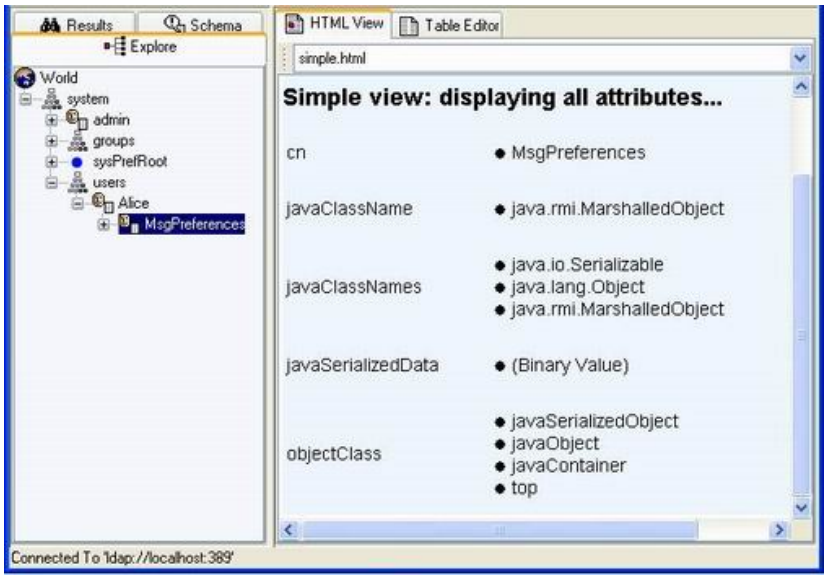


図23を図22 (JXplorer画面の直列化されたオブジェクトを示しています) と比較してください。図23のjavaClassName属性は、エントリーがjava.rmi.MarshalledObjectクラスのインスタンスを含んでいることを示しています。次回の記事では、このクラスを使用してJavaオブジェクトのマーシャルについて説明します。

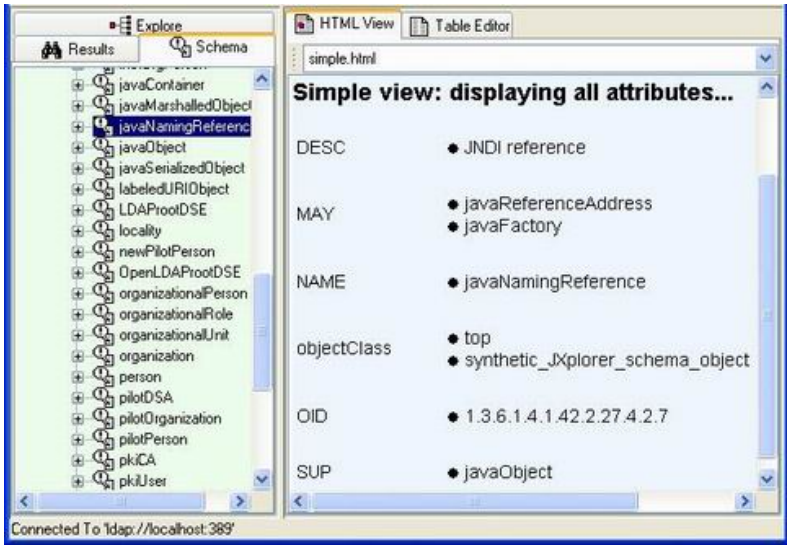
Javaオブジェクトの参照を保管する

LDAPでは、ディレクトリーに実際のオブジェクトを保管する代わりに、Javaオブジェクトの「参照」を保管することもできます。この場合、ApacheDSにJavaオブジェクトのアドレスを保管します。アドレスを逆参照してオブジェクトのインスタンスを取得する場合、ファクトリー・オブジェクトが必要になります。ファクトリー・オブジェクトには、逆参照ロジックが実装されます。

参照を保管する主な利点は、異なるユーザー間でオブジェクトを共有できることです。たとえば、リスト1に関する説明で紹介したMessagingPreferencesクラスを考えてみましょう。複数のユーザーが共通のメッセージング設定を共有する場合、共通設定を表す1つのMessagingPreferencesインスタンスを保管し、設定を使用する各ユーザーの参照を保管すると便利です。

RFC 2713には、javaObjectクラスを拡張し、javaReferenceAddressとjavaFactoryという2つの新しい属性タイプを定義するjavaNamingReferenceというオブジェクト・クラスがあります (図24を参照)。javaReferenceAddress属性タイプにはJavaオブジェクトの参照 (またはアドレス) が格納され、javaFactory属性タイプにはファクトリー・オブジェクトの名前が格納されます。

図24. javaNamingReferenceクラス



まとめ

ApacheDSへのJavaオブジェクトの保管について説明したこの第1回目の記事では、ApacheDSの組み込み可能なプロトコルのサポートに関するさまざまなことを見てきました。また、識別名、オブジェクト・クラス、属性タイプ、LDAP構文など、LDAPの基本的な考え方と用語についても見て

きました。最後に、こうしたLDAPの概念を使用してJavaオブジェクトを表し、ApacheDSにJavaオブジェクトを保管する方法についても見てきました。

この記事の第2回目では、こうした概念をテストします。いくつかのサンプルJavaアプリケーションとさまざまなJavaコードを使用し、ApacheDSへのJavaオブジェクトの保管、検索、および取得を行う方法を説明します。サンプル・アプリケーションでは、ここで紹介したデータ管理シナリオのさまざまな機能を実装します。第2回目の記事の最後では、こうした概念を独自のアプリケーションで使用できる再使用可能なJavaクラスの形にして紹介します。次回の記事に続きます。

ダウンロード

内容	ファイル名	サイズ
Source code	j-apachedssource.zip	40KB

著者について

Bilal Siddiqui

Bilal Siddiqui は、電気工学エンジニア、XML コンサルタント、そして e-business の簡易化を専門とする会社、WaxSys の共同設立者でもあります。1995年に電子工学技術の学位で University of Engineering and Technology, Lahore を卒業した後、産業用制御システムを対象としたソフトウェア・ソリューションの設計を始めました。その後、XML に転向し、C++ のプログラミング経験を生かして Web ベースや Wap ベースの XML 処理ツール、サーバー側の構文解析ソリューション、そしてサービス・アプリケーションを作成しました。技術の熱烈な支持者である彼が書いた技術書は、頻繁に発行されています。