# Apache Directory ServerへのJavaオブジェクトの保管 第2回

## ApacheDS内でのJavaオブジェクトの保管、検索、取得

Bilal Siddiqui
Freelance consultant
WaxSys

2006年 5月 02日

ApacheDS (Apache Directory Server) へのJava(TM) オブジェクトの保管手順を説明するこの記事では、第1回目の記事で説明した概念を、Bilal Siddiquiが9つのアプリケーションを例に取って解説していきます。また、ApacheDSを使用したJavaオブジェクトの保管、検索、取得、変更の各手順をすべて紹介した上で、LDAPスキーマ・コンポーネントを使用してこれらの機能をApacheDS内で組み合わせる再使用可能なJavaクラスを最後に説明します。

第1回目の記事では、ApacheDSへのJavaオブジェクトの保管に関する基本概念を紹介しました。ApacheDSのコア・アーキテクチャーについて説明し、そのディレクトリー・サービスへの実装方法と組み込み可能なプロトコルのサポート方法について取り上げました。また、LDAPの概念と用語も紹介しました。さらに、ApacheDSによるLDAPプロトコル実装の仕組みを説明し、ApacheDS内でのオブジェクトの保管や操作に使用するさまざまなコンポーネントを紹介しました。記事の最後では、JavaオブジェクトをApacheDSへ保管する場合とApacheDSから取得する場合に理解が必要なJavaオブジェクトの直列化とRMIについても触れました。また、アプリケーションの例として製造会社向けのデータ管理システムを紹介し、このシステムを例に取ってこうした概念を説明しました。

第2回目となるこの記事では、全部で9つになる例の紹介を中心に進めていきます。これらの例は、前回の記事で取り上げたデータ管理システムをベースにしています。この例を使用して、Apache DS内でのJavaオブジェクトの保管、検索、取得、更新の各方法を説明していきます。

ApacheDSとJXplorerをまだダウンロードおよびインストールしていない場合は、この記事を読む前にダウンロードしてインストールしてください。この記事で紹介している完全なソースは、いつでもダウンロードできます。

### この記事を読む前に

この記事で説明する例を理解するには、「識別名」(DN)、「相対識別名」(RDN)、「名前付きコンテキスト」、「オブジェクト・クラス」、および「属性タイプ」といった、基本的なLDAPの用語および概念に関する知識が必要です。これらの用語を理解していない場合は、この記事の前に前回の記事を読んでください。

# アプリケーション1. Javaオブジェクトを保管する

初めに、いくつかのアプリケーションを紹介して、オブジェクトのApacheDSへの保管方法を説明します。この場合、ディレクトリー内でオブジェクトと属性を連動させるためのインターフェースとメソッドを提供するJNDI (Java Naming and Directory Interface) を使用する必要があります。ApacheDSがJNDIインターフェースを使用してディレクトリー・サービスを公開する仕組みについては、前回の記事 (「Apache Directory ServerへのJavaオブジェクトの保管 第1回」) 参照してください。

JNDIは、LDAP固有のインターフェースではありません。JNDI実装は、どのような種類のディレクトリー・サービスにも使用できます。たとえば、ディレクトリー・サービスを実装し、JNDIを使用してその機能を公開するには、そのディレクトリー・サービスにJNDIインターフェースを実装します。また、ApacheDSとのやり取りに使用するJ2SE (Java 2 Standard Edition) は、クライアント側JNDI実装に含まれることに注意してください。このクライアント側実装については、この記事全体を通じて使用します。

リスト1は、StoreAlicePreferencesという名前の簡単なアプリケーションです。このアプリケーションを使用して、ユーザーAliceの設定をJavaオブジェクトとしてApacheDSに保管する方法を説明します。

## リスト1. StoreAlicePreferences

```
public class StoreAlicePreferences {

    public StoreAlicePreferences ()
    {
        try {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            //Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-----------------------------------------
            //Step3: Instantiate a Java object
            //-----------------------------------------
            MessagingPreferences preferences = new MessagingPreferences();

            //-----------------------------------------
            //Step4: Store the Java object in ApacheDS
            //-----------------------------------------
            String bindContext = "cn=preferences,uid=alice,ou=users";
            ctx.bind( bindContext, preferences);
        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
       StoreAlicePreferences storeAlicePref = new StoreAlicePreferences();
    }
```

```
}
```

リスト1に書かれているコメントから分かるように、Javaオブジェクト (ここではAliceの設定) をApacheDSへ保管するには4つの手順が必要になります。ここからは、それぞれの手順について詳しく見ていきます。

## Step 1. ApacheDS用にJNDIプロパティーを設定する

リスト1の最初の手順は、ApacheDSプロパティー・ファイルをPropertiesオブジェクトに読み込む作業です。そのためには、リスト2で示すように、まずユーザー自身のJNDIプロパティーを異なるプロパティー・ファイルに書き出す必要があります。

### リスト2. ApacheDSプロパティー・ファイル

```
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url=ldap://localhost:389/ou=system
java.naming.security.authentication=simple
java.naming.security.principal=uid=admin,ou=system
```

クライアント側JNDI実装を使用する各アプリケーションはどのJNDI実装からも独立して動作するという前提になっているため、個別のプロパティー・ファイルが必要になります。指定されたJavaアプリケーション (StoreAlicePreferencesなど) は、ApacheDSやその他のいずれのディレクトリー・サービスとも連動可能である必要があります。ディレクトリー・サービスは、LDAPを使用しません。

プロパティー・ファイルの値の意味は、簡単なシナリオを考えるとよく理解できます。たとえば、クライアント側JNDI実装を使用してLDAPがApacheDSとやり取りできるようなJavaアプリケーションを開発したとします。その後、LDAP以外のプロトコルを使用する別のディレクトリー・サーバーを購入したとします。

この場合、新しいディレクトリー・サーバーと連動可能なクライアント側JNDI実装が新たに必要となります。こうすることにより、このJavaアプリケーションは、新たにコードを記述することなく正常に機能します。この場合に必要な作業は、プロパティー・ファイルを更新して新しいディレクトリー・サーバーのプロパティーを反映させることだけです。

アプリケーション内でプロパティーをハード・コーディングすることは、プログラミング上は問題ありませんが、このアプリケーションはハード・コーディングした特定の実装に依存することになります。この場合、特定の実装に依存しないJNDIを使用するという本来の目的から外れてしまうことになります。

### プロパティー・ファイルの詳細

リスト2で示したApacheDSプロパティー・ファイルを見てみましょう。プロパティー・ファイルは、多数の名前と値のペアで構成されています。それぞれの名前はプロパティーを表しています。

これらのプロパティーは、Contextという名前のJNDIインターフェースを公開するオブジェクトをインストールする際に使用されます。Contextは、JNDIでもっとも重要なインターフェースです。

このインターフェースは、名前付きコンテキストと連動するためのメソッドを定義します (名前付きコンテキストの概念は、前回の記事で紹介したアプリケーションの例で説明しました)。

たとえば、Contextインターフェースで定義された重要なメソッドの1つは、Javaオブジェクトを名前付きコンテキストにバインドするbind() メソッドです。オブジェクトを名前付きコンテキストにバインドするということは、特定のコンテキストのディレクトリー内に特定の名前でオブジェクトを保管するということです。bind()メソッドの使用方法については、この後すぐに説明します。まずは、プロパティー・ファイルの名前と値のペアについて見ていきましょう。

## 名前と値のペア

リスト2で1番目に表示されている名前と値のペア java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactoryは、java.naming.factory.initialというJNDIプロパティーを設定します。java.naming.factory.initialプロパティーは、JNDIクライアント側実装の一部として使用するオブジェクト・ファクトリーの名前を指定します。オブジェクト・ファクトリーは、Contextオブジェクトを作成します。このオブジェクトは、ApacheDS内で名前付きコンテキストと連動させる際に使用します。

ここではJNDIのLDAPベース実装を使用するため、このプロパティーの値として com.sun.jndi.ldap.LdapCtxFactoryを指定します。com.sun.jndi.ldap.LdapCtxFactoryクラスは、LDAPプロトコルに従ってApacheDSと通信できるようにContextオブジェクトを構築します。

リスト2で2番目に表示されている名前と値のペアは、java.naming.provider.url=ldap://localhost:389/ou=systemです。java.naming.provider.urlプロパティー・ファイルは、作業対象となる完全なディレクトリー・コンテキストのURLを指定します。

完全なディレクトリー・コンテキストは、ApacheDSがlistenしているURLと、ApacheDS内にある作業対象の名前付きコンテキストの2つのコンポーネントで構成されています。文字列「ldap://localhost:389/」は、ApacheDSがlistenしているURLを指定します。この文字列の残りの部分 (ou=system) は、作業対象の名前付きコンテキストを指定します。

3番目に表示されている名前と値のペアは、java.naming.security.authentication=simpleです。このプロパティーは、ApacheDSがユーザー認証に使用するセキュリティーの強度を指定します。このプロパティーには、none、 simple、strongという3つの値のいずれか1つを設定します。

- 「none」を選択した場合、ApacheDSは認証を使用しません。この場合、パスワードを指定することなく誰でもログインできます。
- 「simple」を選択した場合、ApacheDSは簡単なパスワード・ベースの認証を使用します。この場合、パスワードはプレーン・テキスト形式 (非暗号化テキスト形式) でネットワーク上に送信されます。
- 「strong」を選択した場合、(プレーン・テキスト形式の実パスワードの代わりに) ハッシュされたユーザーのパスワードの値が認証用にApacheDSへ送信されます。

現行バージョンのApacheDSでは、「strong」レベルの認証はサポートされていません。

リスト2で4番目に表示されている名前と値のペア は、java.naming.security.principal=uid=admin,ou=systemです。このプロパティーは、ApacheDSに

ログオンしようとしているユーザーのDNを指定します (ここでは、このプロパティーの値としてApacheDS管理者 (uid=admin,ou=system) のDNを使用しています)。

リスト2のApacheDSプロパティー・ファイルでは、4つの名前と値のペアを見てきました。ここで、このプロパティー・ファイルをPropertiesオブジェクトに読み込んだリスト1の「Step 1」をもう一度読み直してください。JNDIを操作する際に、このPropertiesオブジェクトを使用することになります。

### ユーザー・パスワードを設定する

Propertiesオブジェクトには、ユーザーのパスワードも含める必要があります。実際のアプリケーションでは、通常はユーザーのパスワードを設定ファイルに保管することはありません。その代わりに、ユーザーからGUIを通じて受け取ります。リスト1では、パスワードをjava.naming.security.credentialsという名前のプロパティー値に設定してあります。このプロパティーが、実際にユーザーのIDを証明するための証明書となります。証明書には、パスワードやKerberosチケットなどのいくつかのタイプがありますが、この記事の便宜上、ここではパスワード・ベースの認証を使用します。

すべてのプロパティーは設定済みで、Propertiesオブジェクトは使用可能な状態になっています。

## Step 2. DirContextオブジェクトを取得する

次に、InitialDirContextという名前のクラスをインスタンス化します。このクラスはJNDIの一部で、DirContextという名前のインターフェースを公開します。InitialDirContextコンストラクターは、上述したPropertiesオブジェクトを取得します。

InitialDirContextオブジェクトは、新規オブジェクトの保管、保管済みオブジェクトの検索、既存のオブジェクトへの属性の追加など、ApacheDS上で実行するすべてのディレクトリー操作を実行することができます。

### DirContextインターフェース

DirContextインターフェースは、Contextインターフェースを拡張します。Contextインターフェースは名前付きコンテキストを表し、DirContextインターフェースは、名前付きコンテキストに関連付けられた属性の追加、編集、管理に関する機能を提供します。

つまり、Contextインターフェースは命名機能を提供し、DirContextインターフェースは属性に対してサポートを追加することによりその命名機能を拡張します。命名機能および属性機能は、連動してディレクトリー・サービスを構築します。

InitialDirContextオブジェクトは、ファクトリー・オブジェクトによりインスタンス化されたDirContextオブジェクトのラッパーと考えることもできます。ここで示す例の場合、InitialDirContextコンストラクターはリスト2の最初のプロパティー (com.sun.jndi.ldap.LdapCtxFactory)で指定されたコンテキスト・ファクトリー・オブジェクトを使用します。このファクトリー・オブジェクトは、DirContextオブジェクトを公開するオブジェクトをインスタンス化します。また、InitialDirContextオブジェクトはこのDirContextオブジェクトを使用して、クライアント・アプリケーションの要求に応じてディレクトリー操作を実行します。

### ApacheDSを使用する利点

ApacheDSディレクトリー・サービスのもっとも大きな利点は、定義上、クライアント・アプリケーションを特定の実装から独立させることにあります。クライアント・アプリケーションは設定ファイルのファクトリー・オブジェクトを指定し、InitialDirContextオブジェクトはそのファクトリー・オブジェクトを使用してDirContextオブジェクトをインスタンス化します。このオブジェクトには、リモート・ディレクトリー・サービスとの通信を処理するためのすべてのロジックが記述されています。

たとえばリスト1では、Sun Microsystemsのファクトリー・オブジェクトcom.sun.jndi.ldap.LdapCtxFactoryを使用しています。このファクトリー・オブジェクトは、ApacheDSが理解できるLDAP要求のオーサリングが可能なDirContextオブジェクトを作成します。

リスト1からStoreAlicePreferencesアプリケーションを複数の非LDAPディレクトリー・サービスとともに実行する場合は、非LDAPサービスのビジネス・ロジックに従って、リスト2のファクトリー・オブジェクト名を新規ファクトリー・オブジェクトと交換するだけで実行できます。こうすると、StoreAlicePreferencesアプリケーションは、この非LDAPサービスとの連動を開始します。

## Step 3. Javaオブジェクトをインスタンス化する

次に、リスト3に示した、Aliceのメッセージング設定を表すクラスMessagingPreferencesをインスタンス化します (前回の記事で説明したメッセージング設定を思い出してください)。

### リスト3. MessagingPreferencesクラス

```
public class MessagingPreferences extends
    Preferences implements java.io.Serializable {
    static final long serialVersionUID = -1240113639782150930L;

    //Methods of the MessagingPreferences class
}
```

この場合、MessagingPreferencesクラスのメソッドを呼び出してAliceの設定を変更することもできます。

リスト3で、MessagingPreferencesクラスは、Serializableインターフェース (前回の記事の「Javaオブジェクトを直列化する」セクションで説明しました) と、この後すぐに説明するserialVersionUIDを実装します。

### serialVersionUIDを取得する

すべての直列化可能クラスには、long型のプライベートな静的データ・メンバーserialVersionUIDを持たせるようにしてください。このデータ・メンバーを直列化可能クラス内で使用する必要はありません。Javaランタイムが、直列化および非直列化を実行する際にこのデータ・メンバーを使用します。

Javaオブジェクトの直列化仕様 (「参考文献」を参照) により、serialVersionUIDの値を計算するための複雑なアルゴリズムが指定されます。このアルゴリズムでは、直列化可能クラスの名前、

それにより実装されるすべてのインターフェースの名前、直列化可能クラスのすべてのデータ・メンバーなどが使用されます。この複雑なアルゴリズムの詳細について心配する必要はありません。Javaプラットフォームには、この値を計算するためのツールserialverが用意されています。

コマンド・ラインから以下のようにserialverツールを呼び出して使用すると、MessagingPreferencesオブジェクトのserialVersionUIDを確立することができます。

```
X:\jdk1.5\bin\serialver MessagingPreferences
```

リスト3を見て分かるように、ここではMessagingPreferencesクラスに対してこの作業をすでに行っています。

## Step 4. ApacheDSへJavaオブジェクトを保管する

これまでの作業で、DirContextオブジェクトとMessagingPreferencesオブジェクトの設定が終了しました。ここからは、DirContextオブジェクトを使用してApacheDSにMessagingPreferencesオブジェクトを保管する手順について見ていきます。

LDAPサーバーにデータ・エントリーを保管することを、バインド操作と呼びます。Contextインターフェースには、ApacheDSへJavaオブジェクトを保管する際に使用するbind() というメソッドが用意されています。bind() メソッドは、リスト1の「Step4」に記述されています。

### Context.bind() のパラメーターを設定する

Context.bind()メソッドには、2つのパラメーターが設定されています。最初のパラメーター(cn=preferences,uid=alice,ou=users,ou=system) には、Javaオブジェクトの保管先となる名前付きコンテキストを指定します。この名前付きコンテキストをコンマで区切ることにより、cn=preferencesとuid=alice,ou=users,ou=systemという2つの部分に分けることができます。

この新しいエントリーはAliceの設定表示を表しているため、cn=preferencesをそのRDNとして使用します。文字列uid=alice,ou=users,ou=systemは、前回の記事の「RDNエントリーを作成する」のセクションで紹介したAliceのデータ・エントリーのDNと同じであることに注意してください。

これらすべてをまとめると、新規エントリーのDNは、bind() メソッドに渡す最初のパラメーターであるcn=preferences,uid=alice,ou=users,ou=systemとなります。

Context.bind()メソッド呼び出しの2番目のパラメーターは、「Step 3」のMessagingPreferencesオブジェクトです。bind() メソッド呼び出しからの戻り値はありません。

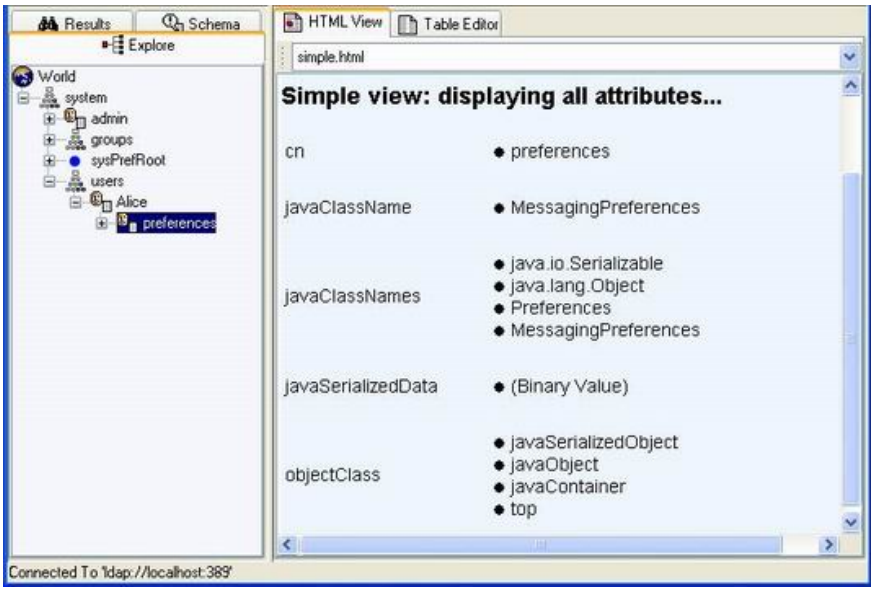
## 最初のアプリケーションを実行する

これまでに見てきた4つの手順をまとめたものが、リスト1で示したサンプルのアプリケーションStoreAlicePreferencesです。この記事のソース・コードからも、このサンプルのアプリケーションを見ることができます。

StoreAlicePreferencesアプリケーションを実行する前に、ApacheDSに保管されたuid=alice,ou=users,ou=systemと同じDNのエントリーが必要です。前回の記事の「RDNエントリーを作成する」セクションでは、Aliceという名前のユーザーを作成しました。

StoreAlicePreferencesアプリケーションの実行後は、LDAPブラウザー (この場合はJXplorer) 内のAliceエントリーを拡張して、Aliceのメッセージング設定がJavaオブジェクトとして保管されているかどうかを確認することができます。Aliceの拡張表示は、図1のようになります。

図1. Aliceのメッセージング設定が保管された状態




## アプリケーションの注意点

図1には、MessagingPreferencesオブジェクトのほかに3つの属性が含まれています。javaClassName、javaClassNames、javaSerializedDataというこれらの属性については、前回の記事の「ApacheDSにJavaオブジェクトを保管する」セクションで説明しました。

bind() メソッド呼び出しが設定されたこれらの属性が「Step 4」のStoreAlicePreferencesアプリケーションに含まれていないため、これらの属性がどのようにApacheDSへ保管されるのか不思議に思うかもしれません。実は、bind() メソッド自身がこれらの属性を書き出しているのです。2つのパラメーターを持つContext.bind() メソッドには属性がありません。ただし、前回の記事の「ApacheDSにJavaオブジェクトを保管する」セクションで説明したように、LDAPはjavaClassName属性、javaClassNames属性、javaSerializedData属性を必要とします。そのため、Context.bind() メソッド自身がこれらの属性を書き出します。

次のセクションでは、3つのパラメーターを持つbind() メソッドを説明します。このメソッドには属性の配列が設定され、これらの属性はJavaオブジェクトと共に保管されます。

図1に示したMessagingPreferencesオブジェクトは、javaContainerオブジェクト・クラスを使用します。このクラスについては、前回の記事の「ApacheDSにJavaオブジェクトを保管する」セクションで説明しました。必要であれば、次のアプリケーションの例で示すように、javaContainerクラスを使用せずにJavaオブジェクトをApacheDSに書き出すこともできます。

# アプリケーション2. 属性が設定されたJavaオブジェクトを保管する

この例では、上で触れた3つのパラメーターを持つbind() メソッドを使用して、Javaオブジェクトへ属性を追加する方法について説明します。リスト4に示したサンプルのアプリケーション

StoreBobPreferencesを見てみましょう。このアプリケーションは、Bobという名前のユーザーのエントリーを作成し、Bobの設定 (つまり属性) をそのエントリーに保管するという2つの手順を1度に実行します。

## リスト4. StoreBobPreferences

```
public class StoreBobPreferences{

    public StoreBobPreferences ()
    {
        try {

            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "apacheds.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------------
            //Step2: Fetching a DirContext object
            //-----------------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-----------------------------------------------
            //Step3A: Instantiate a Java Object
            //-----------------------------------------------
            MessagingPreferences preferences = new MessagingPreferences();

            //-----------------------------------------------
            //Step3B: Instantiate BasicAttribute object
            //-----------------------------------------------
            Attribute objclass = new BasicAttribute("objectClass");

            //-----------------------------------------------
            //Step3C: Supply value of attribute
            //-----------------------------------------------
            objclass.add("person");

            //-----------------------------------------------
            //Step3D: Put the attribute in attribute collection
            //-----------------------------------------------
            Attributes attrs = new BasicAttributes(true);
            attrs.put(objclass);

            //-----------------------------------------------
            //Step4: Store the Java object in ApacheDS
            //-----------------------------------------------
            String bindContext = "uid=Bob,ou=users";
            ctx.bind( bindContext, preferences, attrs);
        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
       StoreBobPreferences storeBobPref = new StoreBobPreferences();
    }
}
```

リスト4のほとんどの部分は、次に説明する「Step 3」の一部の追加コードを除いて、リスト1と同じ手順で構成されています (リスト4の「Step 3A」はリスト1の「Step3」と同じであるため、ここでは「Step 3B」から記述していることに注意してください)。

## Step 3B. BasicAttributeをインスタンス化する

StoreAlicePreferencesアプリケーションとStoreBobPreferencesアプリケーションを区別する最初の手順として、Attributeという名前のJNDIインターフェースを公開するJNDIクラスBasicAttributeをインスタンス化します。Attributeインターフェースは、LDAPデータ・エントリーの1つの属性を表すことができます。BasicAttributeクラスは、Attributeインターフェースの基本的な実装 (機能制限あり) を提供します。各種アプリケーションおよび実装には、(BasicAttributeクラスを拡張することにより) Attributeインターフェース自身の実装を持たせることができます。ただし、BasicAttributeクラスを拡張しなくても、この記事の説明には何の問題もありません。

BasicAttributeコンストラクターは、パラメーターとして属性の名前を持ちます。リスト4の「Step 3B」において、設定済みの最初のBasicAttributeオブジェクトは、パラメーターとしてobjectClassを持つことに注意してください。この場合、BasicAttributeオブジェクトがobjectClassという属性を表すことになります。

Bob用のデータ・エントリーに追加する個々の属性に対して、BasicAttributeクラスのインスタンスを1つずつインスタンス化します。

## Step 3C. 各属性に対して値を指定する

Bobのデータ・エントリーに含める個々の属性にBasicAttributeオブジェクトを1つずつ割り当てる場合、各属性に値を指定します。値を指定するには、Attributeインターフェースのadd() メソッドを呼び出します。このadd() メソッドには、パラメーターが1つだけ設定されています。このパラメーターには、各属性に指定する値を文字列の形式で設定します。

add() メソッドは、実際にはJava Objectクラスのインスタンスを取得します。すべてのJava オブジェクトはObjectクラスから拡張するため、文字列の値をadd() メソッドに渡すことができます。複数の値を持つ属性が存在する場合は、その属性に対して必要な値をすべて指定するまでadd() メソッドを繰り返し呼び出すことができます。

## Step 3D. 属性のコレクションを作成する

ここまでの作業で、すべての属性が揃いました。今度は、これらの属性をAttributeオブジェクトのコレクションに格納します。JNDIには、Attributesというインターフェースが用意されています。また、BasicAttributesというクラス内で、Attributesインターフェースの基本的な機能が実装されます。BasicAttributesオブジェクトをインスタンス化し、このオブジェクトのput() メソッドを任意の回数呼び出して、すべてのAttributeオブジェクトを (1度に1つずつ) コレクションの中に格納します。

リスト4の「Step 4」で示したように、次に3つのパラメーターを持つbind() メソッドを呼び出します。3つのパラメーターを持つbind() メソッドは、リスト1で使用した2つのパラメーターを持つbind() メソッドと似ていますが、3つ目のパラメーターには今作成した属性のコレクションを指定します。

## アプリケーション3. マーシャルされたJavaオブジェクトを保管する

Javaオブジェクトを保管する最後の演習として、前回の記事の「マーシャルされたJavaオブジェクトを表す」セクションで簡単に取り上げた、マーシャルされたJavaオブジェクトの保管方法を説明します。

マーシャルされた形式で保管するJavaオブジェクトは、(リスト1の「Step 3」で作成した直列化可能なMessagingPreferencesオブジェクトのように) 直列化可能である必要があります。ここでは保管方法の仕組みを説明するため、これと同じMessagingPreferencesオブジェクトを使用して、ApacheDSをマーシャルして保管する方法を紹介します。

初めに、リスト5のStoreAlicePreferencesInMarshalledFormアプリケーションを見てみましょう。マーシャルされたJavaオブジェクトをApacheDSに保管する際に必要な手順がすべて示されています。

### リスト5. StoreAlicePreferencesInMarshalledForm

```
public class StoreAlicePreferencesInMarshalledForm {
    public StoreAlicePreferencesInMarshalledForm ()
    {
        try {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            //Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-------------------------------------------
            //Step3: Instantiate a Java Object
            //-------------------------------------------
            MessagingPreferences preferences = new MessagingPreferences();
            MarshalledObject mObj= new MarshalledObject(preferences );

            //-------------------------------------------
            //Step4: Storing Java object in ApacheDS
            //-------------------------------------------
            String bindContext = "cn=marshalledPreferences,uid=alice,ou=users";
            ctx.bind( bindContext, mObj);
        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
        StoreAlicePreferencesInMarshalledForm storeAlicePref =
            new StoreAlicePreferencesInMarshalledForm();
    }
}
```

J2SEはマーシャリング・プロセスを内部処理するため、リスト5のアプリケーションはリスト1のStoreAlicePreferencesアプリケーションと非常によく似ています。この2つのアプリケーションを比較すると、リスト5の「Step 3」には、たった1行だけ余分なコードがあることが分かり

ます。MessagingPreferencesオブジェクトをインスタンス化した後は、java.rmi.MarshalledObjectもインスタンス化して、設定オブジェクトをjava.rmi.MarshalledObjectコンストラクターに渡します。こうすると、java.rmi.MarshalledObjectクラスはマーシャリング・プロセスを処理して、MessagingPreferencesオブジェクトのマーシャルされたバージョンを保管します。

「Step 4」では、元のMessagingPreferencesオブジェクトの代わりに、マーシャルされたオブジェクトを単に保管 (またはバインド) します。

これで、ApacheDSへJavaオブジェクトを保管する場合の説明は終わりです。ここからは、ApacheDSに保管されたデータ検索について示す2つの例を見てみましょう。

## アプリケーション4. 保管されたデータを検索する

まず、ApacheDS上での簡単な検索操作から見ていきます。ApacheDSインスタンス内に多数のユーザーが存在していると仮定し、ユーザーAliceのすべての詳細を検索するとします。Aliceについて現在分かっていることは、以下のとおりです。

- Aliceはユーザーであるため、ユーザー用のデータ組織単位内のデータ・エントリーからAliceを検索する必要がある (組織単位または「ou」の概念は、前回の記事で説明しています)。
- Aliceのユーザー名は「alice」である (大文字/小文字を区別しない)。
- Aliceは人物であるため、Aliceのデータ・エントリーは、直接的または間接的に個人のオブジェクト・クラスを拡張する何らかのオブジェクト・クラスを使用する必要がある。

ではここで、非常に簡単な検索シナリオを実行するサンプル・アプリケーションSearchForAlice.SearchForAliceを示したリスト6を見てみましょう。より詳細な検索シナリオも対象にできるよう、後でこのアプリケーションを拡張します。

### リスト6. SearchForAlice

```
public class SearchForAlice {
    public SearchForAlice() {
        try
        {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            // Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-----------------------------------------
            //Step3: Setting search context
            //-----------------------------------------
            String searchContext = "ou=users";

            //-----------------------------------------
            //Step4: Creating search attributes for Alice
            //-----------------------------------------
            Attribute uid = new BasicAttribute("uid");
            Attribute objclass = new BasicAttribute("objectClass");
```

```
            //adding attribute values
            uid.add("Alice");
            objclass.add("person");

            //Instantiate Attributes object and put search attributes in it.
            Attributes attrs = new BasicAttributes(true);
            attrs.put(uid);
            attrs.put(objclass);

            //----------------------------------------
            //Step5: Executing search
            //----------------------------------------
            NamingEnumeration ne = ctx.search(searchContext, attrs);

            if (ne != null)
            {
                //Step 6: Iterating through SearchResults
               while (ne.hasMore()) {
                    //Step 7: Getting individual SearchResult object
                    SearchResult sr = (SearchResult) ne.next();

                    //Step 8:
                    String entryRDN = sr.getName();
                    System.out.println("RDN of the Searched entry: "+entryRDN);

                    //Step 9:
                    Attributes srAttrs = sr.getAttributes();

                    if (srAttrs != null) {
                          //Step 10:
                          for (Enumeration e = attrs.getAll() ; e.hasMoreElements() ;)
                          {
                              Attribute attr = (Attribute) e.nextElement();

                              //Step 11:
                              String attrID = attr.getID();
                              System.out.println("Attribute Name: "+attrID);
                              System.out.println("Attribute Value(s):");

                              NamingEnumeration e1 = attr.getAll();
                              while (e1.hasMore())
                                 System.out.println("\t\t"+e1.nextElement());
                          }//for()
                    }//if (srAttrs)
               }
            }//if (ne != null)

        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
      SearchForAlice searchAlice = new SearchForAlice();
    }
}
```

リスト6の検索アプリケーションは、11の手順から構成されています。このリストに示した最初の2つの手順が、リスト1に示した最初の2つの手順 (JNDIプロパティーの読み込み手順とDirContexオブジェクトのインスタンス化手順) と似ていることが分かります。

リスト1の「Step 1」にあるJNDIプロパティーjava.naming.provider.urlについての説明で、プロバイダーのURLが2つのコンポーネントから構成されており、その1つが作業対象となるディレクト

リー・コンテキストであると説明しました。リスト4のjava.naming.provider.urlプロパティーの値が「ou=system」となっていることが分かると思います。ou=system文字列は、作業対象となるディレクトリー・コンテキストです。そのため、すべての検索操作は、このディレクトリー・コンテキスト内で自動的に実行されます。

この例では検索操作を行うため、ou=systemディレクトリー・コンテキストを検索コンテキストと呼ぶことができます。それでは、この検索アプリケーションの残りの手順を見ていきましょう。

- **Step 3. 検索コンテキストを絞り込みます。**Aliceがユーザーであることが分かっているので、ou=system検索コンテキスト全体に渡ってAliceを検索する代わりに、ユーザーの組織単位ou=users内で検索を実行します。
- **Step 4. 検索属性を作成します。**Aliceについて分かっている情報が、検索属性になります。Aliceのuidとオブジェクト・クラスは分かっているので、uidとobjectClassという2つの属性だけのコレクションを作成します。この操作は、リスト6の「Step 4」に記述されています(uidはRDNのコンポーネントであり、属性ではないことは、前回の記事で説明しました。ただし検索パラメーターを指定する場合には、uid値を属性値のように指定することがJNDIの要件になります)。
- **Step 5. 検索を実行します。**ここでは、リスト6の「Step 2」で取得したDirContextオブジェクトのsearch() メソッドを呼び出します。このsearch() メソッドには、2つのパラメーターが設定されています。1つ目のパラメーターは、この演習の「Step3」で作成した検索コンテキストで、2つ目のパラメーターは、「Step 4」の2つの属性のコレクションです。search() メソッドは、検索結果を含むNamingEnumerationオブジェクトを返します。

「Step 1」から「Step 5」は、検索操作を設定する手順です。残りの手順は、NamingEnumerationオブジェクトを処理して検索結果を抽出する手順になります。

- **Step 6. 検索結果を取得します。**リスト6の「Step 6」にあるNamingEnumerationオブジェクトには、検索結果のコレクションが含まれています。このコレクションの各検索結果は、SearchResultオブジェクトによって表されたものです。個々の検索結果は、NamingEnumerationオブジェクトを通した作業を繰り返すだけで取得することができます。
- **Step 7. 個々の検索結果を処理します。**それぞれの検索結果には、1つのデータ・エントリーについての情報しか含まれていないことに注意してください。SearchResultオブジェクトからは、データ・エントリーについての2つの情報 (RDNとそのすべての属性) を取得することができます。
- **Step 8. getName() を呼び出します。**SearchResultオブジェクトのgetName() メソッドは、検索しているエントリーのRDNを返します。AliceのRDNはuid=aliceです。
- **Step 9. getAttributes() を呼び出します。**SearchResultオブジェクトのgetAttributes() メソッドは、検索しているエントリーに関連するすべての属性値を含んだAttributesオブジェクトを返します。このAttributesオブジェクトは、リスト6の「Step 4」で作成した属性のコレクションと同様の属性のコレクションを表しています。
- **Step 10. getAll() を呼び出します。**Attributesオブジェクトのget All() メソッドは、コレクション内のすべての属性を含んだ列挙を返します。
- **Step 11. 属性を処理します。**ここでようやく、コレクションの中から1つの属性を取り出して、そのgetID() メソッドとgetAll() メソッドを呼び出します。getID() メソッドは、文字列として属性の名前を返します。getAll() メソッドは、列挙形式で属性のすべての値を返します。

# アプリケーション5. 名前で検索

上の検索の例では、ユーザーのuidが分かっている場合にユーザーを検索する方法について見てきました。今度の例では、uidの代わりに名前を使用してAliceを検索できるようにアプリケーションを変更する方法について説明します。

前回の記事では、ユーザーの名前がユーザーのデータ・エントリーのcn属性値として保管されている様子を図18で示しました。そのため、このアプリケーションでは、リスト7で示すようにcn属性で検索を行います。

## リスト7. SearchForAliceByCN

```
public class SearchForAliceByCN {
    public SearchForAliceByCN() {
        try
        {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            // Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-----------------------------------------
            //Step3: Setting search context
            //-----------------------------------------
            String searchContext = "ou=users";

            //-----------------------------------------
            //Step4: Creating search attributes for Alice
            //-----------------------------------------
            Attribute cn = new BasicAttribute("cn");
            Attribute objclass = new BasicAttribute("objectClass");

            //putting attribute values
            cn.add("Alice");
            objclass.add("person");

            //Instantiate an Attributes object and put search attributes in it
            Attributes attrs = new BasicAttributes(true);
            attrs.put(cn);
            attrs.put(objclass);

            //-----------------------------------------
            //Step5: Executing search
            //-----------------------------------------
            NamingEnumeration ne = ctx.search(searchContext, attrs);

            if (ne != null)
            {
                //Step 6: Iterating through SearchResults
                while (ne.hasMore()) {
                    //Step 7: Getting individual SearchResult object
                    SearchResult sr = (SearchResult) ne.next();

                    //Step 8:
                    String entryRDN = sr.getName();
```

```
                    //Step 9:
                    Attributes srAttrs = sr.getAttributes();

                    if (srAttrs != null) {
                          //Step 10:
                         for (Enumeration e = attrs.getAll() ; e.hasMoreElements() ;)
                         {
                              Attribute attr = (Attribute) e.nextElement();

                              //Step 11:
                              String attrID = attr.getID();
                              System.out.println("Attribute Name: "+attrID);
                              System.out.println("Attribute Value(s):");

                              NamingEnumeration e1 = attr.getAll();
                                while (e1.hasMore())
                                   System.out.println("\t\t"+e1.nextElement());
                         }//for()
                    }//if (srAttrs)
                }
            }//if (ne != null)

        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
        SearchForAliceByCN searchAlice = new SearchForAliceByCN();
    }
}
```

SearchForAliceByCNアプリケーションは、名前を使用してAliceを検索する手順を示したものです。このアプリケーションは、1つの点を除いて、前のSearchForAliceアプリケーションと非常によく似ています。リスト6の「Step 4」では、検索のためにuidおよびobjectClass属性のコレクションを作成しましたが、このアプリケーションの「Step 4」では、cnおよびobjectClass属性のコレクションを作成します。

### マッチング規則について

cn属性で検索する場合には、注意すべき重要な点が1つあります。前回の記事の図13で、cn属性タイプには、サブストリング一致のためのマッチング規則を定義するフィールドSUBSTRが設定されていると説明しました。

cn属性の場合は、SUBSTRフィールドの値はcaseIgnoreMatchになります。そのため、cn属性の特定の値を検索する場合は、検索対象の名前がcn属性値のサブストリングと一致した場合でも、検索に成功したと判断されてしまいます。さらに、サブストリングの一致は、大文字/小文字を区別しません。

したがって、「alice」を検索すると、ファーストネーム、ミドルネーム、ラストネームのいずれかに「Alice」が含まれるすべてのユーザーが検索結果として表示されます。

## アプリケーション6. Javaオブジェクトを非直列化する

ここまでに、ApacheDSへのJavaオブジェクトの保管方法と、保管されたオブジェクトに関連付けられた属性の検索方法について見てきました。ここでは、Javaオブジェクトの検索および非直列化

の方法について説明します。非直列化は直列化の場合とは逆に、Javaオブジェクトの直列化された形式からJavaオブジェクトを作成します。

リスト8に示したアプリケーションは、AliceのMessagingPreferencesオブジェクトを検索して非直列化します。リスト1で、MessagingPreferencesオブジェクトをApacheDSに保管したことを思い出してください。

FetchAliceMessagingPreferencesアプリケーションは、リスト7のSearchForAliceByCNアプリケーションの拡張版です。実際にリスト8の手順は、Aliceのデータ・エントリーのRDNを拡張するリスト7の「Step 8」までは同じです。リスト8の「Step 8」以降では、AliceのPreferencesオブジェクトの検索から説明を始めます。

## リスト8. FetchAliceMessagingPreferences

```
public class FetchAliceMessagingPreferences {
    public FetchAliceMessagingPreferences() {
        try
        {

            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            // Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-------------------------------------------
            //Step3: Setting search context
            //-------------------------------------------
            String searchContext = "ou=users";

            //-------------------------------------------
            //Step4: Creating search attributes for Alice
            //-------------------------------------------
            Attribute cn = new BasicAttribute("cn");
            Attribute objclass = new BasicAttribute("objectClass");

            //putting attribute values
            cn.add("Alice");
            objclass.add("person");

            //Instantiate an Attributes object and put search attributes in it
            Attributes attrs = new BasicAttributes(true);
            attrs.put(cn);
            attrs.put(objclass);

            //-----------------------------------------
            //Step5: Executing search
            //-----------------------------------------
            NamingEnumeration ne = ctx.search(searchContext, attrs);

            if (ne != null)
            {
                //Step 6: Iterating through SearchResults
                while (ne.hasMore()) {
                    //Step 7: Getting individual SearchResult object
```

```
                    SearchResult sr = (SearchResult) ne.next();

                    //Step 8:
                    String entryRDN = sr.getName();

                    //-------------------------------------------
                    //Step9: Setting a new search context
                    //-------------------------------------------
                    searchContext = entryRDN + "," + searchContext;

                    //-------------------------------------------
                    //Step10: Creating search controls
                    //-------------------------------------------
                    SearchControls ctls = new SearchControls();
                    ctls.setReturningObjFlag(true);
                    ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

                    //-------------------------------------------
                    //Step11: Creating filter
                    //-------------------------------------------
                    String filter = "(|(javaClassName=MessagingPreferences)
                                       (javaClassName=ShippingPreferences))";

                    //-----------------------------------------
                    //Step12: Executing search
                    //-----------------------------------------
                    NamingEnumeration ne1 = ctx.search(searchContext, filter, ctls);

                    if (ne != null)
                    {
                        //Step13: Iterating through SearchResults
                        while (ne1.hasMore()) {
                            //Step14: Getting individual SearchResult object
                            SearchResult sr1 = (SearchResult) ne1.next();

                            //Step15: Getting preferences object
                            Object obj = sr1.getObject();

                            if (obj != null) {
                               if (obj instanceof MessagingPreferences){
                                   MessagingPreferences pref =
                                     (MessagingPreferences) obj;
                               }
                               else if (obj instanceof ShippingPreferences) {
                                  ShippingPreferences pref = (ShippingPreferences) obj;
                               }
                            }
                        }//while(
                    }//if
                }//while
            }//if (ne != null)
        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
        FetchAliceMessagingPreferences searchAlicePreferences =
             new FetchAliceMessagingPreferences();
    }
}
```

## 検索コンテキストの復習

AliceのMessagingPreferencesオブジェクトを検索して非直線化する手順に進む前に、Aliceのデータ・エントリー内にあるAliceのMessagingPreferencesオブジェクトを示した図1を振り返ってみま

しょう。そのためには、Aliceのデータ・エントリー内のMessagingPreferencesオブジェクトを検索する必要があります。

データ・エントリーの「内部を検索」するには、どうすればいいでしょうか。そのためには、検索コンテキストの概念を応用します。特にこの目的のためには、検索コンテキストの概念が必要になります。この場合、リスト6で示した最初の検索アプリケーションSearchForAliceの検索コンテキストに対して、絞り込みを行う必要があります。

この作業は、リスト8の「Step 9」で実行します。この手順で、Aliceのデータ・エントリー (ou=users,ou=system) の検索に使用した元の検索コンテキストにAliceのRDN (uid=alice) を連結します。その結果できた検索コンテキスト (uid=alice,ou=users,ou=system) により、Aliceのデータ・エントリー内の検索が可能になります。

それでは、FetchAliceMessagingPreferencesアプリケーションの残りの手順に進みましょう。

## 検索制御の構築と使用

リスト8の「Step 10」では、SearchControlsオブジェクトをインスタンス化します。インスタンス化したSearchControlsオブジェクトは、一部の検索制御を構築する際に使用できます。検索制御は、主に以下の2つの目的で使用します。

- 検索結果に含まれるデータの種類の指定。この場合、Javaオブジェクトを受信するには、SearchControlsオブジェクトのsetReturningObjFlag() メソッドを呼び出します。このメソッドは、検索を実行することによってオブジェクトが取得されるように、検索制御内にフラグを立てます。
- 検索範囲の特定。「検索範囲」とは、特定のデータ・エントリー内を検索するか、そのエントリーに従属する下位レベルも同様に検索するかという条件を表します。検索範囲は、SearchControlsオブジェクトのsetSearchScope() メソッドを呼び出すことで設定します。

## 検索結果のフィルター処理

リスト8の「Step 11」では、文字列「filter」を作成します。フィルター文字列の値は (|(javaClassName=MessagingPreferences) (javaClassName=ShippingPreferences)) であることが分かります。括弧でくくられた、属性と値から成る2つのペアは、単一の属性javaClassNameに対して異なる値を指定します。また、属性と値から成るこれら2つのペアの前にある「OR」にも注意してください。これは、MessagingPreferencesオブジェクトまたはShippingPreferencesオブジェクトのいずれかを検索対象にするという意味です。

このフィルター文字列は、検索結果に対するフィルターとして機能します。つまり、検索操作の後で返された検索結果には、検索フィルターで指定された条件を満たす結果のみが含まれることになります。

このタイプの検索フィルターは、多数ある属性値のいずれかを検索する場合に使用します。検索フィルターの詳細については、「参考文献」のセクションを参照してください。LDAPアプリケーションでの検索フィルターの概念を紹介する記事へのリンクがあります。

### 検索結果を取得して処理する

リスト8の「Step 12」では、search() メソッドを呼び出して、検索コンテキスト、検索フィルター、および検索制御をメソッド呼び出しと共に渡します。

リスト6の「Step 5」で使用した検索メソッド呼び出しと、リスト8で使用した検索メソッド呼び出しの違いに注意してください。リスト6では、2つの引数を持つsearch() メソッド形式を使用していますが、リスト8では、同じメソッドで3つの引数を持つオーバーロード形式を使用します。

リスト8の「Step 13」と「Step 14」は、リスト7の「Step6」と「Step 7」とそれぞれ同じ手順になります。これらの手順では、NamingEnumerationオブジェクトを処理し、個々の検索結果をSearchResultオブジェクトの形式で取得します。

最後に「Step 15」で、SearchResultオブジェクトのgetObject() メソッドを呼び出します。getObject() メソッドは、Javaオブジェクトを返します。

検索要求では2つのクラス (MessagingPreferencesまたはShippingPreferences) を指定しているため、getObject() メソッドによってどのインスタンスのクラスが返されたのかは判断できません。返された結果は、これら2つのクラスに属するインスタンスのうち、どちらのインスタンスでもある可能性があるためです。この場合、最初にそのオブジェクトがどちらのインスタンスに属するかを確認し、その結果に応じてそのオブジェクトをキャストする必要があります。この作業が完了すると、Javaオブジェクトの制御が可能になり、このオブジェクトのメソッドを呼び出すことができるようになります。

## アプリケーション7. Javaオブジェクトのアンマーシャル

上のアプリケーションでは、直列化されたJavaオブジェクトを非直列化する方法について説明しました。次に、FetchAliceMarshalledPreferencesアプリケーションを使って、マーシャルされたJavaオブジェクトをアンマーシャルする方法について見ていきます。

### リスト9. FetchAliceMarshalledPreferences

```
public class FetchAliceMarshalledPreferences {
    public FetchAliceMarshalledPreferences() {
        try
        {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");

            //-----------------------------------------
            // Step2: Fetching a DirContext object
            //-----------------------------------------
            DirContext ctx = new InitialDirContext(properties);

            //-------------------------------------------
            //Step3: Setting search context
            //-------------------------------------------
            String searchContext = "ou=users";

            //-------------------------------------------
            //Step4: Creating search attributes for Alice
```

```
            //-----------------------------------------
            Attribute cn = new BasicAttribute("cn");
            Attribute objclass = new BasicAttribute("objectClass");

            //putting attribute values
            cn.add("Alice");
            objclass.add("person");

            //Instantiate an Attributes object and put search attributes in it
            Attributes attrs = new BasicAttributes(true);
            attrs.put(cn);
            attrs.put(objclass);

            //-----------------------------------------
            //Step5: Executing search
            //-----------------------------------------
            NamingEnumeration ne = ctx.search(searchContext, attrs);

            if (ne != null)
            {
                //Step 6: Iterating through SearchResults
                while (ne.hasMore()) {
                    //Step 7: Getting individual SearchResult object
                    SearchResult sr = (SearchResult) ne.next();

                    //Step 8:
                    String entryRDN = sr.getName();
                    System.out.println("RDN of the Searched entry: "+entryRDN);

                    //-------------------------------------------
                    //Step9: Setting a new search context
                    //-------------------------------------------
                    searchContext = entryRDN + "," + searchContext;

                    System.out.println("new SearchContext: "+searchContext);

                    //-------------------------------------------
                    //Step10: Creating search controls
                    //-------------------------------------------
                    SearchControls ctls = new SearchControls();
                    ctls.setReturningObjFlag(true);
                    ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

                    //-------------------------------------------
                    //Step11: Creating filter
                    //-------------------------------------------
                    String filter = "(javaClassName=java.rmi.MarshalledObject)";

                    //-----------------------------------------
                    //Step12: Executing searchl
                    //-----------------------------------------
                    NamingEnumeration ne1 = ctx.search(searchContext, filter, ctls);

                    if (ne != null)
                    {
                        //Step13: Iterating through SearchResults
                        while (ne1.hasMore()) {
                            //Step14: Getting individual SearchResult object
                            SearchResult sr1 = (SearchResult) ne1.next();

                            //Step15: Getting preferences object
                            Object obj = sr1.getObject();

                            if (obj instanceof MarshalledObject) {
                                MarshalledObject mObj= (MarshalledObject) obj;
                                Object obj1 = mObj.get();
```

```
                                if (obj1 instanceof MarshalledObject) {
                                    MessagingPreferences pref =
                                      (MessagingPreferences) obj;
                                }
                                else if (obj1 instanceof ShippingPreferences) {
                                    ShippingPreferences pref =
                                      (ShippingPreferences) obj;
                                }
                            }//if(obj)

                        }//while
                    }//if
                }//while
            }//if (ne != null)

        } catch (Exception e) {
            System.out.println("Operation failed: " + e);
        }
    }

    public static void main(String[] args) {
        FetchAliceMarshalledPreferences fetchAlicePref =
            new FetchAliceMarshalledPreferences();
    }
}
```

リスト9には15の手順から構成されています。そのうちの「Step 14」までは、リスト8の「Step 14」までとまったく同じ手順です。リスト8とリスト9のアプリケーションにおける唯一の違いは、「Step 15」だけです。この手順では、SearchResult.getObject() メソッドがマーシャルされたオブジェクトを返します。この返されたオブジェクトのクラス名を確認することにより、自分自身で「Step 15」の内容を確かめることができます。確認後、このオブジェクトをMarshalledObjectとしてキャストします。

Javaオブジェクトの直列化された形式はこのマーシャルされたオブジェクト内にあるため、MarshalledObjectクラスのget() メソッドを呼び出してJavaオブジェクトを取得します。このJavaオブジェクトは、クラスの確認後にキャストすることができます。

## アプリケーション8. 保管されたオブジェクトを編集して更新する

ここまでの例では、Javaオブジェクトの保管と検索を中心に見てきました。次のアプリケーションでは、すでにApacheDSに保管されているJavaオブジェクトの編集および更新の方法について説明します。

リスト10のUpdateAlicePreferencesアプリケーションは、リスト4で個人のオブジェクト・クラスと統合したJavaオブジェクトを更新するアプリケーションです。

### リスト10. UpdateAlicePreferences

```
public class UpdateAlicePreferences {
    public UpdateAlicePreferences() {
        try
        {
            //-----------------------------------------
            //Step1: Setting up JNDI properties for ApacheDS
            //-----------------------------------------
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
            properties.setProperty("java.naming.security.credentials", "secret");
```

```
        //-----------------------------------------
        // Step2: Fetching a DirContext object
        //-----------------------------------------
        DirContext ctx = new InitialDirContext(properties);

        //-------------------------------------------
        //Step3: Setting search context
        //-------------------------------------------
        String searchContext = "ou=users";

        //-------------------------------------------
        //Step4: Creating search attributes for Alice
        //-------------------------------------------
        Attribute cn = new BasicAttribute("cn");
        Attribute objclass = new BasicAttribute("objectClass");

        //putting attribute values
        cn.add("Alice");
        objclass.add("person");

        //Instantiate an Attributes object and put search attributes in it
        Attributes attrs = new BasicAttributes(true);
        attrs.put(cn);
        attrs.put(objclass);

        //-----------------------------------------
        //Step5: Executing search
        //-----------------------------------------
        NamingEnumeration ne = ctx.search(searchContext, attrs);

        if (ne != null)
        {
            //Step 6: Iterating through SearchResults
            while (ne.hasMore()) {
                //Step 7: Getting individual SearchResult object
                SearchResult sr = (SearchResult) ne.next();

                //Step 8:
                String entryRDN = sr.getName();

                //-------------------------------------------
                //Step9: Setting a new search context
                //-------------------------------------------
                searchContext = entryRDN + "," + searchContext;

                //-------------------------------------------
                //Step10: Creating search controls
                //-------------------------------------------
                SearchControls ctls = new SearchControls();
                ctls.setReturningObjFlag(true);
                ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

                //-------------------------------------------
                //Step11: Creating filter
                //-------------------------------------------
                String filter = "(javaClassName=java.rmi.MarshalledObject)";

                //-----------------------------------------
                //Step12: Executing search
                //-----------------------------------------
                NamingEnumeration ne1 = ctx.search(searchContext, filter, ctls);

                if (ne != null)
                {
                    //Step13: Iterating through SearchResults
                    while (ne1.hasMore()) {
```

```
                        //Step14: Getting individual SearchResult object
                        SearchResult sr1 = (SearchResult) ne1.next();

                        //Step15: Getting preferences object
                        Object obj = sr1.getObject();
                        entryRDN = sr1.getName();

                        if (obj != null && obj instanceof MarshalledObject)

                        {
                            MarshalledObject mObj= (MarshalledObject) obj;
                            MessagingPreferences pref =
                              (MessagingPreferences) mObj.get();

                            //Step16: Updating your java object
                            pref.setStyles("http://www.mystyles.com/preferences");

                            //Step17: Setting a new context for data updation
                            String bindContext = entryRDN + "," + searchContext;

                            //Step18: Reading attributes in a new collection
                            Attributes attrs1 = sr1.getAttributes();

                            //Step19:Updating data
                            ctx.rebind(bindContext, pref, attrs);
                            System.out.println("Updating
                              the MessagingPreferences succeed");
                        }
                    }//while
                }//if
            }//while
        }//if (ne != null)

    } catch (Exception e) {
        System.out.println("Operation failed: " + e);
    }
  }

  public static void main(String[] args) {
    UpdateAlicePreferences updateAlicePreferences =
        new UpdateAlicePreferences();
  }
}
```

リスト10で示す最初の15の手順は、これまでの説明から理解できると思います。これらの手順は、更新するJavaオブジェクトを検索して取得するだけのものです。これ以降の手順では、そのオブジェクトを更新して保管する作業になります。

## Javaオブジェクトを更新して保管する

「Step 16」では、Javaオブジェクトのメソッドを呼び出して、Javaオブジェクト内のデータを更新します。データの更新後は、編集したバージョンのJavaオブジェクトを保管します。しかしその前に、以下の2つの作業を行う必要があります。

- **名前付きコンテキストの取得**:既存のエントリーを更新するということは、以前に書き出された同じ名前付きコンテキストに対してデータを更新するということです。そのためには、更新対象とするエントリーの名前付きコンテキストを把握しておく必要があります。このJavaオブジェクトの名前付きコンテキストは、オブジェクトのRDNと (オブジェクトの検索に使

用した) 検索コンテキストを連結させることで作成できます。この操作は、リスト10の「Step 17」で確認することができます。
- **更新されたJavaオブジェクトにすべての属性を記述する**:更新したJavaオブジェクトをApacheDSに戻して書き込むと、同じ名前付きコンテキスト上に新しいデータ・エントリーとして書き出されます。このデータ・エントリーに関連付けられた属性は、すべて失われます。このため、データ・エントリーのすべての属性を、更新されたJavaオブジェクトに戻して記述する必要があります。リスト10の「Step 18」では、検索結果からすべての属性を読み込み、それらを属性のコレクションにまとめることで、この処理を行っています。

最後に「Step 19」で、DirContext.rebind() メソッドを呼び出します。rebind() メソッドには、リスト4のStoreBobPreferencesアプリケーションの「Step 4」で使用した、3つのパラメーターを持つbind() メソッドに渡したものとまったく同じパラメーターが設定されています。

bind() とrebind() メソッドの唯一の違いは、rebind() メソッドの場合、既存のデータ・エントリーによってすでに占有されている名前付きコンテキストにデータ・エントリーを格納し、この既存のエントリーを新しいデータで効果的に更新するという点です。

## アプリケーション9. すべての機能をまとめる

最後のアプリケーションでは、これまでに見てきたすべての概念を集約し、ApacheDS内でJavaオブジェクトの保管、検索、削除、更新が可能な、単純で再使用可能なオブジェクトにまとめます。このLDAP4JavaObjectsクラスを、リスト11に示します。

### リスト11. LDAP4JavaObjects

```
public class LDAP4JavaObjects {

    protected String commonName = null;
    protected String surName = null;
    protected String userID  = null;
    protected String javaObjectCN = null;
    protected String userName = null;
    protected String password = null;
    protected String initialContext = null;
    private String workingContext = null;
    protected DirContext dirContext = null;
    protected Properties properties = null;
    protected Object javaObject = null;
    protected Attributes attributes = null;

    public LDAP4JavaObjects() {
        properties = new Properties();
        attributes = new BasicAttributes(true);

        workingContext = "ou=users";
        initialContext = workingContext;

        try {
            InputStream inputStream = new FileInputStream( "ApacheDS.properties");
            properties.load(inputStream);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }//LDAPJavaObjects

    public void setUserName(String userName){
       properties.setProperty("java.naming.security.principal", userName);
       this.userName = userName;
    }
```

```
public void setPassword(String password) {
   properties.setProperty("java.naming.security.credentials", password);
   this.password = password;
}

protected void connect() {
    try {
        // Fetch the directory context.
        dirContext= new InitialDirContext(properties);
    } catch (NamingException e) {
        System.out.println("Getting initial context operation failed: " + e);
    }
}

protected void close() {
    try {
        // Close the directory context.
        dirContext.close();
    } catch (NamingException e) {
        System.out.println("Closing initial context operation failed: " + e);
    }
}

public void setJavaObject(Object obj) throws java.io.NotSerializableException {
   if (obj instanceof java.io.Serializable)
       javaObject = obj;
}

protected boolean isObjectPresent(String uid, String cn, String workContext) {
    NamingEnumeration ne = search(uid, cn, workContext);
    if (ne != null)
       return true;
    else
       return false;
}//isObjectPresent

protected NamingEnumeration search(String user, String object, String workContext) {

    NamingEnumeration  ne = null;
    String filter =  new String();
    SearchControls ctls = new SearchControls();
    ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

    try {
        if (user != null && object == null) {
            addUserAttribute ("uid", user);
            addUserAttribute ("objectClass", "person");
            ne =  dirContext.search(workContext, attributes);
            if (ne != null && ne.hasMore())
                return ne;
            else
                return null;
        }else if (user == null &&  object != null)
        {
            filter = "(cn="+object+")";
            ctls.setReturningObjFlag(true);
            ne =  dirContext.search(workContext, filter, ctls);

            if (ne != null && ne.hasMore())
               return ne;
            else
                return null;
        }
    } catch (NamingException e) {
        e.printStackTrace();
    }
```

```
        return null;
    }//search

    protected void update(String uid, String cn, String workContext) {
        if (uid != null && cn == null)
        {
           String[] objClassValues = {"top","person"};
           addUserAttribute ("uid", uid);

           if (commonName != null)
               addUserAttribute ("cn", commonName);

           addUserAttributes ("objectClass", objClassValues);

           try  {
               dirContext.rebind(workContext, null, attributes);
           } catch (NamingException e) {
               System.out.println("Storing  operation failed: " + e);
           }
        } else if (uid == null && cn != null)  {
            addUserAttribute ("cn", cn);

            try  {
                dirContext.rebind(workContext, javaObject, attributes);
            } catch (NamingException e) {
                e.printStackTrace();
            }
        }
    }//update


    protected void store(String uid, String cn, String workContext) {
        if (uid != null && cn == null)
        {
            String[] objClassValues = {"top","person"};
            addUserAttribute ("uid", uid);

            if (commonName != null)
                addUserAttribute ("cn", commonName);

            addUserAttributes ("objectClass", objClassValues);
            try
            {
               dirContext.bind(workContext, null, attributes);
            } catch (NamingException e) {
               e.printStackTrace();
            }
        } else if (uid == null && cn != null) {
            addUserAttribute ("cn", cn);
            try  {
                dirContext.bind(workContext, javaObject, attributes);
            } catch (NamingException e) {
                e.printStackTrace();
            }
        }
    }//store


    public void addUserAttribute (String name, String value){
        Attribute attr = new BasicAttribute(name);
        attr.add(value);
        attributes.put(attr);
    }//addUserAttribute


    public void addUserAttributes(String name, String[] value) {
        Attribute attr = new BasicAttribute (name);
```

```
        if (value != null) {
            for (int i =0; i < value.length; i++)
                attr.add(value[i]);
        }
        attributes.put(attr);
    }//addUserAttribute

    public void writeToServer () {
        connect();

        if (userID == null && commonName != null)
            userID = commonName;

        //Check if the cn of the Java object is set.
        //If not, use commonName as object cn.
        if (javaObject != null)
        {
            if (javaObjectCN == null && commonName != null)
                javaObjectCN = commonName;
        }

       if (!(isObjectPresent(userID, null, initialContext)))
       {
          workingContext = "uid="+userID+","+initialContext;
          store(userID, null, workingContext);
          workingContext = "cn="+javaObjectCN +",uid="+userID+","+initialContext;
          store( null, javaObjectCN, workingContext);
       } else if (isObjectPresent(null, javaObjectCN,
         "uid="+userID+","+initialContext)) {
          workingContext = "cn="+javaObjectCN +",uid="+userID +","+ initialContext;
          update(null, javaObjectCN, workingContext);
       } else {
          workingContext = "cn="+javaObjectCN +",uid="+userID +","+ initialContext;
          store(null, javaObjectCN, workingContext);
       }
       close();
    }//writeToServer()


    public void searchFromServer(String user, String object) {
        connect();
        NamingEnumeration ne = search(user, object, initialContext);

        if (ne != null)
            processSearchResult(ne);
        else
            System.out.println ("searchFromServer... failed");
        close();
    }//searchFromServer

    private void processSearchResult (NamingEnumeration ne){
        try {
            if (ne!=null)
            {
                while (ne.hasMore()) {
                    SearchResult sr = (SearchResult) ne.next();
                    Object obj = sr.getObject();

                    if (obj != null) {
                        javaObject = obj;
                        javaObjectCN = sr.getName();
                    } else {
                        userID = sr.getName();
                        processAttributes(sr.getAttributes());
                    }
                }//while
            }//if (ne != null)
```

```
        } catch (javax.naming.NamingException e) {
            e.printStackTrace();
        }
    }//processSearchResult

    private void processAttributes(Attributes attrs){
        try {
            for (Enumeration e = attrs.getAll() ; e.hasMoreElements() ;) {
                Attribute attr = (Attribute) e.nextElement();
                if (attr.getID().equals("cn"))
                    commonName = (String)attr.get();
                else if (attr.getID().equals("sn"))
                    surName = (String)attr.get();
            }
        } catch (javax.naming.NamingException ne){
            ne.printStackTrace();
        }
    }//processAttributes

    public void setUserID(String userID) {
        this.userID = userID;
    }

    public String getUserID() {
        return userID;
    }

    public void setJavaObjectCN(String objectCN) {
        this.javaObjectCN = objectCN;
    }

    public String getJavaObjectCN() {
        return javaObjectCN;
    }

    public void setCommonName (String cn) {
       commonName = cn;
    }

    public void setSurName (String sn) {
       surName = sn;
    }

    public String getCommonName() {
        return commonName;
    }

    public String getSurName() {
        return surName;
    }

    public static void main(String[] args) {
        LDAP4JavaObjects javaLdapClient = new LDAP4JavaObjects();
        javaLdapClient.setPassword("secret");
        javaLdapClient.setUserID("Alice");

        //Instantiating a Java object.
        MessagingPreferences msgPreferences =
            new MessagingPreferences ();

        //Setting a Java object.
        try {
            javaLdapClient.setJavaObject (msgPreferences);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        javaLdapClient.setJavaObjectCN ("preferences");
        javaLdapClient.writeToServer();
        System.out.println ("Entry stored in ApacheDS..");
    }//main
}
```

## アプリケーションに関する注意点

LDAP4Javaクラスについて最初に気が付くことは、個人のオブジェクト・クラスに属するユーザーのデータ・エントリーを管理するためのメソッドが含まれていることです。setUserName() およびsetPassword() メソッドは、このアプリケーションのユーザー名とパスワードを設定します。

サーバーのアドレスを設定すると、setCommonName() やsetSurName() などのsetterメソッドが使用できるようになります。これらのsetterメソッドによって、ユーザーの属性値が設定されます。setCommonName() メソッドとsetSurName() メソッドに加えて、LDAP4JavaObjectsにはaddUserAttribute() メソッドも含まれています。このaddUserAttribute() メソッドを使用すると、追加のユーザー属性を指定することができます。

属性値を設定すると、LDAP4JavaObjectsでJavaオブジェクトを設定するためのsetJavaObject() メソッドの呼び出しが可能になります。setJavaObject() メソッドを使用すると、Javaオブジェクトの記述や検索を行うサーバー・コンテキストを設定することができます。

LDAP4JavaObjectsのすべての必須データを指定すると、writeToServer() メソッドの呼び出しが可能になります。このメソッドは便利なメソッドです。writeToServer() メソッドは最初に、ユーザーのエントリーがリモート・サーバーに存在するかどうかを確認します。そのエントリーが検出できない場合は、新しいエントリーを記述します。リモート・サーバーにすでにエントリーが存在する場合は、そのエントリーを更新します。

LDAP4JavaObjectsには、ApacheDS内でのユーザー・データまたはJavaオブジェクトの検索に使用できるメソッドsearchFromServer() も用意されています。searchFromServer() メソッドにより、検索操作の完了後、LDAP4JavaObjectsデータ・メンバーが更新されます。

LDAP4JavaObjectsアプリケーションは、ユーザーのアプリケーションのビジネス・ロジックに合わせて拡張することができます。このため、すべての下位レベルのJNDIロジックは、LDAP4JavaObjectsの保護されたメソッドに記述されます。この保護されたメソッドは、LDAP4JavaObjectsを拡張するユーザー各自のクラス内で使用することができます。

リスト11のmain() メソッドを実行すると、これらすべてのLDAP4JavaObjectsメソッドを使用したデモンストレーションを確認することができます。

# まとめ

この2部構成の記事では、ApacheDSによってサーバー側LDAP機能を提供する方法について説明しました。また、ApacheDSのディレクトリー・サービス・アーキテクチャーに関する背景や、ApacheDSがLDAPサーバーとして機能する仕組みの説明に加えて、ApacheDS内でのJavaオブジェクトの保管、検索、取得、更新の方法を示した9つのアプリケーション例も紹介しました。

この記事の完全なソース・コードには、ApacheDS内のJavaオブジェクトと共に使用することができるJNDIコードのサンプルが含まれています。また、これまでに説明したすべての機能を含む再

使用可能なクラスと連動して使用することができるJNDIコードのサンプルも含まれています。このクラスは、ApacheDS内のJavaオブジェクトを操作する練習に使用できるほか、個人用のJavaアプリケーション内で拡張することもできます。

ApacheDSおよびこの記事内で説明した関連技術の詳細については、「参考文献」を参照してください。

---

# 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。