Java におけるネーミングサービスと ディレクトリサービスのためのインタフェース

Code Portability by JNDI™ on Java™ Application Servers

樫 井 一 臣

要 約 最近，普及が加速してきた Java™ アプリケーションサーバでは，多様なリソースを利用する API が充実してきた一方，それに伴う開発者の負担が大きくなってきている．また，コードの移植性も重要なテーマである．ここでは，JNDI（TM）を利用することにより，統一的なインタフェースでリソースにアクセスすることにより，コードの移植性を高め，開発者の負担を軽減する可能性について検証する．

Abstract JNDI（Java Naming and Directory Interface）is a Java programming interface to access various naming services and directory services in a unified method.

After introducing some examples to use LDAP and Distributed Objects, using JNDI with J 2 EE（Java 2 Platform, Enterprise Edition）for Java application servers,this paper shows that JNDI is very effective,especially for the portablity of applications.

1. はじめに

Java が登場してから 4 年が経過し，データベースやトランザクション，また分散オブジェクトなどの多様なリソースにアクセスするための API が充実してきた一方，それらをアプリケーションから統一的に利用する必要性が高まってきている．

特に，アプリケーションサーバにおいては，コアパッケージやスタンダード拡張パッケージに含まれる多種多様な API を利用することになる．例えば，コアパッケージには，JDK 1.1 から含まれている JDBC（Java DataBase Connectivity）によるデータベースへの接続や RMI（Remote Method Invocation）による Java のリモートオブジェクトの利用，また，Java 2 SDK 1.2 から含まれる JavaIDL(Java Interface Definition Language）による CORBA オブジェクトにアクセスするための API がある．さらに，スタンダード拡張パッケージにも，JTA（Java Transaction API）によるトランザクションの利用や，サーバにおける Java コンポーネントである EJB（Enterprise JavaBeans）と，多種多様なリソースを活用できるようになってきた（表 1）．

しかし，アプリケーション開発者にとっては，各リソースの種別ごとに個別の API を利用することが求められるため，負担はだんだん大きくなってきている．

特に，今後のアプリケーション開発の中心となるであろう分散オブジェクト技術においては，名前からオブジェクトを参照するため，ネーミングサービスを利用するのが一般的であるが，統一的なインタフェースでオブジェクトが得られれば，コードの移植性が飛躍的に高まることが期待される．

一方，DNS（Domain Name Service）などのネーミングサービスや LDAP（Lightweight Directory Access Protocol）などのディレクトリサービスについては，JNDI により統一的な API で利用することができるようになっている．

そこで，1999年6月に発表され，12月にリリースされる予定のJ2EE（Java 2 Platform, Enterprise Edition）は，Javaによるサーバ・コンポーネントのフレームワークとなることが期待されているが，上記の多様なリソースへのアクセスにJNDIを採用することにより，開発者の負担を軽減すると同時に，コードの移植性を高め，結果としてアプリケーションの開発効率を大幅に高めることを目標にしている．

ここでは，まず，今まであまり紹介されていないJNDIについて概要を述べ，次に，Javaアプリケーションサーバにおける JNDIの応用により，具体的な例から，コードの移植性を検証することにする．

表 1　Java エンタープライズ API

リソース	API	リソースにアクセスするための主なクラス
ファイルシステム	java.io パッケージ	java.io.File
データベース	JDBC	java.sql.DriverManager
リモートの Java オブジェクト	RMI	java.rmi.Naming
CORBA オブジェクト	JavaIDL	org.omg.CORBA.ORB
トランザクション	JTA	(JNDI 利用)
サーバの Java コンポーネント	EJB	(JNDI 利用)
(各種)ネーミングサービス ディレクトリサービス	JNDI	javax.naming.InitialContext
その他のリソース	個別 API	個別のクラス

2. JNDI（Java Naming and Directory Interface）の概要

2.1 ネーミングサービスとディレクトリサービス

まず，ネーミングサービスとディレクトリサービスについての一般的な概念について簡単に説明する．

2.1.1 ネーミングサービス

ネーミングサービスとは，文字列で表される名前と，特定のオブジェクトとの対応を扱うサービスである．代表的な操作は，登録と参照である．登録は，名前とオブジェクトを対応させることであり，参照は，名前から対応するオブジェクトを得ることである．代表的な例に，DNS（Domain Name Service）やNIS（Network Information Service）などがある．また，分散オブジェクト技術においても，リモートのオブジェクトはアドレスで参照できないため，最初にネーミングサービスにより，オブジェクトのリファレンスを得てから利用するのが一般的である．また，名前空間は，階層構造をもつことができる．代表的な例は，UNIXのファイルシステムである．

2.1.2 ディレクトリサービス

ディレクトリサービスは，上記の基本的なネーミングサービスに加えて，各オブジェクトが属性をもつことができ，属性による検索をサポートする．代表的な例に，LDAP（Lightweight Directory Access Protocol）やNDS（Novell Directory Service）がある．例えば，上記のサービスでは，企業の組織図に対応したディレクトリから，名前や電話番号で従業員を検索することができる．

2.2 JNDI のアーキテクチャ

JNDI は，Java アプリケーションから統一的なインタフェースにより，上記で紹介したような多様なサービスを利用するための Java のプログラミング・インタフェースであり，図 1 のようなアーキテクチャとなっている．

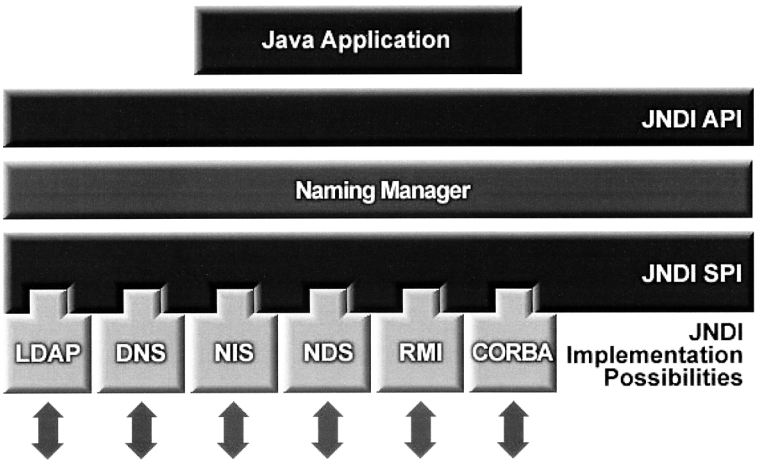

図 1　JNI アーキテクチャ

まず，アプリケーションは，統一的な API により，ネーミング・マネージャにアクセスする．ネーミング・マネージャは，ネーミング・コンテキストにより，利用するサービスに対応したサービス・プロバイダにアクセスすることになる．よって JNDI のインタフェースとクラスを大きく分類すると，アプリケーションが使用する API（Application Programming Interface）と，各サービス・プロバイダが実装するための SPI（Service Provider Interface）の二つからなる．これは，Java テクノロジにおいて，多様なサービスに統一的なインタフェースでアクセスするための一般的なアーキテクチャである．例えば JDK 1.1 から含まれているデータベースに接続するための JDBC（Java DataBase Conncectivity）に詳しい方ならば，JDBC におけるドライバ・マネージャに各種のデータベースに対応した JDBC ドライバをプラグインするアーキテクチャと同様のものであることが理解していただけるであろう．

2.3 パッケージ

JNDI のパッケージは，次の五つからなる．

2.3.1 javax.naming

基本的なネーミングサービスにアクセスするためのインタフェースとクラスからなるパッケージである．

2.3.2 javax.naming.directory

ディレクトリ・サービスの機能を利用するために javax.naming を拡張したパッケージである．

2.3.3 javax.naming.event

イベントを扱うためのインタフェースとクラスからなるパッケージである．

2.3.4 javax.naming.ldap

LDAP バージョン 3 の機能を利用するためのパッケージである．

2.3.5 javax.naming.spi

各種サービスプロバイダが実装するべきインタフェースからなるパッケージである．これを実装することにより，新しいサービスをネーミング・マネージャに結合して，上記の API から利用することが可能となる．

2.4 JNDI ブラウザデモによる例

サン・マイクロシステムズは，JNDI のリファレンス実装に加えて，JNDI ブラウザのデモを提供している．これは，LDAP，ファイルシステム，RMI，CORBA などの複数のサービスを，ネーミングコンテキストを切り替えることでブラウズする GUI アプリケーションであり，JNDI の有用性を示す典型的な例と言えるだろう．図 2 はファイルシステムの例である．

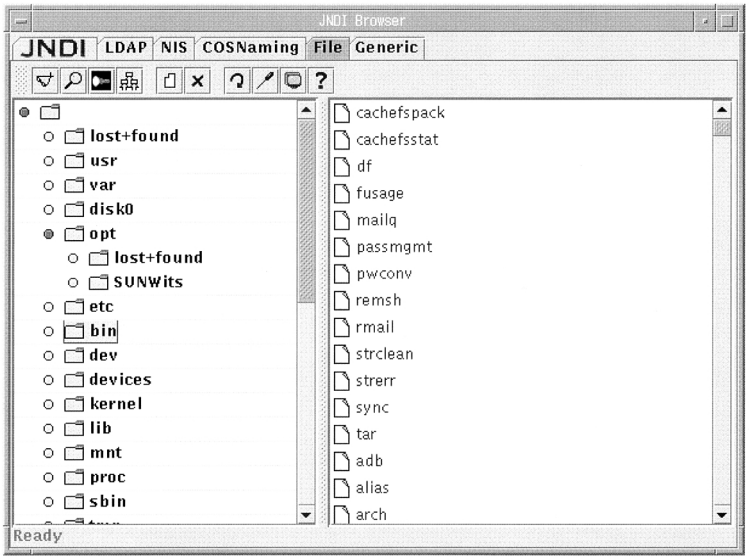

図 2　ファイルシステムの例

また，図 3 は LDAP の例である．

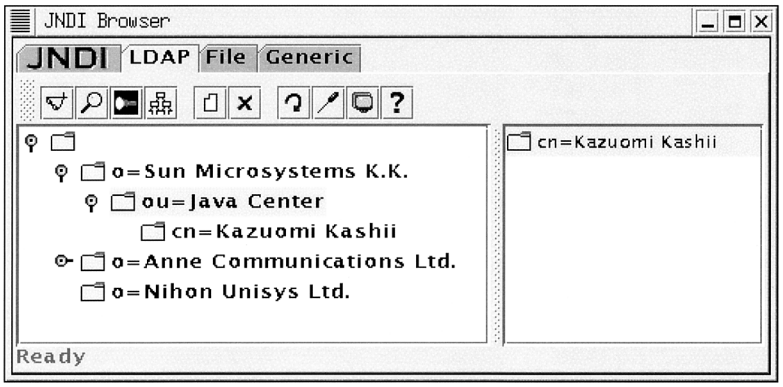

図 3　LDAP の例

図4は，LDAPにおいて，ひとつのディレクトリ・オブジェクトの属性を表示したところである．

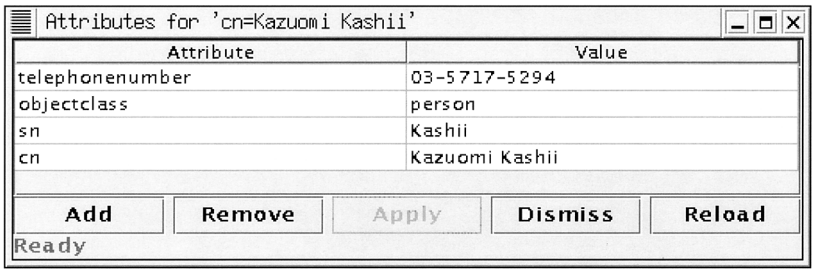

図 4　LDAP の例（ディレクトリオブジェクトの属性表示）

このように，ネーミングコンテキストを変更するだけで，異なるサービスに統一的なインタフェースでアクセスして，情報を表示することが可能である．

3.　Java アプリケーションサーバにおける JNDI の応用

3.1　RMI における具体例

まずは，Java の分散オブジェクト技術である RMI（Remote Method Invocation）を例にとり，コンテキストの獲得から，登録，参照の一連の操作に関して，みてみる．

3.1.1　RMI での登録と検索

RMI のネーミング・サービスを利用するには，java.rmi.Naming のスタティック・メソッドを利用する．

```
Naming.bind("rmi://localhost:1099/hello", new HelloImpl()); ..... (1)

Hello obj = (Hello)Naming.lookup("rmi://remoteserver:1099/hello"); ..... (2)
```

（1）で，“ hello ”という名前で，HelloImpl というリモートオブジェクトのスタブを登録し，（2）は，（1）で登録された RMI のリモートオブジェクトの代理オブジェクトを別のホストで獲得している．

3.1.2　コンテキストの獲得

JNDI を用いるアプリケーションが，最初に行うのはコンテキストの獲得である．プロパティを設定した Hashtable か Properties を用いて，InitialContext のコンストラクタを呼ぶことにより，目的とするサービスに対応したコンテキストが得られる．このとき，最低限，サービスに対応したコンテキストのファクトリクラスの名前と，プロバイダの URL をプロパティを与える．場合によっては，認証に関する情報を与える必要のあることもある．

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jini.rmi.registry.RegistryContextFactory");
env.put(Context.PROVIDER_URL, "rmi://localhost:1099/");
Context ctx = new InitialContext(env); ..... (3)
```

この例では，ローカルホスト上の RMI ネーミングサービスを利用するための，コンテキストを獲得している．Hashtable のかわりに，システムのプロパティを用いてもよい．

3.1.3 登　　録

Context の bind メソッドに，名前とオブジェクトを渡すことにより，新規の対応を登録することができる．

```
ctx.bind("hello", new HelloImpl()); ..... (4)
```

この例では，上で得られた RMI のネーミングサービスに，“ hello ”という名前で，HelloImpl というリモートオブジェクトのスタブを登録している．これは，上記の(1)に対応する．

3.1.4 参　　照

Context の lookup メソッドに，名前を与えることにより，対応するオブジェクトを参照することができる．

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jini.rmi.registry.RegistryContextFactory");
env.put(Context.PROVIDER_URL, "rmi://remoteserver:1099/");
Context ctx = new InitialContext(env);
Hello obj = (Hello)ctx.lookup("hello"); ..... (5)
```

この例では，上で登録された RMI のリモートオブジェクトに対して，別のホストにおいて代理オブジェクトを獲得している．これは，上記の（2）に対応する．

3.1.5 他の ORB への移植性

以上により，通常，java.rmi.Naming クラスのスタティック・メソッドを用いて行っている操作が，JNDI を用いて行えることがわかった．また，CORBA のネーミング・サービスに関しても，同様の方法で JNDI によりアクセスできるため，アプリケーションは，ORB（オブジェクト・リクエスト・ブローカ）に依存する特定のクラスを使わずに記述できることになる．この結果として，移植性の高いコードを書くことができる．

3.2 J 2 EE における JNDI の利用

次に，アプリケーション・サーバ向けの Java 2 プラットフォームである J 2 EE における JNDI の応用例について紹介する．アプリケーション・サーバにおけるリソースには，1）データベース 2）トランザクション 3）EJB コンポーネントなどがある．以下，これらのリソースへの具体的なアクセス方法について述べるが，ここでは，EJB サーバとして，BullSoft 社によるオープンソースの JOnAS（Java Open Application Server）を用いることとする．また，ネーミングサービスには，RMI を用いることとし，以下では，3.1 節の手順により，すでに RMI レジストリのコンテキストが得られているものとする．

```
Context ctx = new InitialContext();
```

3.3 データベース

従来の JDBC によるコネクションを得る一般的なオペレーションは次にようになる．

```
Class.forName("postgresql.Driver"); ..... (1)
Connection conn = DriverManager.getConnection(
                "jdbc:postgresql:hellodb", "postgres", "########"); .... (2)
```

まず，(1) で JDBC ドライバのクラスをロードし，次に，(2) で接続するデータベースの URL，ユーザ，パスワードを与えて，コネクション・オブジェクトを得ている．この例では，データベースとして，オープンソースの PostgreSQL を用いている．一方，J 2 EE のベースとなる JDBC 2 では，データソースの概念を導入し，JNDI によりアクセスすることにより，コードの汎用性を高めることができる．特に，永続性をもつ EJB コンポーネントである EntityBean においては，デプロイメント・ディスクリプタの環境プロパティで，データソースに関する設定を行うことにより，ソフトウェア部品としての EJB コンポーネントの汎用性が高まることになる．JOnAS では，JDBC 用のプロパティをテキストで記述し，デプロイメント・ツールを用いることにより，Bean の中では，つぎのようなコードで，データベースのコネクションを得ることができる．

```
DataSource ds = (DataSource) ctx.lookup("jdbc_hello"); ..... (3)
Connection conn = ds.getConnection(); ..... (4)
```

3.4 トランザクション

EJB コンポーネントにおいては，メソッドに対しる属性の設定により，トランザクションを扱うことができる．ここでは，クライアントが，トランザクションを開始する場合のコードを紹介する．

```
UserTransaction tx = (UserTransaction)ctx.lookup( "javax.transaction.UserTransaction"); ..... (5)
tx.begin();
 try {
    ...
      tx.commit();
 } catch (...Exception e) {
    tx.abort();
}
```

3.5 EJB コンポーネントのデプロイメント

デプロイメント・ディスクリプタにおいて，コンポーネントの名前を与える．クライアントは，この名前により EJB コンポーネントのホームオブジェクトを得ることができる．ホームオブジェクトが得られれば，Factory メソッドや Finder メソッド（EntityBean の場合）によって，目的とするリモートオブジェクトを得ることができる．

```
HelloHome home = (HelloHome)ctx.lookup("HelloImplHome"); ..... (6)
Hello obj = home.create(); ..... (7)
```

3.6 JNDI ブラウザによる確認

通常，EJB サーバはモニタリング用のツールを提供しているが，ここでは 6 章で述べた JNDI ブラウザを利用して，リソースの状況を見てみる．今回の JOnAS では，ネーミングサービスとしては，RMI レジストリを用いているので，5 章と同様の INITIAL_CONTEXT_FACTORY と PROVIDER_URL を指定して，ブラウザを立ち上げると，図 5 のようになる．

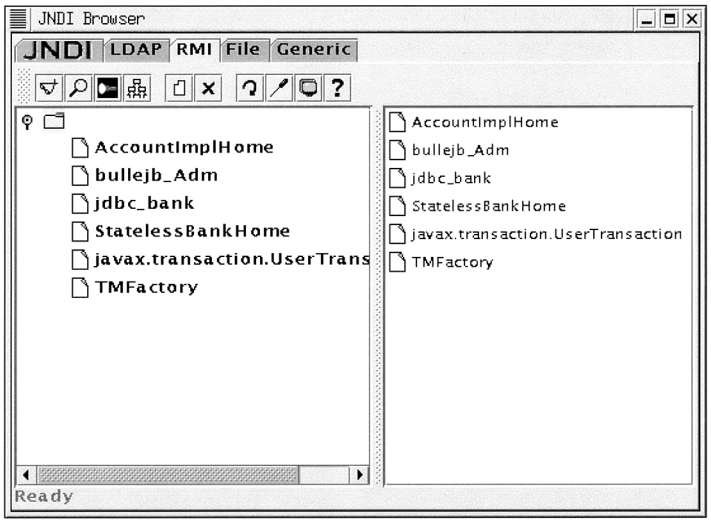

図 5　JNDI ブラウザ

これは，銀行の預金口座を扱うサンプルアプリケーションの例であるが，上から順に，

1）預金口座の EntityBean のホームオブジェクト
2）JOnAS の管理オブジェクト
3）JDBC のデータソース
4）ビジネスロジックをもつ SessionBean のホームオブジェクト
5）ユーザトランザクション
6）トランザクションマネージャのファクトリ

である．ここで，上から 3 番目の jdbc_bank をクリックすると，図 6 のようなデータベースに対応したデータソースの情報が得られる．

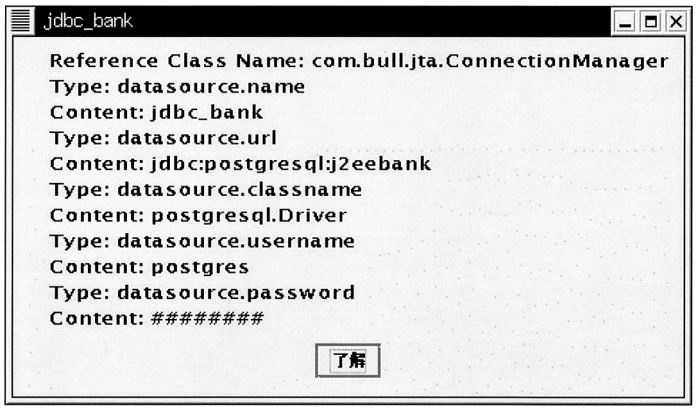

図 6　データソースの情報

図 7 は，下から 2 番目のユーザ・トランザクションの情報である．

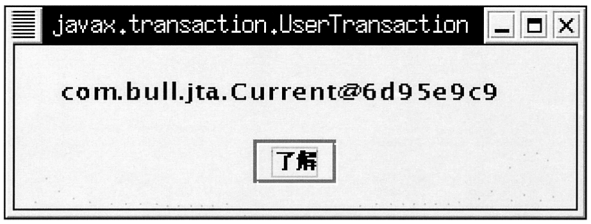

図 7　ユーザ・トランザクションの情報

その他，図 5 に現れている EJB コンポーネントのホーム・オブジェクトなどについても，同様に情報を得ることができる．これは，上で記述したように，同一の API により，オブジェクトの参照を得られることを意味しており，アプリケーション開発者にとっては，オブジェクトの獲得では，多様な API の使用から解放されることになる．

4. お わ り に

以上，見てきたように，JNDI により多様なリソースに，統一的な API でアクセスでき，移植性の高いアプリケーションが記述できることがわかった．これにより，特にアプリケーション・サーバにおいては，J 2 EE をプラットフォームとする汎用的なソフトウェア・コンポーネントが作成できることになり，アプリケーションの開発効率が飛躍的に高まることであろう．また，最新の Java 2 SDK Standard Edition 1.3 においても，コアの API として採用される予定であり，今後ますます重要な API として多方面で利用されることであろう．

参考文献　［1］ http : //java.sun.com/products/jndi/
［2］ http : //www.openmaster.com/ejb/

執筆者紹介　樫　井　一　臣（Kazuomi Kashii）

1958 年生．1981 年東京工業大学理学部数学科卒業．富士通 FIP，ソニーテクトロニクスを経て，1987 年サンマイクロシステムズ(株)入社．システムエンジニアとして，ソフトウェア開発環境, 分散オブジェクトなどをサポート．現在は，Java センターにて，Java テクノロジ全般，特に EJB（Enterprise JavaBeans）を中心とした J 2 EE や Jini に関するサポートを中心に活動している．