JNDI - Java Naming and Directory Interface

Naming Concept:

A fundamental facility in any computing system is the *naming service*--the means by which names are associated with objects and objects are found based on their names. A naming service allows you to look up an object given its name.

Names

To lookup an object in a naming system, you supply it the *name* of the object. The naming system determines the syntax that the name must follow. This syntax is sometimes called the naming system's *naming convention*.

Bindings

The association of a name with an object is called a *binding*. For example, a file name is *bound* to a file.

Context

A *context* is a set of name-to-object bindings. Every context has an associated naming convention. A context provides a lookup (*resolution*) operation that returns the object and may provide operations such as those for binding names, unbinding names, and listing bound names.

Naming Systems

A *naming system* is a connected set of contexts of the same type (they have the same naming convention) and provides a common set of operations.

Namespaces

A *namespace* is the set of names in a naming system.

Directory Concepts

Many naming services are extended with a *directory service*. A directory service associates names with objects and also allows such objects to have *attributes*. Thus, you not

only can look up an object by its name but also get the object's attributes or search for the object based on its attributes.

Attributes

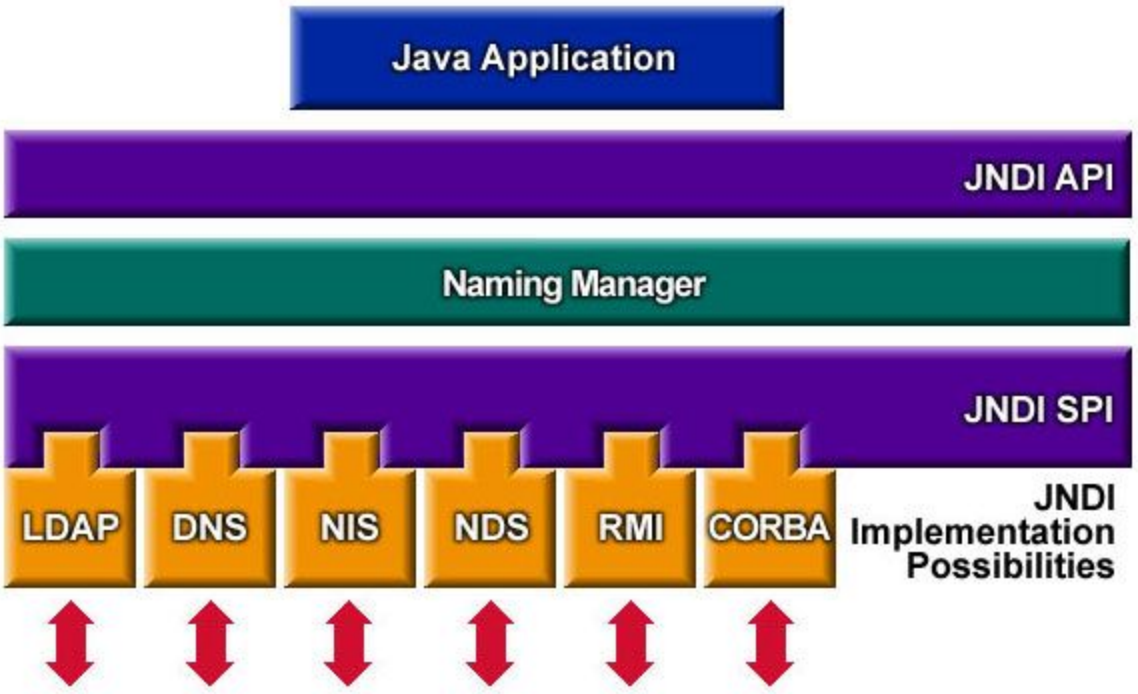
A directory object can have *attributes*. For example, a printer might be represented by a directory object that has as attributes its speed, resolution, and color.

Combining Naming and Directory Services

Directories often arrange their objects in a hierarchy. For example, the LDAP arranges all directory objects in a tree, called a *directory information tree (DIT)*. Within the DIT, an organization object, for example, might contain group objects that might in turn contain person objects. When directory objects are arranged in this way, they play the role of naming contexts in addition to that of containers of attributes.

JNDI Overview

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) that provides naming and directory functionality to applications written using the Java programming language.



JNDI Tomcat Configuration:

DataSource is first important object you configure with JNDI. Configure the JNDI DataSource in Tomcat by adding a declaration for your resource to your context (tomcat/conf/context.xml)

For example:

```
<Context>
   <!-- maxTotal: Maximum number of database connections in pool. Make sure you
        configure your mysqld max_connections large enough to handle
        all of your db connections. Set to -1 for no limit.
        -->
   <!-- maxIdle: Maximum number of idle database connections to retain in pool.
        Set to -1 for no limit.  See also the DBCP documentation on this
        and the minEvictableIdleTimeMillis configuration parameter.
        -->
   <!-- maxWaitMillis: Maximum time to wait for a database connection to become available
        in ms, in this example 10 seconds. An Exception is thrown if
        this timeout is exceeded.  Set to -1 to wait indefinitely.
        -->
   <!-- username and password: MySQL username and password for database connections  -->
   <!-- driverClassName: Class name for the old mm.mysql JDBC driver is
        org.gjt.mm.mysql.Driver - we recommend using Connector/J though.
        Class name for the official MySQL Connector/J driver is com.mysql.jdbc.Driver.
        -->
   <!-- url: The JDBC connection url for connecting to your MySQL database.
        -->

  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
            maxTotal="100" maxIdle="30" maxWaitMillis="10000"
            username="javauser" password="javadude" driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/javatest"/>

</Context>
```

Oracle requires minimal changes from the MySQL configuration except for the usual gotchas :-)

```
<Resource name="jdbc/myoracle" auth="Container"
          type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
          url="jdbc:oracle:thin:@127.0.0.1:1521:mysid"
          username="scott" password="tiger" maxTotal="20" maxIdle="10"
          maxWaitMillis="-1"/>
```

How to read JNDI Object from Context in Java Code:

```
Context initContext = new InitialContext();
Context envContext  = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/myoracle");
Connection conn = ds.getConnection();
```

Environment Entries:

You can configure named values that will be made visible to the web application as environment entry resources, by nesting `<Environment>` entries inside this element. For example, you can create an environment entry like this:

```
<Context>
  ...
  <Environment name="maxExemptions" value="10" type="java.lang.Integer" override="false"/>
  ...
</Context>
```

Attribute	Description
description	Optional, human-readable description of this environment entry.
name	The name of the environment entry to be created, relative to the **java:comp/env** context.
override	Set this to false if you do not want an **<env-entry>** for the same environment entry name, found in the web application deployment descriptor, to override the value specified here. By default, overrides are allowed.
type	The fully qualified Java class name expected by the web application for this environment entry. Must be a legal value for **<env-entry-type>** in the web application deployment descriptor.
value	The parameter value that will be presented to the application when requested from the JNDI context. This value must be convertable to the Java type defined by the type attribute.

ActiveMQ Setup in Tomcat JNDI

```
<Resource
        name="jms/ConnectionFactory"
        auth="Container"
        type="org.apache.activemq.ActiveMQConnectionFactory"
        description="JMS Connection Factory"
        factory="org.apache.activemq.jndi.JNDIReferenceFactory"
        brokerURL="tcp://localhost:61616"
        brokerName="LocalActiveMQBroker"
        useEmbeddedBroker="false"/>

   <Resource name="jms/topic/MyTopic"
        auth="Container"
        type="org.apache.activemq.command.ActiveMQTopic"
        factory="org.apache.activemq.jndi.JNDIReferenceFactory"
        physicalName="MY.TEST.FOO"/>

<Resource name="jms/queue/MyQueue"
        auth="Container"
      type="org.apache.activemq.command.ActiveMQQueue"
      factory="org.apache.activemq.jndi.JNDIReferenceFactory"
```

```
    physicalName="MY.TEST.FOO.QUEUE"/>
```

This will setup the JNDI for the ConectionFactory and Topic to work within Tomcat.
Here is some example code that will publish a test message to the MY.TEST.FOO Topic:

```
try {
   InitialContext initCtx = new InitialContext();
   Context envContext = (Context) initCtx.lookup("java:comp/env");
   ConnectionFactory connectionFactory = (ConnectionFactory)
envContext.lookup("jms/ConnectionFactory");
   Connection connection = connectionFactory.createConnection();
   Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
   MessageProducer producer = session.createProducer((Destination)
envContext.lookup("jms/topic/MyTopic"));

   Message testMessage = session.createMessage();
   testMessage.setStringProperty("testKey", "testValue");
   producer.send(testMessage);
} catch (NamingException e) {
   // TODO handle exception
} catch (JMSException e) {
   // TODO handle exception
}
```