Overview of JNDI

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. It is defined to be independent of any specific directory service implementation. Thus a variety of directories -new, emerging, and already deployed can be accessed in a common way.

Naming Package Context Interface:

The javax.naming package defines a Context interface, which is the core interface for looking up, binding/unbinding, renaming objects and creating and destroying subcontexts.

Lookup

The most commonly used operation is lookup(). You supply lookup() the name of the object you want to look up, and it returns the object bound to that name.

Bindings

listBindings() returns an enumeration of name-to-object bindings. A binding is a tuple containing the name of the bound object, the name of the object's class, and the object itself.

List

list() is similar to listBindings(), except that it returns an enumeration of names containing an object's name and the name of the object's class. list() is useful for applications such as browsers that want to discover information about the objects bound within a context but that don't need all of the actual objects. Although listBindings() provides all of the same information, it is potentially a much more expensive operation.

Name

Name is an interface that represents a generic name--an ordered sequence of zero or more components. The Naming Systems use this interface to define the names that follow its conventions as described in the Naming and Directory Concepts lesson.

References

Objects are stored in naming and directory services in different ways. A reference might be a very compact representation of an object.

The Initial Context

In the JNDI, all naming and directory operations are performed relative to a context. There are no absolute roots. Therefore the JNDI defines an InitialContext, which provides a starting point for naming and directory operations. Once you have an initial context, you can use it to look up other contexts and objects.

Exceptions

The JNDI defines a class hierarchy for exceptions that can be thrown in the course of performing naming and directory operations. The root of this class hierarchy is NamingException. Programs interested in dealing with a particular exception can catch the corresponding subclass of the exception. Otherwise, they should catch NamingException.

JNDI with Tomcat:

web.xml configuration

The following elements may be used in the web application deployment descriptor (`/WEB-INF/web.xml`) of your web application to define resources:

- **`<env-entry>`** - Environment entry, a single-value parameter that can be used to configure how the application will operate.
- **`<resource-ref>`** - Resource reference, which is typically to an object factory for resources such as a JDBC `DataSource`, a JavaMail `Session`, or custom object factories configured into Tomcat.
- **`<resource-env-ref>`** - Resource environment reference, a new variation of `resource-ref` added in Servlet 2.4 that is simpler to configure for resources that do not require authentication information.

Providing that Tomcat is able to identify an appropriate resource factory to use to create the resource and that no further configuration information is required, Tomcat will use the information in `/WEB-INF/web.xml` to create the resource.

context.xml configuration

Tomcat specific resource configuration is entered in the `<Context>` elements that can be specified in either `$CATALINA_BASE/conf/server.xml` or, preferably, the per-web-application context XML file (`META-INF/context.xml`).

Tomcat specific resource configuration is performed using the following elements in the `<Context>` element:

- <Environment> - Configure names and values for scalar environment entries that will be exposed to the web application through the

JNDI `InitialContext` (equivalent to the inclusion of an `<env-entry>` element in the web application deployment descriptor).

- <Resource> - Configure the name and data type of a resource made available to the application (equivalent to the inclusion of a `<resource-ref>` element in the web application deployment descriptor).
- <ResourceLink> - Add a link to a resource defined in the global JNDI context. Use resource links to give a web application access to a resource defined in the <GlobalNamingResources> child element of the <Server> element.
- <Transaction> - Add a resource factory for instantiating the UserTransaction object instance that is available at `java:comp/UserTransaction`.

Any number of these elements may be nested inside a `<Context>` element and will be associated only with that particular web application.

If a resource has been defined in a `<Context>` element it is not necessary for that resource to be defined in `/WEB-INF/web.xml`. However, it is recommended to keep the entry in `/WEB-INF/web.xml` to document the resource requirements for the web application.

Using resources

The `InitialContext` is configured as a web application is initially deployed, and is made available to web application components (for read-only access). All configured entries and resources are placed in the `java:comp/env` portion of the JNDI namespace, so a typical access to a resource - in this case, to a JDBC `DataSource` - would look something like this:

```
// Obtain our environment naming context
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");

// Look up our data source
DataSource ds = (DataSource)
  envCtx.lookup("jdbc/EmployeeDB");

// Allocate and use a connection from the pool
Connection conn = ds.getConnection();
... use this connection to access the database ...
conn.close();
```

MySQL Database Entry:

```
<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"/>
```

ActiveMQ JNDI Entry:

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


<Resource
        name="jms/someTopic"
        auth="Container"
        type="org.apache.activemq.command.ActiveMQTopic"
        description="my Topic"
       factory="org.apache.activemq.jndi.JNDIReferenceFactory"
        physicalName="FOO.BAR"/>

<Resource
        name="jms/aQueue"
        auth="Container"
        type="org.apache.activemq.command.ActiveMQQueue"
        description="my Queue"
       factory="org.apache.activemq.jndi.JNDIReferenceFactory"
        physicalName="FOO.BAR"/>
```