JNDI

Java Naming and Directory Interface

See also:
http://java.sun.com/products/jndi/tutorial/trailmap.html

Naming service

A naming service is an entity that

•associates names with objects.We call this *binding* names to objects. *This is similar to a telephone company 's associating a person 's name with a specific residence 's telephone number*

•provides a facility to find an object based on a name.We call this *looking up* or *searching* for an object.*This is similar to a telephone operator finding a person 's telephone number based on that person 's name and connecting the two people.*

In general,a naming service can be used to find any kind of generic object, like a file handle on your hard drive or a printer located across the network.

Directory service

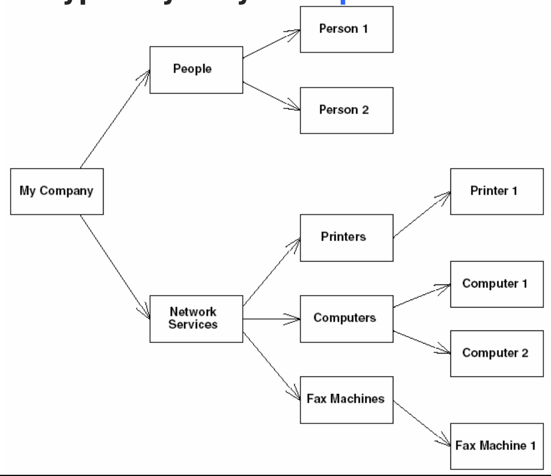
A *directory object* differs from a generic object because you can store *attributes* with directory objects. *For example,you can use a directory object to represent a user in your company.You can store information about that user,like the user 's password,as attributes in the directory object.*

A *directory service* is a naming service that has been extended and enhanced to provide directory object operations for manipulating attributes.

A *directory* is a system of directory objects that are all connected. *Some examples of directory products are Netscape Directory Server and Microsoft 's Active Directory.*

Directory service

Directories are similar to DataBases, except that they typically are organized in a hierarchical tree-like structure. Typically they are optimized for reading.



Directory service

A ***directory object*** **differs from a generic object because you can store *attributes* with directory objects.** *For example,you can use a directory object to represent a user in your company.You can store information about that user,like the user 's password,as attributes in the directory object.*

A ***directory service*** **is a naming service that has been extended and enhanced to provide directory object operations for manipulating attributes.**

A ***directory*** **is a system of directory objects that are all connected.** *Some examples of directory products are Netscape Directory Server and Microsoft 's Active Directory.*

Examples of Directory services

Netscape Directory Server

Microsoft 's Active Directory

Lotus Notes (IBM)

NIS (Network Information System) by Sun

NDS (Network Directory Service) by Novell

LDAP (Lightweight Directory Access Protocol)

JNDI concepts

JNDI is a system for Java-based clients to interact with naming and directory systems. JNDI is a bridge over naming and directory services, that provides one common interface to disparate directories.

Users who need to access an LDAP directory use the same API as users who want to access an NIS directory or Novell's directory. All directory operations are done through the JNDI interface, providing a common framework.

JNDI advantages

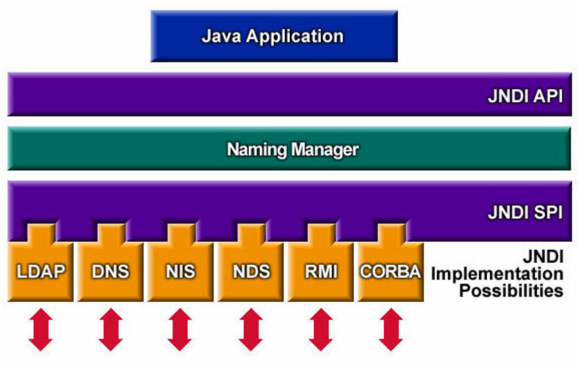
-You only need to learn a single API to access all sorts of directory service information, such as security credentials, phone numbers, electronic and postal mail addresses, application preferences, network addresses, machine configurations, and more.

-JNDI insulates the application from protocol and implementation details.

-You can use JNDI to read and write whole Java objects from directories.

- You can link different types of directories, such as an LDAP directory with an NDS directory, and have the combination appear to be one large, federated directory.

JNDI Architecture



The JNDI homepage
http://java.sun.com/products/jndi
has a list of service providers.

JNDI concepts

*An **atomic name** is a simple,basic,indivisible component of a name.For example,in the string /etc/fstab ,etc and fstab are atomic names.*

*A **binding** is an association of a name with an object.*

*A **context** is an object that contains zero or more bindings. Each binding has a distinct atomic name. Each of the mtab and exports atomic names is bound to a file on the hard disk.*

*A **compound name** is zero or more atomic names put together. e.g. the entire string /etc/fstab is a compound name. Note that a compound name consists of multiple bindings.*

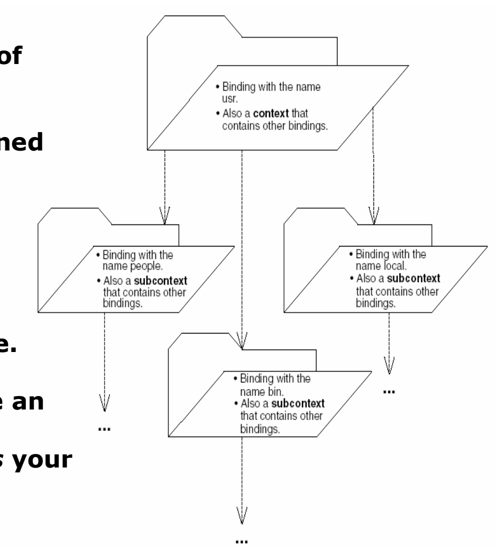
Contexts and Subcontexts

A *naming system* is a connected set of contexts.

A *namespace* is all the names contained within naming system.

The starting point of exploring a namespace is called an *initial context*. An initial context is the first context you happen to use.

To acquire an initial context, you use an *initial context factory*. An initial context factory basically *is* your JNDI driver.


• Binding with the name usr.
• Also a context that contains other bindings.
• Binding with the name people.
• Also a subcontext that contains other bindings.
• Binding with the name local.
• Also a subcontext that contains other bindings.
• Binding with the name bin.
• Also a subcontext that contains other bindings.
...
...
...

Acquiring an initial context

When you acquire an initial context, you must supply the necessary information for JNDI to acquire that initial context.

For example, if you're trying to access a JNDI implementation that runs within a given server, you might supply:
- The IP address of the server
- The port number that the server accepts
- The starting location within the JNDI tree
- Any username/password necessary to use the server

Acquiring an initial context

```
package examples;

public class InitCtx {
  public static void main(String args[]) throws Exception {
    // Form an Initial Context
    javax.naming.Context ctx =
        new javax.naming.InitialContext();
    System.err.println("Success!");
    Object result = ctx.lookup("PermissionManager");
  }
}
```

```
java
-Djava.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
-Djava.naming.provider.url=jnp://193.205.194.162:1099
-Djava.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
examples.InitCtx
```

Another example

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.Hashtable;
class Lookup {
  public static void main(String[] args) {
   // Check that user has supplied name of file to lookup
   if (args.length != 1) {
     System.err.println("usage: java Lookup <filename>");
     System.exit(-1);
   }
   String name = args[0];
   // Identify service provider to use
   Hashtable env = new Hashtable(11);
   env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.sun.jndi.fscontext.RefFSContextFactory");
```

```
   try {
     // Create the initial context
     Context ctx = new InitialContext(env);
     // Look up an object
     Object obj = ctx.lookup(name);
     // Print it out
     System.out.println(name +
                     " is bound to: " + obj);
     // Close the context when we're done
     ctx.close();
   } catch (NamingException e) {
     System.err.println("Problem looking up "
                              + name + ": " + e);
   }
  }
}
```

LDAP example

```
package jndiaccesstoldap;
import javax.naming.Context;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.DirContext;
import javax.naming.directory.Attributes;
import javax.naming.NamingException;
import java.util.Hashtable;
public class Getattr {
   public static void main(String[] args) {
      // Identify service provider to use
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.sun.jndi.ldap.LdapCtxFactory");
   //env.put(Context.PROVIDER_URL, "ldap://ldap.unitn.it:389/o=JNDITutorial");
   env.put(Context.PROVIDER_URL, "ldap://ldap.unitn.it:389/o=personale");
```

```
   try {
      // Create the initial directory context
      DirContext ctx = new InitialDirContext(env);

      // Ask for all attributes of the object
      Attributes attrs = ctx.getAttributes("cn=Ronchetti
   Marco");

      // Find the surname ("sn") and print it
      System.out.println("sn: " + attrs.get("sn").get());

      // Close the context when we're done
      ctx.close();
    } catch (NamingException e) {
      System.err.println("Problem getting attribute: " + e);
}}}
```

Operations on a JNDI context

list() *retrieves a list of contents available at the current context.This typically includes names of objects bound to the JNDI tree,as well as subcontexts.*

lookup() *moves from one context to another context,such as going from c:\ to c:\windows. You can also use lookup()to look up objects bound to the JNDI tree.The return type of lookup()is JNDI driver specific.*

rename() *gives a context a new name*

Operations on a JNDI context

createSubcontext()*creates a subcontext from the current context,such as creating c:\foo \bar from the folder c:\foo.*

destroySubcontext()*destroys a subcontext from the current context,such as destroying c:\foo \bar from the folder c:\foo.*

bind()*writes something to the JNDI tree at the current context.As with lookup(),JNDI drivers accept different parameters to bind().*

rebind()*is the same operation as bind,except it forces a bind even if there is already something in the JNDI tree with the same name.*