Introduction to JNDI and Naming Services

Naming Services

- Naming Services provide a human-friendly mapping of names to machine-friendly items or objects.
- Naming and directory services play a vital role in intranets and the Internet by providing network-wide sharing of a variety of information about users, machines, networks, services, and applications.

Naming Services

- Finding resources is very important in large-scale enterprise applications environment. Usually we build applications depend on services provided by other applications
- A naming service maintains a set of bindings. Bindings relate names to objects.
- All objects in a naming system are named in the same way(same naming convention). Clients use the naming service to locate objects by name.

Naming Services Implementation

- COS (Common Object Services) Naming: The naming service for CORBA applications; allows applications to store and access references to CORBA objects.
- DNS (Domain Name System): The Internet's naming service, it maps human-friendly names (e.g. www.sun.com) into computer-friendly IP addresses. DNS is a distributed naming service, the service and its underlying database is spread across many hosts known as DNS servers on the Internet.

Naming Services Implementation

- LDAP (Lightweight Directory Access Protocol): Developed by the University of Michigan; it is a lightweight version of DAP (Directory Access Protocol), and is part of X.500, a standard for network directory services. It is common infrastructure widely adopted in enterprise computing
- NIS (Network Information System) and NIS+: Network naming services developed by Sun Microsystems. Both allow users to access files and applications on any host with a single ID and password.

Common features of Naming Services

- All naming services implementation bind names to objects and provide look up services of objects by name.
- Many naming systems do not store objects directly. Instead, they store references to objects. For example, In the DNS case. The address 205.18.172.30 is a reference to a computer's location on the Internet, not the computer itself.

Differences of Naming Services Implementation

- Each naming service requires different naming convention.
- DNS, names are built from components that are separated by dots ("."). The name "www.sun.com" names a machine called "www" in the "sun.com" domain.
- In LDAP, Names are built from components that are separated by commas. Components in an LDAP name must be specified as name/value pairs. Example "cn=May Chan,ou=employees,dc=SUN,dc=com "

What is JNDI

- JNDI provides a common interface / abstraction to a variety of existing naming services include DNS, LDAP, Active Directory, RMI registry, COS registry, NIS, and file systems.
- JNDI is an API specified in Java technology that provides naming and directory functionality to applications written in the Java programming language.
- JNDI is a standard Java API that is bundled with JDK1.3 and higher.

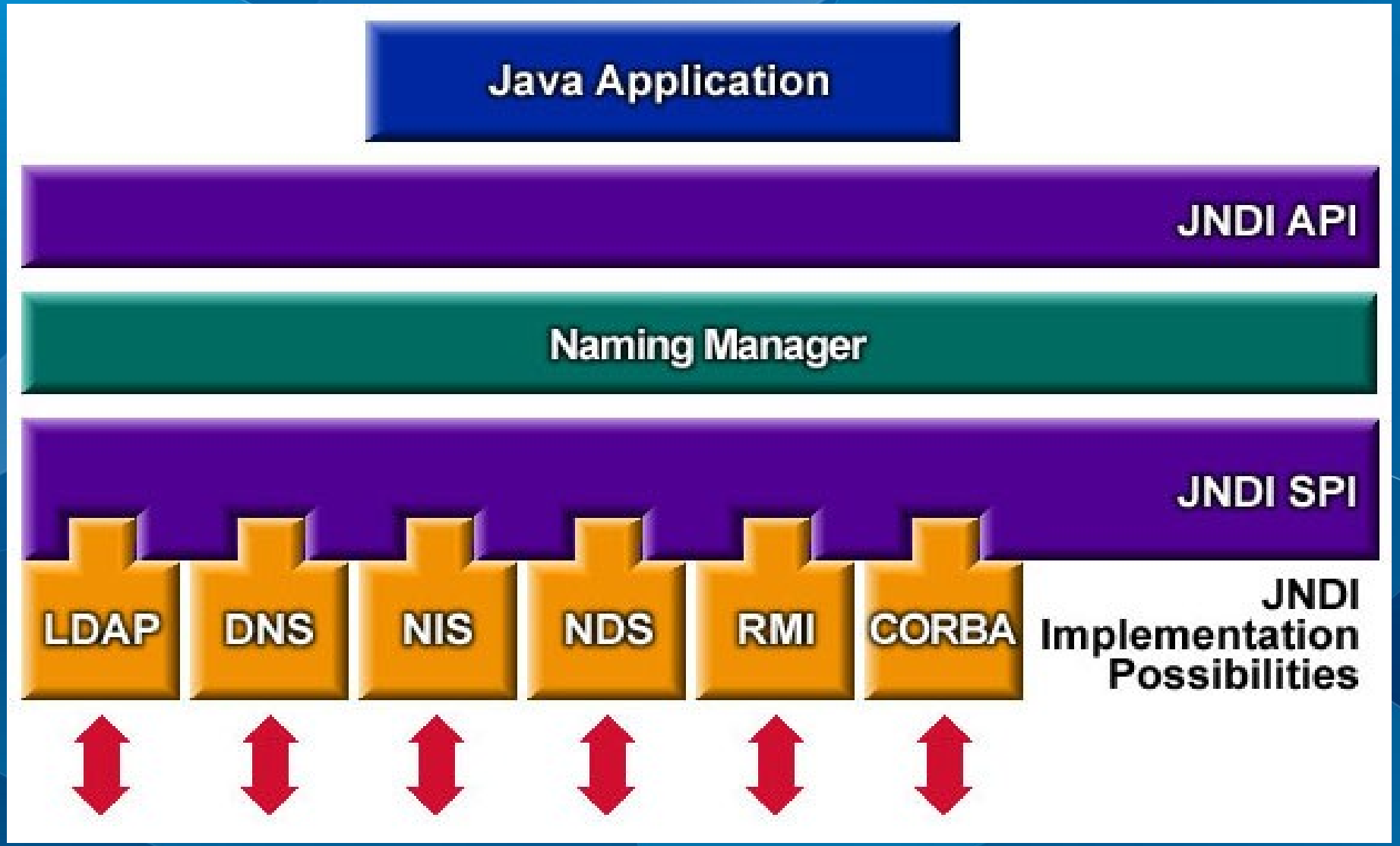
JNDI overview

- JNDI must provide a workable abstraction that gets around differences among naming services to provide Location Transparent
- The JNDI API is divided logically into a client API that is used to access naming services, and a service provider interface (SPI) that allows the user to create JNDI implementations for naming services.

JNDI Overview

- The SPI layer is an abstraction that naming service providers must implement to enable the core JNDI classes to expose the naming service using the common JNDI client interface. An implementation of JNDI for a naming service is referred to as a JNDI provider.

JNDI overview



JNDI usage

- Using JNDI, applications based on Java technology can store and retrieve named Java objects of any type.
- In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

JNDI usage

- JNDI solves problems with the Name class and its subclasses and helper classes. The Name class represents a name composed of an ordered sequences of subnames, and provides methods for working with names independent of the underlying naming service
- JNDI use hierarchical namespace, it usually represent as strings. e.g. "China/HongKong/Wanchai"

JNDI usage

- Nodes in the Naming tree are known as Contexts. That is to say a context represents a set of bindings within a naming service that all share the same naming convention
- The format just like the directories in a filesystem.
- Objects can be bound into the Contexts
- The same object may be bound into multiple contxts.

Context

- The Context interface plays a central role in JNDI, it provide methods for manipulating JNDI trees. The most common usage is lookup and bind.
- A Context object provides the methods for binding names to objects and unbinding names from objects, for renaming objects, and for listing the bindings.
- In Java EE most of the binding of objects is done by the server.

InitialContext

- Some naming services also provide subcontext functionality. In this case, the Context class also provides methods for creating and destroying subcontexts.
- JNDI performs all naming operations relative to a context. To assist in finding a place to start, the JNDI specification defines an InitialContext class, it just like the root directory in a filesystem.
- The InitialContext class implement Context interface
- It provide Further navigation of the namespace is performed relative to it.

InitialContext

```
Import javax.naming.*;
...
//Obtain the initial context
Context ctx = new InitialContxt();
//Locate object relative to Initial Context
Object o = ctx .lookup("ObjectName")
...
```

Context's methods

- void bind(String stringName, Object object): Binds a name to an object. The name must not be bound to another object. All intermediate contexts must already exist.
- void rebind(String stringName, Object object): Binds a name to an object. All intermediate contexts must already exist.
- Object lookup(String stringName): Returns the specified object.
- void unbind(String stringName): Unbinds the specified object.
- The Context interface also provides methods for renaming and listing bindings.
- void rename(String stringOldName, String stringNewName): Changes the name to which an object is bound.
- NamingEnumeration listBindings(String stringName): Returns an enumeration containing the names bound to the specified context, along with the objects and the class names of the objects bound to them.
- NamingEnumeration list(String stringName): Returns an enumeration containing the names bound to the specified context, along with the class

JNDI 1.2 Service Providers

- LDAP 1.2.4 maintenance release of Sun's LDAP service provider, supporting versions 2 and 3 of the LDAP protocol.
- COS Naming 1.2.1 maintenance release of Sun's service provider for COS Naming, providing access to CORBA naming services through the standard JNDI interface.
- RMI Registry 1.2.1 maintenance release of Sun's service provider for the RMI Registry.

JNDI 1.2 Service Providers

- NIS 1.2.1 maintenance release of Sun's service provider for NIS, the Network Information Service (formerly known as YP).
- DSML v1 1.2 FCS release of Sun's service provider for the Directory Services Markup Language (DSML) v1.0.
- DSML v2 Early access release of Sun's service providers for the Directory Services Markup Language (DSML) v2.0.
- DNS 1.2 FCS release of Sun's service provider for the Domain Name System

JNDI 1.2 Service Providers

- File System 1.2 Beta 3 release of Sun's service provider for accessing the file system.
- MirrorJNDI Smardec's open source (LGPL) JNDI service provider for accessing arbitrary Java objects in a hierarchical namespace. Allows to create full copy of existing service provider's data, serialize it (XML or binary) and then work with local copy.

JNDI 1.2 Service Providers

- Novell For access to the industry-leading directory service NDS, and to NetWare 3X's Bindery, the Novell File Systems, and other Novell services such as Extended NCP, etc.
- Windows Registry Cogent Logic Corporation's JNDI service provider for accessing the Windows registry on Windows XP/2000/NT/Me/9x.
- JNDI2R Scand's JNDI service provider for accessing the Windows registry on Windows XP/2000/NT/Me/9x.

JNDI 1.2 Service Providers

- XNam Ervacon's JNDI service provider that reads naming information from XML files using the JSR-57 schema (long-term JavaBeans persistence).
- Stibium Provides JNDI naming and directory services, based on an xml storage and persistence model, that are accessible via SOAP xml web-services.

Overview of Interfaces

- The Naming Interface – javax.naming: Context is the core interface that specifies a naming context. It defines basic operations such as adding a name-to-object binding, looking up the object bound to a specified name, listing the bindings, removing a name-to-object binding, crea ting and destroying subcontexts of the same type, etc.

```
Printer printer = (Printer)
 ctx.lookup("treekiller");
printer.print(report);
```

- The application is not exposed to any naming service implementation. In fact, a new type of naming service can be introduced without requiring the application to be modified or even disrupted if it is running.

Overviews of interfaces

- The Event Interface – javax.naming.event
- Naming Events. The NamingEvent class represents an event generated by a naming or directory service. Examples of a NamingEvent are a change to a directory entry's attribute or a rename of a directory entry.
- Naming Listeners. A NamingListener is an object that registers interests in NamingEvents. Listeners register with a context to receive notification of changes in the context, its children, or its subtree.

Overview of interfaces

- The LDAP Interface - javax.naming.ldap
- The LdapContext interface allows an application to use LDAP v3-specific features, including extensions and controls, not already covered by the more generic DirContext interface.

Overview of Interfaces

- The Service Provider Interface – javax.naming.spi
- The JNDI SPI provides the means by which different naming/directory service providers can develop and hook up their implementations so that the corresponding services are accessible from applications that use JNDI.
- JNDI allows specification of names that span multiple namespaces, if one service provider implementation needs to interact with another in order to complete an operation, the SPI provides methods that allow different provider implementations to cooperate to complete client JNDI operations.

Java EE with JNDI

- A resource object and its JNDI name are bound together by the naming and directory service, which is included with the Application Server.
- A JNDI name is a people-friendly name for an object. Because Java EE components access this service through the JNDI API, the object usually uses its JNDI name. e.g. the JNDI name of the Java DB database is jdbc/derby. When it starts up, the Application Server reads information from the configuration file and automatically adds JNDI database names to the name space.

Java EE with JNDI

- A Java EE container implements the application component's environment, and provides it to the application component instance as a JNDI naming context
- The application component's business methods access the environment using the JNDI interfaces. The application component provider declares in the deployment descriptor all the environment entries that the application component expects to be provided in its environment at runtime

Java EE with JNDI

- The container provides an implementation of the JNDI naming context that stores the application component environment. The container also provides the tools that allow the deployer to create and manage the environment of each application component.
- A deployer uses the tools provided by the container to initialize the environment entries that are declared in the application component's deployment descriptor. The deployer sets and modifies the values of the environment entries.

Java EE with JNDI

- The container makes the environment naming context available to the application component instances at runtime. The application component's instances use the JNDI interfaces to obtain the values of the environment entries.

JNDI lookups and their associated references

- java:comp/env
  Application environment entries
- java:comp/env/jdbc
  JDBC DataSource resource manager connection factories
- java:comp/env/ejb
  EJB References
- java:comp/UserTransaction
  UserTransaction references
- java:comp/env/mail
  JavaMail Session Connection Factories
- java:comp/env/url
  URL Connection Factories
- java:comp/env/jms
  JMS Connection Factories and Destinations
- java:comp/ORB
  ORB instance shared across application components

Using Custom Resources

- custom resource accesses a local JNDI repository and an external resource accesses an external JNDI repository. Both types of resources need user-specified factory class elements, JNDI name attributes, etc.
- Use the Admin Console to configure JNDI connection factory resources, J2EE resources, and access for these resources.

Using External JNDI Repositories and Resources

- Often applications running on the Application Server require access to resources stored in an external JNDI repository. For example, generic Java objects could be stored in an LDAP server as per the Java schema. External JNDI resource elements let users configure such external resource repositories. The external JNDI factory must implement javax.naming.spi.InitialContextFactory interface

An example of the use of an external JNDI resource

```
<resources>
 <!-- external-jndi-resource element specifies how to access J2EE resources
 -- stored in an external JNDI repository. The following example
 -- illustrates how to access a java object stored in LDAP.
 -- factory-class element specifies the JNDI InitialContext factory that
 -- needs to be used to access the resource factory. property element
 -- corresponds to the environment applicable to the external JNDI context
 -- and jndi-lookup-name refers to the JNDI name to lookup to fetch the
 -- designated (in this case the java) object.
 -->
```

An example of the use of an external JNDI resource

```
<external-jndi-resource jndi-name="test/myBean"
    jndi-lookup-name="cn=myBean"
    res-type="test.myBean"
    factory-class="com.sun.jndi.ldap.LdapCtxFactory">
  <property name="PROVIDER-URL"
   value="ldap://ldapserver:389/o=myObjects" />
  <property name="SECURITY_AUTHENTICATION" value="simple" />
  <property name="SECURITY_PRINCIPAL", value="cn=joeSmith,
   o=Engineering" />
  <property name="SECURITY_CREDENTIALS" value="changeit" />
 </external-jndi-resource>
</resources>
```

Accessing EJB Components in a Remote Application Server

- The recommended approach for looking up an EJB component in a remote Application Server from a client (a servlet or EJB component) is to use the Interoperable Naming Service syntax. Host and port information is perpended to any global JNDI names and is automatically resolved during the lookup. The syntax for an interoperable global name is as follows:
- corbaname:iiop:host:port#a/b/name

Accessing EJB Components in a Remote Application Server

- This makes the programming model for accessing EJB components in another Application Server exactly the same as accessing them in the same server. The deployer can change the way the EJB components are physically distributed without having to change the code.

Accessing EJB Components in a Remote Application Server

- For Java EE components, the code still performs a java:comp/env lookup on an EJB reference. The only difference is that the deployer maps the ejb-reference element to an interoperable name in an Application Server deployment descriptor file instead of to a simple global JNDI name.

Accessing EJB Components in a Remote Application Server

- The ejb-ref element in sun-web.xml looks like

```
<ejb-ref>
  <ejb-ref-name>ejb/Foo</ejb-ref-name>
  <jndi-name>corbaname:iiop:host:port#a/b/Foo</jndi-name>
<ejb-ref>
```

Code Example

- Java EE client:

```
Context ic = new InitialContext();
Object o = ic.lookup("java:comp/env/ejb/Foo");
```

- For non-Java EE client

```
Context ic = new InitialContext();
Object o =
   ic.lookup("corbaname:iiop:host:port#a/b/Foo"
   );
```

References

JNDI:

- http://java.sun.com/javase/technologies/core/jndi/index.jsp
- http://java.sun.com/javase/6/docs/technotes/guides/jndi/jndi-cos.html
- http://glassfish.dev.java.net/javaee5/ejb/EJB_FAQ.html
- http://www.theserverside.com/news/thread.tss?thread_id=39828
- http://java.sun.com/javaee/5/docs/tutorial/doc/ (chapter 34)
- http://www.sun.com/software/products/appsrvr_pe/index.xml
- http://developers.sun.com/appserver/reference/techart/orb.html

Reference

DS/LDAP:

- ww.opds.org
- http://www.openldap.org/
- http://www.sun.com/software/products/directory_srvr_ee/dir_srvr/index.xml