

## JNDI and LDAP Interview Questions

How To



### What is JNDI ? And what are the typical uses within a J2EE application ?

JNDI stands for Java Naming and Directory Interface. It provides a generic interface to LDAP (Lightweight Directory Access Protocol) and other directory services like NDS, DNS (Domain Name System) etc. It provides a means for an application to locate components that exist in a name space according to certain attributes. A J2EE application component uses JNDI interfaces to look up and reference system-provided and user-defined objects in a component environment. JNDI is not specific to a particular naming or directory service. It can be used to access many different kinds of systems including file systems. The JNDI API enables applications to look up objects such as DataSources, EJBs, MailSessions, JMS connection factories and destinations (Topics/Queues) by name. The Objects can be loaded into the JNDI tree using a J2EE application server's administration console. To load an object in a JNDI tree, choose a name under which you want the object to appear in a JNDI tree. J2EE deployment descriptors indicate the placement of J2EE components in a JNDI tree. The parameters you have to define for JNDI service are as follows:

- The name service provider class name (WsnInitialContext for WebSphere application server).

```
Hashtable env = new Hashtable();  
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.WsnInitialContextFactory");
```

- The provider URL :
- The name service hostname.
- The name service port number.

Crush  
Goblins.  
Not  
Candy.



HEARTHSTONE  
HEROES OF WARCRAFT

Play Free

```
env.put(Context.PROVIDER_URL, " iiop://localhost:1050");  
Context ctx = new InitialContext(env);
```

## **Explain the difference between the look up of "java:comp/env/ejb/MyBean" and "ejb/MyBean" ?**

### **java:comp/env/ejb/MyBean**

This is a logical reference, which will be used in your code.

### **ejb/MyBean**

This is a physical reference where an object will be mapped to in a JNDI tree.

The logical reference (or alias) `java:comp/env/ejb/MyBean` is the recommended approach because you cannot guarantee that the physical JNDI location `ejb/MyBean` you specify in your code will be available. Your code will break if the physical location is changed. The deployer will not be able to modify your code. Logical references solve this problem by binding the logical name to the physical name in the application server. The logical names will be declared in the deployment descriptors (`web.xml` and/or `ejb-jar.xml`) as follows and these will be mapped to physical JNDI locations in the application server specific deployment descriptors. To look up a JDBC resource from either Web (`web.xml`) or EJB (`ejb-jar.xml`) tier, the deployment descriptor should have the following entry:

```
<resource-ref>  
<description>The DataSource</description>  
<res-ref-name>jdbc/MyDataSource</res-ref-name>  
<res-type>javax.sql.DataSource</res-type>  
<res-auth>Container</res-auth>  
</resource-ref>
```

To use it:

```
Context ctx = new InitialContext();  
Object ref = ctx.lookup(java:comp/env/jdbc/MyDataSource);
```

To look up EJBs from another EJB or a Web module, the deployment descriptor should have the following entry:

```
<ejb-ref>  
<description>myBean</description>  
<ejb-ref-name>ejb/MyBean</ejb-ref-name>  
<ejb-ref-type>Entity</ejb-ref-type>  
<ejb-link>Region</ejb-link>  
<home>com.MyBeanHome</home>  
<remote>com.MyBean</remote>  
</ejb-ref>
```

To use it:

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

```
Object ref = ctx.lookup(java:comp/env/ejb/MyBean);
```

## What is a JNDI InitialContext ?

All naming operations are relative to a context. The InitialContext implements the Context interface and provides an entry point for the resolution of names.

## What is an LDAP server ? And what is it used for in an enterprise environment ?

LDAP stands for Lightweight Directory Access Protocol. This is an extensible open network protocol standard that provides access to distributed directory services. LDAP is an Internet standard for directory services that run on TCP/IP. Under OpenLDAP and related servers, there are two servers slapd, the LDAP daemon where the queries are sent to and slurpd, the replication daemon where data from one server is pushed to one or more slave servers. By having multiple servers hosting the same data, you can increase reliability, scalability, and availability.

- It defines the operations one may perform like search, add, delete, modify, change name
- It defines how operations and data are conveyed.

LDAP has the potential to consolidate all the existing application specific information like user, company phone and e-mail lists. This means that the change made on an LDAP server will take effect on every directory service based application that uses this piece of user information. The variety of information about a new user can be added through a single interface which will be made available to Unix account, NT account, e-mail server, Web Server, Job specific news groups etc. When the user leaves his account can be disabled to all the services in a single operation. So LDAP is most useful to provide "white pages" (e.g. names, phone numbers, roles etc) and "yellow pages" (e.g. location of printers, application servers etc) like services. Typically in a J2EE application environment it will be used to authenticate and authorize users.

## Why use LDAP when you can do the same with relational database (RDBMS) ?

In general LDAP servers and RDBMS are designed to provide different types of services. LDAP is an open standard access mechanism, so an RDBMS can talk LDAP. However the servers, which are built on LDAP, are optimized for read access so likely to be much faster than RDBMS in providing read access. So in a nutshell, LDAP is more useful when the information is often searched but rarely modified. (Another difference is that RDBMS systems store information in rows of tables whereas LDAP uses object oriented hierarchies of entries.) . Key LDAP Terms:

- DIT: Directory Information Tree. Hierarchical structure of entries, those make up a directory.
- DN: Distinguished Name. This uniquely identifies an entry in the directory. A DN is made up of relative DNs of the entry and each of entry's parent entries up to the root of the tree. DN is read from right to left and commas separate these names. For example 'cn=Peter Smith, o=ACME, c=AUS'.
- objectClass: An objectClass is a formal definition of a specific kind of objects that can be stored in the directory. An ObjectClass is a distinct, named set of attributes that represent something concrete such as a user, a computer, or an application.
- LDAP URL: This is a string that specifies the location of an LDAP resource. An LDAP URL consists of a server host and a port, search scope, baseDN, filter, attributes and extensions.
- LDAP schema: defines rules that specify the types of objects that a directory may contain and the required optional attributes that entries of different types should have.
- Filters: In LDAP the basic way to retrieve data is done with filters. There is a wide variety of operators that can be used as follows: & (and), | (or), ! (not), ~= (approx equal), >= (greater than or equal), <= (less than or equal), \* (any) etc.

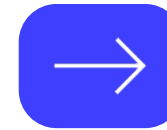
## So where does JNDI fit into this LDAP ?

JNDI provides a standard API for interacting with naming and directory services using a service provider interface (SPI), which is analogous to JDBC driver. To connect to an LDAP server, you must obtain a reference to an object that implements the DirContext. In most applications, this is done by using an InitialDirContext object that takes a Hashtable as an argument:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:387");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=Directory Manager");
env.put(Context.SECURITY_CREDENTIALS, "myPassword");
DirContext ctx = new InitialDirContext(env);
```

# Hadoop Hive ODBC Drivers

Improve Data Throughput with ODBC Hadoop Hive Driver. Free Trial



[RMI Interview Questions >>>](#)

[Home Clouds](#)

---