How JNDI works in Tomcat

Tomcat provides a JNDI InitialContext implementation instance for each web application running under it.   
  
Using this initalContext object we traverse into the JNDI namespace,

Context ctx = intctx.lookup("java:comp/env");

Why **java:comp**?  
  
Quoting JNDI Docs:  
*The JNDI does not define any naming policy on its own. However, one important platform that does define a limited set of naming policies for using the JNDI is the J2EE specification. It defines a logical namespace that application components (such as EJBs, servlets, and Jsps) can use to name resources, components, and other data.*  
  
J2EE specification requires that the namespace be provided to a component by its container, the entity that executes the component.  
The specification states that the containers must provide the minimum, java: namespace and it's child namespaces with the below naming convention:  
  
**java:comp**  
The name *comp* is short for component. *java:comp* is bound to a subtree reserved for component related bindings.  
  
**java:comp/env**  
The subcontext *java:comp/env* is reserved for the component's environment-related bindings, usually defined by its deployment descriptor(web.xml file).   
*env* is short for environment.   
  
The J2EE recommends (but does not require) the following structure for the *env* namespace.

|  |  |
| --- | --- |
| **JNDI Subcontext** | **Resource Manager Type (Connection Factory Type)** |
| java:comp/env/ejb | Enterprise JavaBeansTM are placed under the "ejb" subtree. |
| java:comp/env/jdbc | For JDBC dataSource references (javax.sql.DataSource) |
| java:comp/env/jms | For JMS connection factories (javax.jms.TopicConnectionFactory/ javax.jms.QueueConnectionFactory) |
| java:comp/env/mail | For Java mail connection factories (javax.mail.Session) |
| java:comp/env/url | For URL connection factories (java.net.URL) |
| java:comp/env/eis | For Enterprise information system (javax.resource.cci.ConnectionFactory |
| java:comp/env/eis/JAXR | Not sure, maybe for JAX-RPC connections for EIS (javax.xml.registry.ConnectionFactory) |

**Resource Factories**  
  
A factory is an object that creates other objects on demand. A resource factory creates resource objects, such as beans, database connections or message service connections.  
Resource factories are not needed for wrapper classes but when it comes to user defined beans or database connections we need to specify the factory class. The container needs this information to create the requested object.   
  
In the upcoming examples you will notice that we specify the Resource Factory type in web.xml.   
  
Other namespaces optional for a container are:

|  |
| --- |
| **JNDI Subcontext** |
| java:comp/UserTransaction |
| java:comp/EJBContext |
| java:comp/ORB |
| java:comp/TransactionSynchronizationRegistry |

The new J2EE 6 specification adds new namespaces:  
java:global, java:app, and java:module.  
  
To know the differences between each checkout this [sof](http://stackoverflow.com/questions/7458114) answer.  
  
**Printing JNDI tree**  
Let's check what are the namespaces available under tomcat's jndi initalContext. We print the JNDI tree by calling InitalContext's list() method iteratively.

**protected** **void** **doGet**(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

System.out.println("doGet");

Context initialContext;

**try** {

initialContext = **new** InitialContext();

findSubContexts(initialContext, "java:");

} **catch** (NamingException e) {

System.out.println("Oh no, this shouldn't happen");

e.printStackTrace(System.out);

} **catch** (Exception e) {

}

**private** **void** **findSubContexts**(Context initialContext,

String name) {

NameClassPair ncp = **null**;

**try** {

NamingEnumeration ne = initialContext.list(name);

**while** (ne.hasMoreElements()) {

ncp = (NameClassPair) ne.nextElement();

System.out.println(name+" ::: "+ncp.getName());

findSubContexts(initialContext, name + "/" + ncp.getName());

}

} **catch** (NamingException e) {

System.out.println("No names bound under : "+name);

*//System.out.println(e.getMessage());*

}

}

Output:

java: --> comp  
java:/comp --> UserTransaction  
No names bound under : java:/comp/UserTransaction  
java:/comp --> env  
java:/comp/env --> isConnected  
No names bound under : java:/comp/env/isConnected  
java:/comp --> Resources  
java:/comp/Resources --> META-INF  
java:/comp/Resources/META-INF --> MANIFEST.MF  
java:/comp/Resources --> WEB-INF  
java:/comp/Resources/WEB-INF --> classes  
java:/comp/Resources/WEB-INF/classes --> sample  
java:/comp/Resources/WEB-INF/classes/sample --> SampleServlet.class  
java:/comp/Resources/WEB-INF/classes/sample --> bean  
java:/comp/Resources/WEB-INF/classes/sample/bean --> MyBean.class  
java:/comp/Resources/WEB-INF/classes/sample/bean --> factory  
java:/comp/Resources/WEB-INF/classes/sample/bean/factory --> MyBeanFactory.class  
java:/comp/Resources/WEB-INF --> lib  
java:/comp/Resources/WEB-INF --> web.xml  
java:/comp/Resources --> index.html

The Java EE standard also defines the xml elements for the deployment descriptor (web.xml) to define java:comp/env bindings. The containers sees to it that any bindings defined in web.xml are created, initialized and available (through lookups) before the container starts the web application/component.  
  
Note: Guess you noticed the inner namespaces (java:/comp/Resources/...) created for every file and folder in the deployment. This availablility of runtime resources through JNDI makes a lot of things possible.

**import** org.apache.naming.resources.FileDirContext;

...

FileDirContext value = (FileDirContext) initialContext.lookup ("java:/comp/Resources/WEB-INF/classes/sample/bean/factory");

Next step is to learn how to set jndi env in tomcat