Deployment Descriptor

Deployment descriptor: An XML file that specifies information about the bean such as its transaction attributes.

- You package the files in the preceding list into an EJB JAR file, the module that stores the enterprise bean.
- To assemble a J2EE application, you package one or more modules--such as EJB JAR files--into an EAR file, the archive file that holds the application.

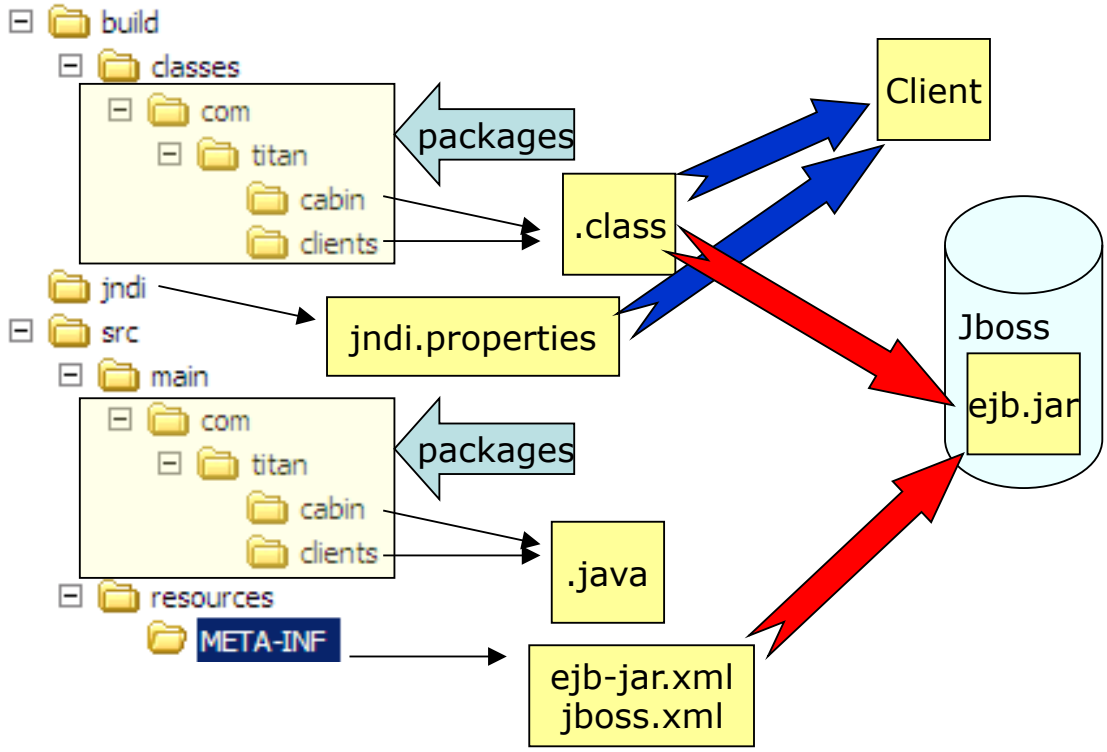
ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
    version="2.1">
    <enterprise-beans>
        <session>
            <ejb-name>HelloWorldEJB</ejb-name>
            <home>examples.ejb21.HelloHome</home>
            <remote>examples.ejb21.Hello</remote>
            <local-home>examples.ejb21.HelloLocalHome</local-home>
            <local>examples.ejb21.HelloLocal</local>
            <ejb-class>examples.ejb21.HelloBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    ...
</ejb-jar>
```

ejb-jar.xml (continued)

```
<assembly-descriptor>
   <security-role>
      <description>   This role represents everyone who is allowed
                   full access to the HelloWorldEJB. </description>
     <role-name>everyone</role-name>
   </security-role>
   <method-permission>
     <role-name>everyone</role-name>
     <method>
         <ejb-name>HelloWorldEJB</ejb-name>
         <method-name>*</method-name>
     </method>
   </method-permission>
   <container-transaction>
     <method>
        <ejb-name>HelloWorldEJB</ejb-name>
        <method-name>*</method-name>
     </method>
     <trans-attribute>Required</trans-attribute>
   </container-transaction>
 </assembly-descriptor>
```

The file structure



Introduction to Session beans

EJB 3.0

Remote Interface

```
EJB 2.1 ===================================================

public interface Hello extends javax.ejb.EJBObject
{
/**
* The one method - hello - returns a greeting to the client.
*/
   public String hello() throws java.rmi.RemoteException;
}

EJB 3.0 ===================================================

package examples.session.stateless;
public interface Hello {
    public String hello();
}
```

business interface

Bean Implementation

```
EJB 2.1 ============================================================
public class HelloBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext ctx;
    public void ejbCreate() { System.out.println("ejbCreate()"); }
    public void ejbRemove() { System.out.println("ejbRemove()"); }
    public void ejbActivate() { System.out.println("ejbActivate()");}
    public void ejbPassivate() {System.out.println("ejbPassivate()");}
    public void setSessionContext(javax.ejb.SessionContext ctx) {
      this.ctx = ctx; }
    public String hello() {
        System.out.println("hello()");  return "Hello, World!";
    }
}
EJB 3.0 ============================================================
package examples.session.stateless;
import javax.ejb.Remote; import javax.ejb.Stateless;
@Stateless
@Remote(Hello.class)
public class HelloBean implements Hello {
    public String hello() {
        System.out.println("hello()"); return "Hello, World!";
    }
}
```

enterprise bean instance

The remote client – 3.0

```
package examples.session.stateless;
import javax.naming.Context;
import javax.naming.InitialContext;
public class HelloClient {
    public static void main(String[] args) throws Exception {
    /*
    * Obtain the JNDI initial context.
    *
    * The initial context is a starting point for
    * connecting to a JNDI tree.
    */
        Context ctx = new InitialContext();
        Hello hello = (Hello)
        ctx.lookup("examples.session.stateless.Hello");
    /*
    * Call the hello() method on the bean.
    * We then print the result to the screen.
    */
        System.out.println(hello.hello());
    }
}
```

ejb-jar.xml – 3.0

```
<?xml version="1.0" encoding="UTF-8" ?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd"
version="3.0">
<enterprise-beans>
</enterprise-beans>
</ejb-jar>
```

Keep in mind these terms…

- The ***enterprise bean instance*** is a plain old Java object instance of an enterprise bean class. It contains business method implementations of the methods defined in the remote/local business interface, for session beans.

- The ***business interface*** is a plain old Java interface that enumerates the business methods exposed by the enterprise bean. Depending on the client view supported by the bean, the business interface can be further classified into a local business interface or a remote business interface.

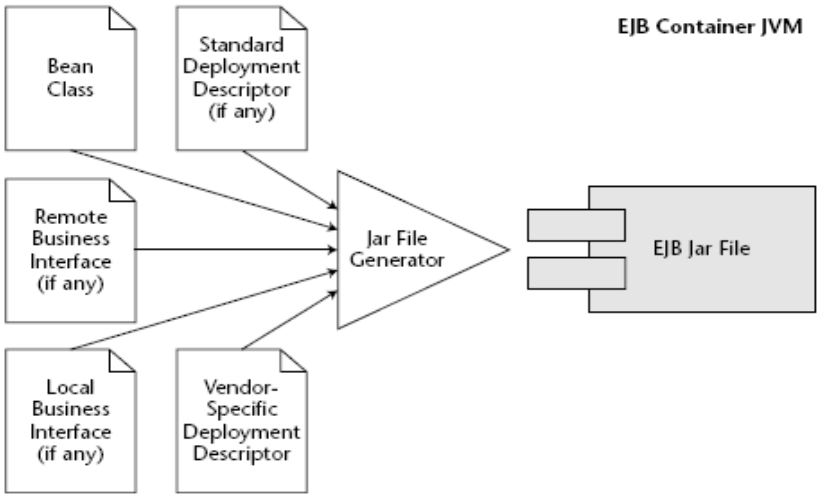
- The ***deployment descriptor*** is an XML file that specifies the middleware requirements for your bean. You use the deployment descriptor to inform the container about the services you need for the bean, such as transaction services, security, and so on. Alternatively, you can specify the middleware requirements using deployment annotations within the bean class as well.

Keep in mind these terms…

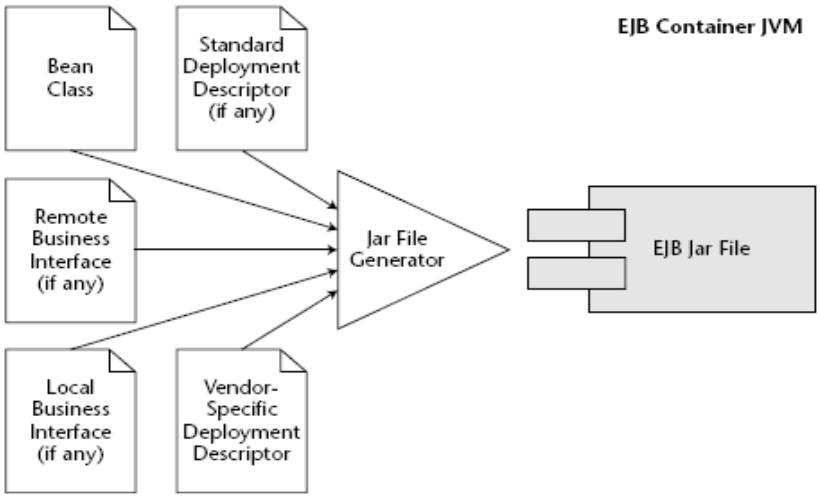
- The ***Ejb-jar*** file is the packaging unit for an enterprise bean, consisting of all the above artifacts. An EJB 3.0 Ejb-jar file can also consist of the old-style beans, if your application uses components defined using pre–EJB 3.0 technologies.

- The ***vendor-specific deployment descriptor*** lets you specify your bean's needs for proprietary container services such as clustering, load balancing, and so on. Avendor can alternatively provide deployment metadata for these services, which, like standard metadata, can be used within the bean class to specify the configuration for these services. The vendor-specific deployment descriptor's definition changes from vendor to vendor.

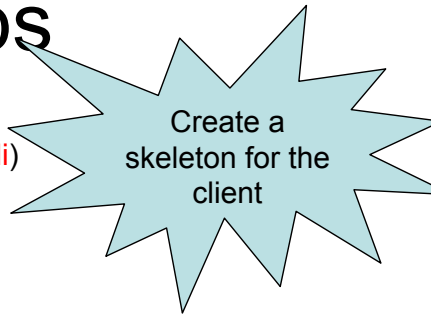
3.0 Packaging



3.0 Packaging



Development steps

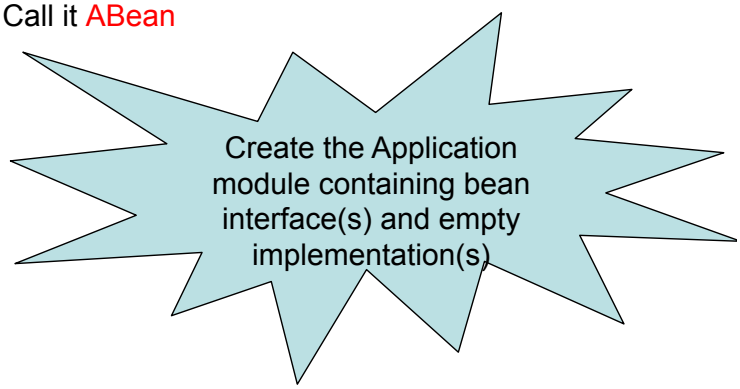


Create a new project for an empty EJB Application Client (Class AppCli)

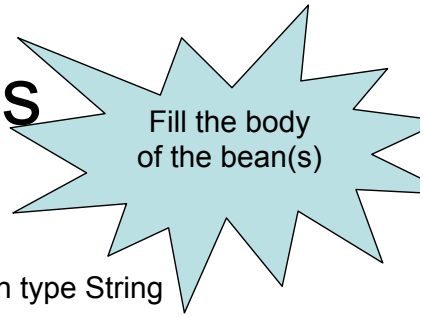
- New Project -> Java → Java Application
- name it AppCli in package pack1

Create New Project for the EJB Application

- New Project -> Java EE -> Enterprise Application
- name it EntApp
- choose ONLY “Create EJB Module” (not “Web Application Module”) and name it AppServ, create a Remote Interface in project AppCli. Call it ABean

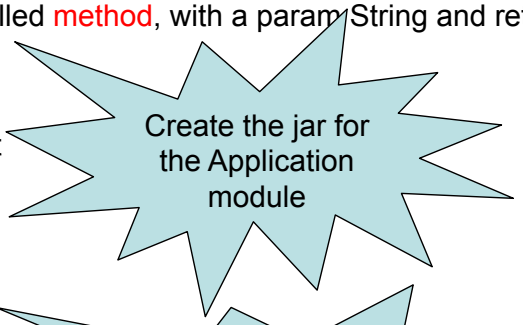


Development steps



Go to the Project AppServ

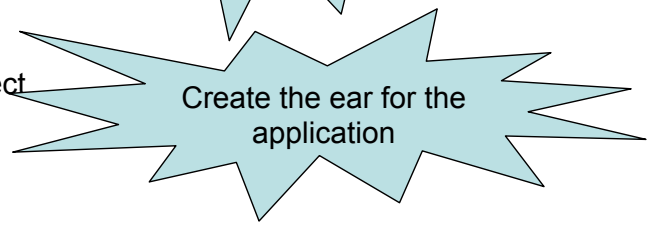
- go to the class ABean, right click “insert code”
- add a Business method called method, with a param String and return type String



- Clean and build the project

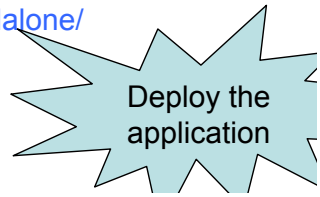
Go to the EntApp

- Clean and build the project



- Look in its dist folder: take the EntApp.ear file and drop it in the JBOSS standalone/deployments folder.
- Look at the JBoss console to find the JNDI reference

(java:jboss/exported/EntApp/AppServer/ABean!package_name.BBeanRemote)



Development steps

Go to AppCli ,

- fix the JNDI Access to the bean (using the info above).
- Remove any unneeded library
- Add the to the library the jboss-client.jar

Run the client!