

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>
```

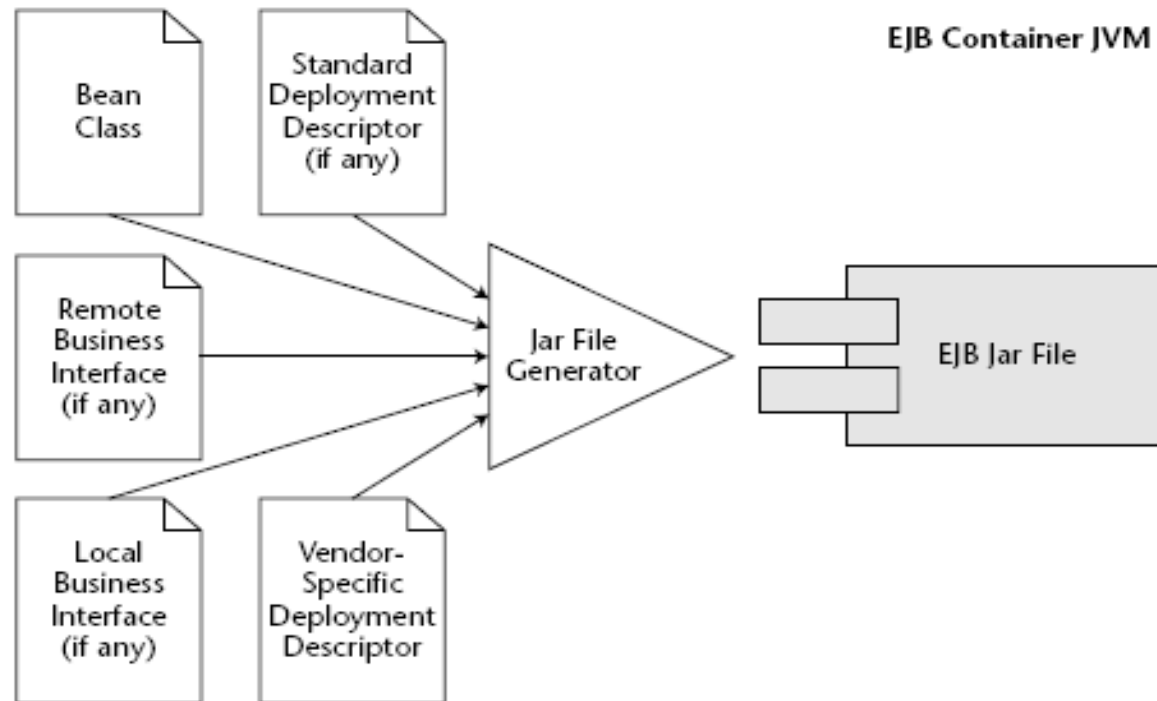
Keep in mind these terms...

- 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.

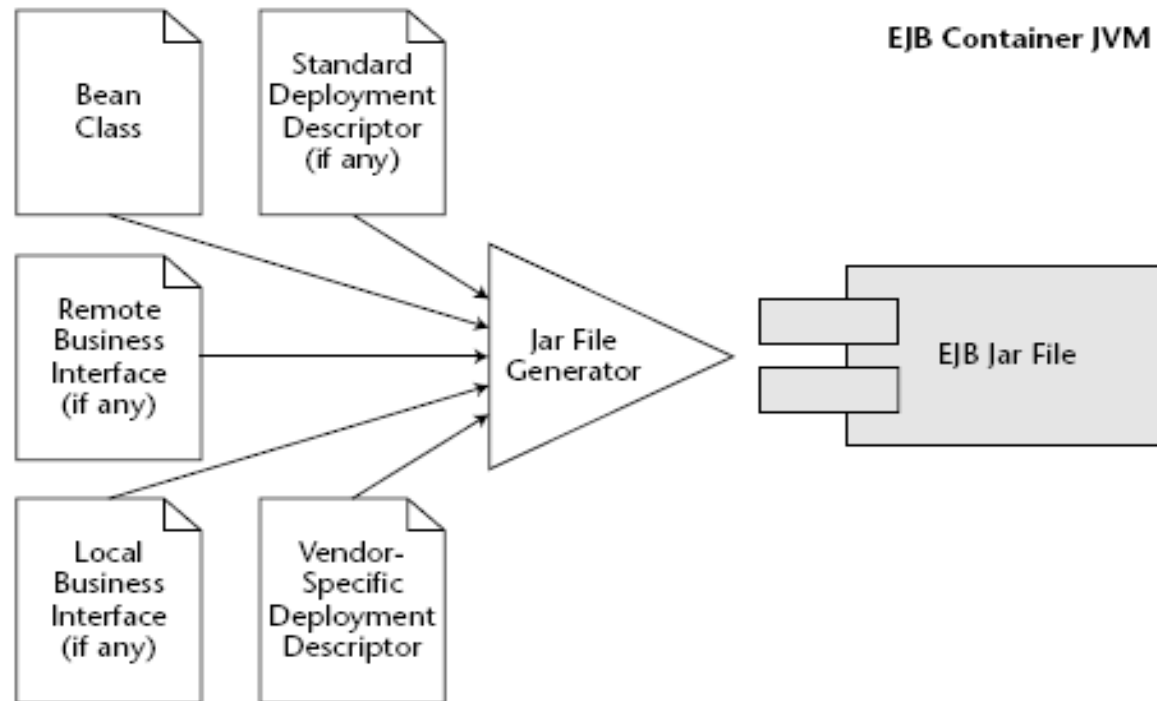
Keep in mind these terms...

- 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. A vendor 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.

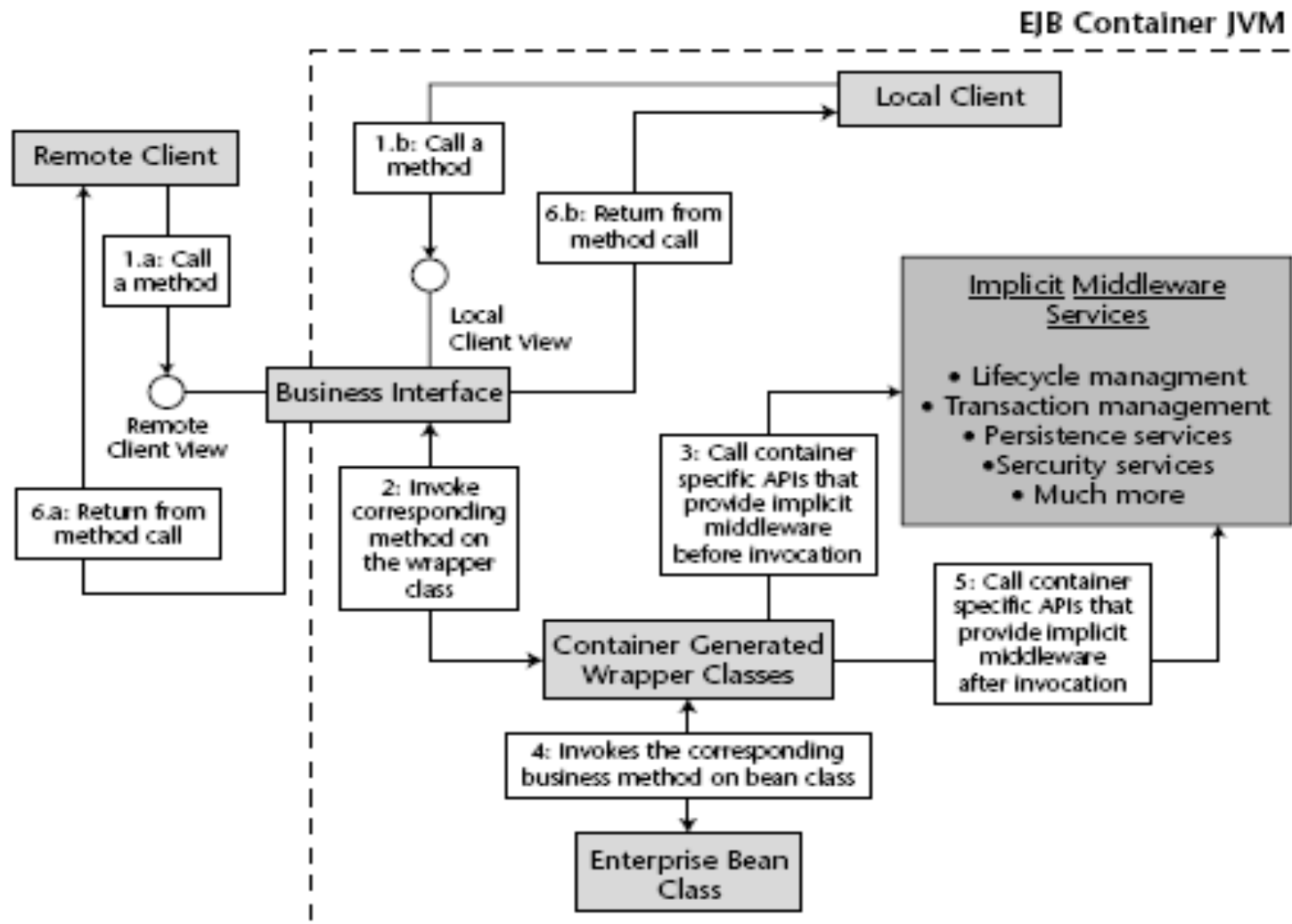
3.0 Packaging



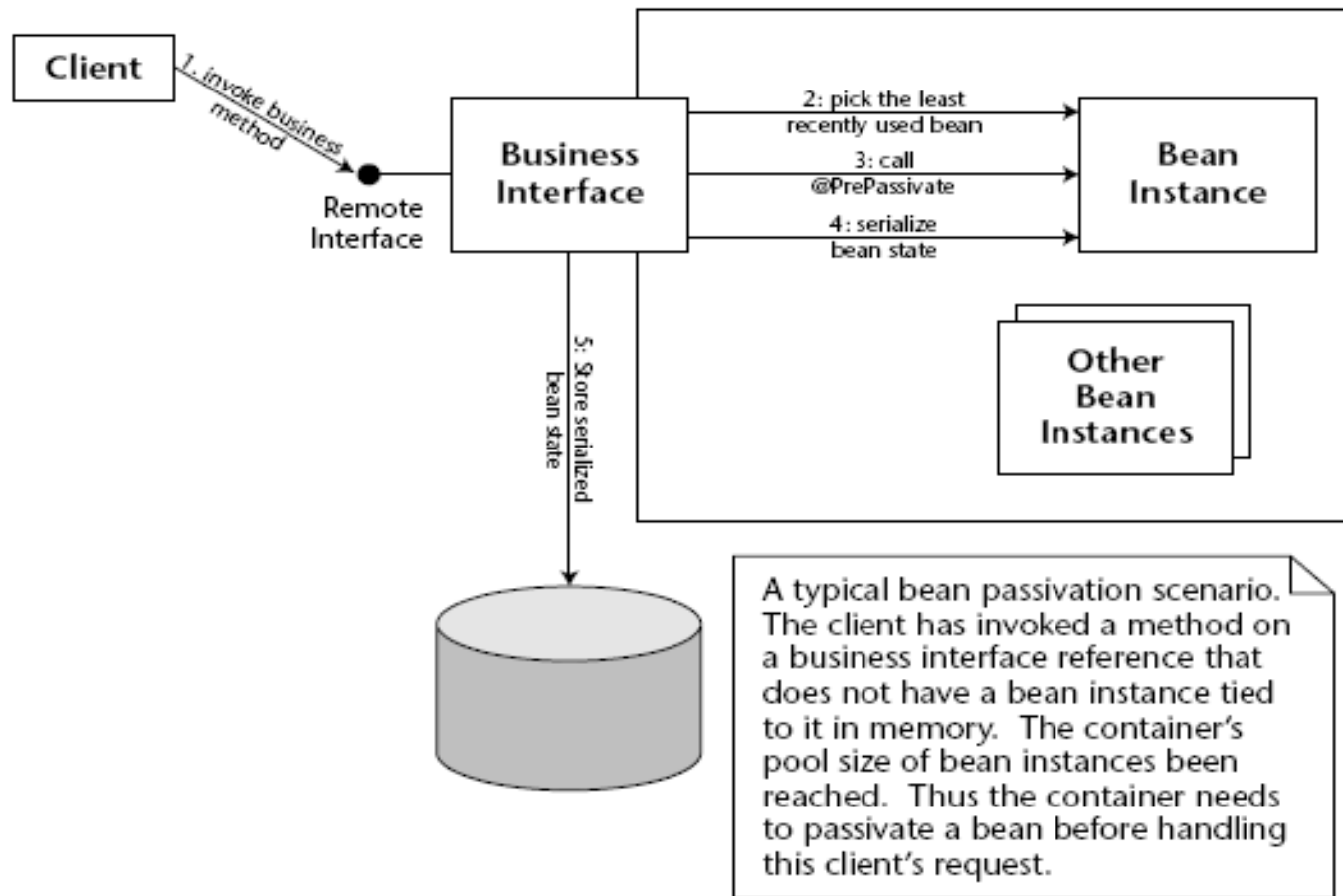
3.0 Packaging



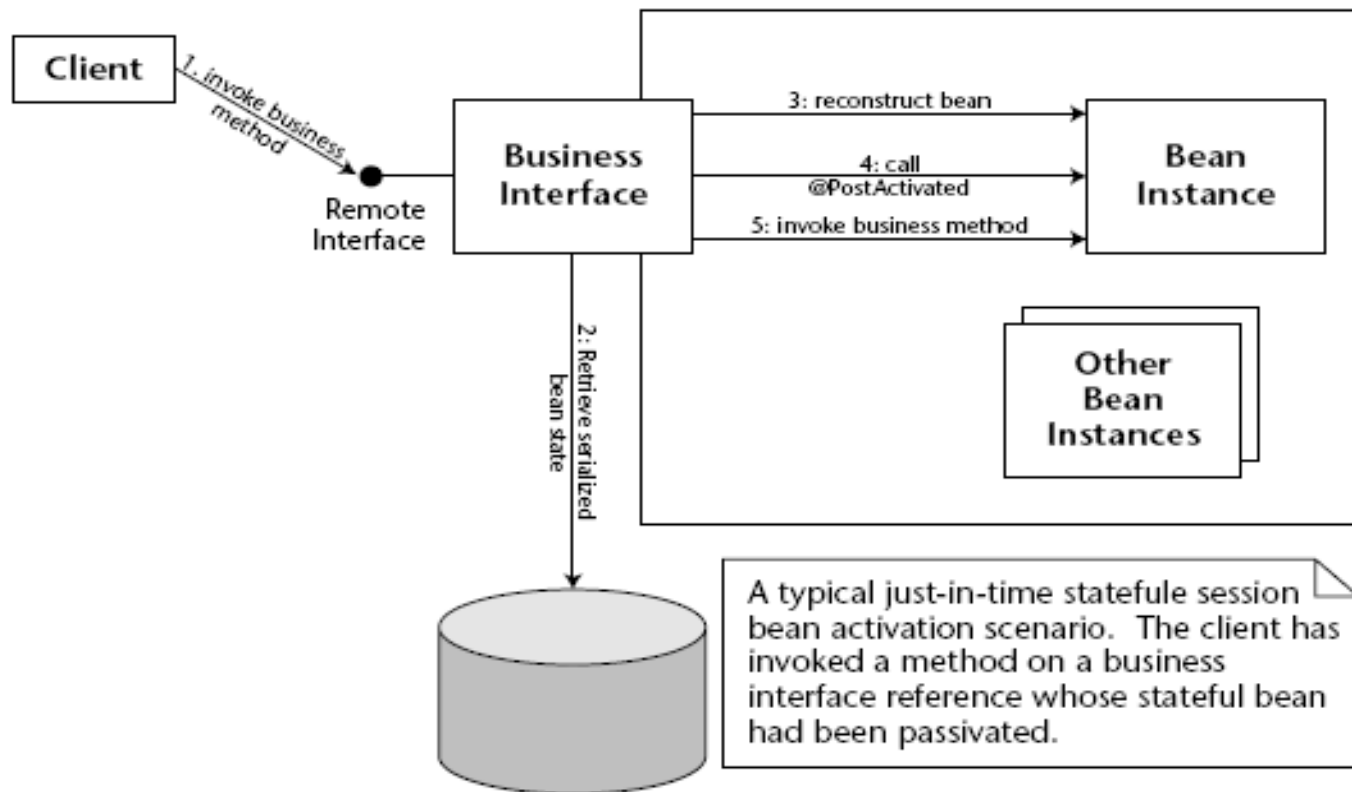
3.0 Lifecycle



Passivation



Activation



Managing the lifecycle – 3.0

```
@Stateful
public class MyBean {
    @PrePassivate
    public void passivate() {
        <close socket connections, etc...>
    }
    ...
    @PostActivate
    public void activate() {
        <open socket connections, etc...>
    }
    ...
}
```

JBOSS, Glassfish, Eclipse, NetBeans

Download Jboss 7.1.1

<http://www.jboss.org/jbossas/downloads/>

<http://www.netbeans.org/community/releases>

Setting the JNDI properties

```
private HelloBeanRemote lookupHelloBeanBean() {
    Properties props= new Properties();
    props.setProperty("java.naming.factory.initial",
        "org.jnp.interfaces.NamingContextFactory");
    props.setProperty("java.naming.provider.url",
        "jnp://localhost:1099");
    props.setProperty("java.naming.factory.url.pkgs",
        "org.jboss.naming:org.jnp.interfaces");
    try {
        Context c = new InitialContext(props);
        //return (HelloBeanRemote) c.lookup("java:comp/env/HelloBeanBean");
        return (HelloBeanRemote) c.lookup("EnterpriseDemo/HelloBeanBean/remote");
    } catch (NamingException ne) {
        ...
    }
}
```

JBOSS: see the JNDI names

<http://localhost:8080/jmx-console/HtmlAdaptor>

Click on Service=JNDI

Choose List