jar & jar files

jar command

Java Archive

- inherits from tar : Tape Archive

commands:

jar cvf filename | jar tvf filename | jar xvf filename

java –jar filename.jar

jar file

A JAR file can contain Java class files, XML descriptor files, auxiliary resources, static HTML files, and other files

META-INF - Manifest

see http://docs.oracle.com/javase/tutorial/deployment/jar/

specialized jars:

- war
- ear

Quick introduction to Java beans

Java Bean

JavaBeans are reusable software components for Java.

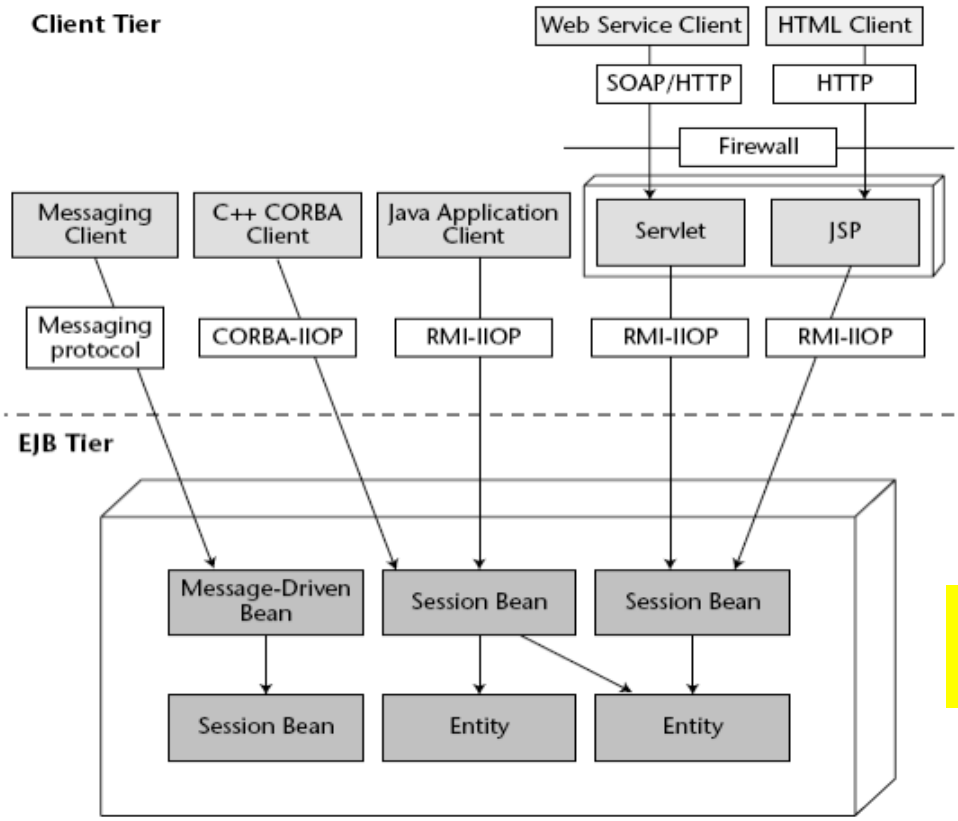
They are classes that encapsulate information and behavior into a single object (the bean).

They are serializable, have a 0-argument constructor, and allow access to properties using getter and setter methods.

Introduction to Session beans

Enterprise Java Beans

Architecture



Stateless session Beans

- **A stateless session bean does not maintain a conversational state for a particular client.**

When a client invokes the method of a stateless bean, the bean's instance variables may contain a state, but only for the duration of the invocation. When the method is finished, the state is no longer retained.

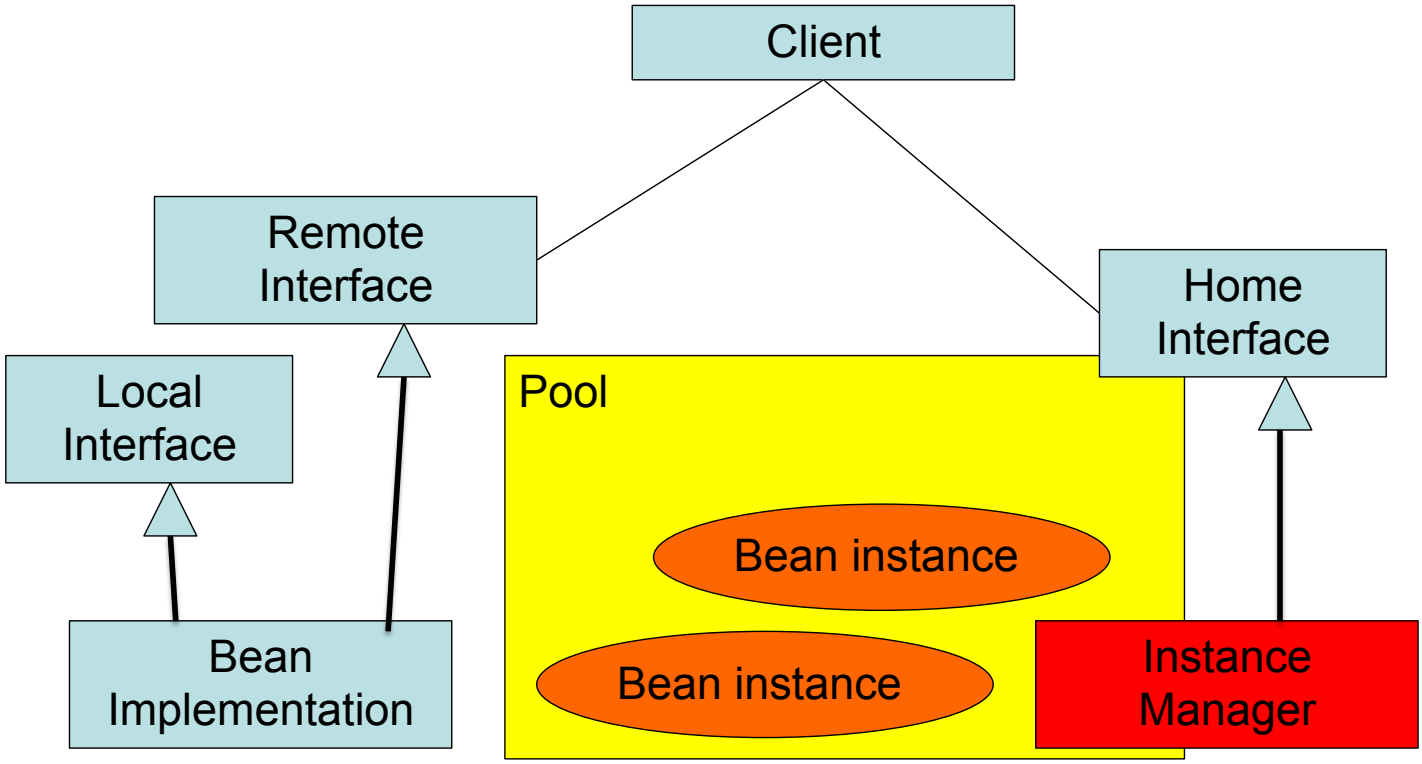
Stateless vs. stateful session Beans

- **All instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client.**

  => Stateless session beans can support multiple clients, and offer better scalability for applications that require large numbers of clients.

  Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

Logical structure



EJB ingredients

Interfaces: The remote and home interfaces are required for remote access. For local access, the local and local home interfaces are required.

Enterprise bean class: Implements the methods defined in the interfaces.

Helper classes: Other classes needed by the enterprise bean class, such as exception and utility classes.

Deployment descriptor: see later

Remote Interface

```
/**
* This is the HelloBean remote interface.
*
* This interface is what clients operate on when
* they interact with EJB objects. The container
* vendor will implement this interface; the
* implemented object is the EJB object, which
* delegates invocations to the actual bean.
*/
public interface Hello extends javax.ejb.EJBObject
{
/**
* The one method - hello - returns a greeting to the client.
*/
   public String hello() throws java.rmi.RemoteException;
}
```

Must throw RemoteException

Home Interface

```
/**
* This is the home interface for HelloBean. This interface
* is implemented by the EJB Server's tools - the
* implemented object is called the Home Object, and serves
* as a factory for EJB Objects.
*
* One create() method is in this Home Interface, which
* corresponds to the ejbCreate() method in HelloBean.
*/
public interface HelloHome extends javax.ejb.EJBHome
{
/*
* This method creates the EJB Object.
*
* @return The newly created EJB Object.
*/
  Hello create() throws java.rmi.RemoteException,
                        javax.ejb.CreateException;
}
```

Bean Implementation

```
/**
* Demonstration stateless session bean.
*/
public class HelloBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext ctx;
    //
    // EJB-required methods
    //
    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; }
    //
    // Business methods
    //
    public String hello() {
        System.out.println("hello()");
        return "Hello, World!";
    }
}
```

Client Implementation

```
import javax.naming.Context;
import javax.naming.InitialContext;
import java.util.Properties;
/**
* This class is an example of client code that invokes
* methods on a simple stateless session bean.
*/
public class HelloClient {
    public static void main(String[] args) throws Exception {
        /*
        * Setup properties for JNDI initialization.
        * These properties will be read in from the command line.
        */
        Properties props = System.getProperties();
        /*
        * Obtain the JNDI initial context.
        * The initial context is a starting point for
        * connecting to a JNDI tree. We choose our JNDI
        * driver, the network location of the server, etc.
        * by passing in the environment properties.
        */
        Context ctx = new InitialContext(props);
```

Client Implementation

```
        /* Get a reference to the home object - the
        * factory for Hello EJB Objects
        */
        Object obj = ctx.lookup("HelloHome");
        /* Home objects are RMI-IIOP objects, and so they must be cast
        * into RMI-IIOP objects using a special RMI-IIOP cast.
        */
        HelloHome home = (HelloHome)
        javax.rmi.PortableRemoteObject.narrow(obj, HelloHome.class);
        /* Use the factory to create the Hello EJB Object
        */
        Hello hello = home.create();
        /*Call the hello() method on the EJB object. The
        * EJB object will delegate the call to the bean,
        * receive the result, and return it to us.
        * We then print the result to the screen.
        */
        System.out.println(hello.hello());
        /*
        * Done with EJB Object, so remove it.
        * The container will destroy the EJB object.
        */
        hello.remove();
    }
}
```

The logical architecture