EJB3: Session Beans

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/msajsp.html

© 2009 Marty Hall

Agenda

- Stateless session beans
- Deploying EJB projects to server
- Remote clients for stateless session beans
- Stateful session beans
- Local access to EJBs

Overview

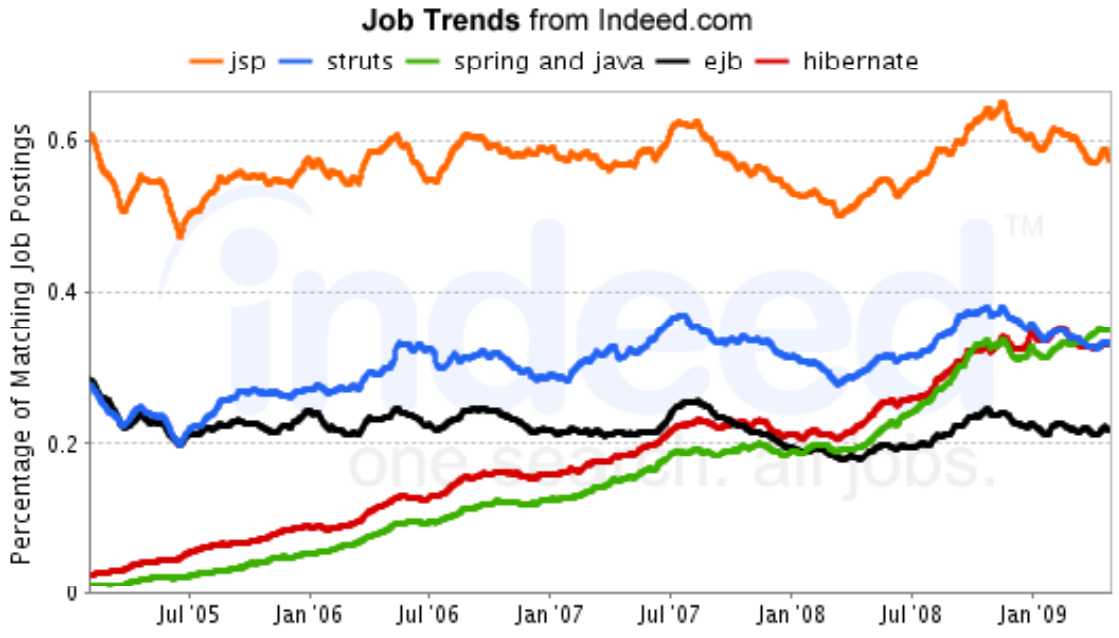
Benefits of EJB

- **Encapsulating business logic**
  - Business logic separated from control and presentation
- **Remote access**
  - Multiple apps on different servers can access EJBs
- **Simplicity**
  - Relatively easy to use compared to other remote-object systems
- **Broad vendor support**
  - JBoss, Oracle AS, WebLogic, WebSphere, Glassfish, etc.
- **Scalability**
  - Virtually all Java EE app servers support clustering, load balancing, and failover

Disadvantages of EJB

- **Complexity**
  - Although EJB3 might be simpler than other remote-object systems, remote-object frameworks are much more complex than local-object approaches.
    - Spring is easier and more powerful for local access
- **Requires Java EE server**
  - Can't run on Tomcat, Jetty, Resin, JRun, Resin
  - Java EE servers are usually much harder to configure, dramatically slower to start/restart during development and testing, and usually cost money
- **Requires latest Java EE version**
  - Must upgrade to latest server releases
- **Bad reputation due to earlier releases**
  - EJB2 was so complex that EJB has bad rap to this day

Industry Usage (Keywords in Worldwide Job Postings)



Session Beans: Overview

- **Stateless session beans**
  - On server, you make interface (marked with @Remote) and POJO that implements it (marked with @Stateless)
    - Ordinary Java object with ordinary methods that use ordinary Java objects as arguments and return types
    - No state (instance vars) maintained between method calls
  - Client uses InitialContext.lookup("name") to get ref.
    - Client makes normal method calls (data serialized on net.)
- **Stateful session beans**
  - Mark POJO with @Stateful instead of @Stateless
  - Mark special method with @Remove
  - Client does similar lookup and method calls, but needs to call the special method when done.
    - State (instance vars) maintained until that method called

Servers

- **Servers that support Java EE 5**
  - JBoss 5
  - Glassfish 2.1 (and Sun App Server built on it)
  - BEA WebLogic 10
  - Oracle AS 11
  - IBM WebSphere 7
  - Apache Geronimo 2
  - SAP NetWeaver Java EE 5 Edition
  - TmaxSoft JEUS 6
- **JBoss and Glassfish**
  - Used to demonstrate examples in this tutorial
  - Installation and setup details given in previous lecture

Stateless Session Beans

Stateless Session Beans: Idea

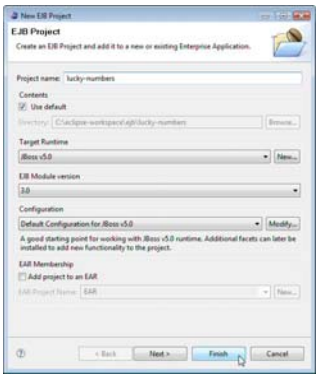
- **POJOs**
  - Ordinary Java classes; no special interfaces or parent classes. Often provide some sort of service such as mapping customer ids to Customer objects.
- **Local or remote access**
  - Can be accessed either on local app server (same machine as app that uses the beans) or remote app server (different machine from app that uses the beans)
- **Do not maintain client state**
  - Instance variables are protected from race conditions only during a single method call. Should not store client-specific state that needs to be accessed across multiple method calls.

Stateless Session Beans: Approach

- **Make new EJB project**
  - File → New → EJB Project
    - Can choose JBoss or Glassfish as target runtime. Either way, you can deploy project to both servers.
- **Define an interface**
  - Mark with @Remote
    - For access from other servers or from projects on same server
  - Mark with @Local
    - To only allow access from projects on same server (default)
- **Create a class that implements interface**
  - Mark with @Stateless for server's default JNDI mapping
  - Mark with @Stateless(mappedName="SomeJndiName")
- **Deploy to app server**
  - R-click server, Add/Remove Projects, start server

EJB Project

- **Making project**
  - File → New → EJB Project
  - Pick a name
  - JBoss or Glassfish as target runtime
    - You can later deploy to any compatible server; you are not restricted to the one you chose initially.
  - Optional: add to an EAR
    - If you make Dynamic Web project later that wants to use @EJB, you can add it to the same EAR



- **Deploying project**
  - R-click on JBoss or Glassfish (or any Java EE 5 server that you have registered with Eclipse), Add and Remove Projects, choose Project, Add, Finish
  - R-click on JBoss or Glassfish (or other), Start

Interface

```
package coreservlets.bean;

import javax.ejb.*;

@Remote
public interface NumberService {
  public double getNumber(double range);
}
```

Means that you can use InitialContext to access and access bean from either same server or remote machine. Code in same EAR can also use @EJB to access it. If you use @Local (or nothing at all), only code on same server in same EAR can access it.

Remote client will pass in normal arguments and get normal return values. Behind the scenes, however, data will be serialized and sent across network.

Class that Implements Interface

```
package coreservlets.bean;

import javax.ejb.*;

@Stateless(mappedName="NumberCreator")
public class NumberServiceBean
       implements NumberService {
  public double getNumber(double range) {
    return(Math.random() * range);
  }
}
```

If you just use @Stateless, remote clients need to use server-specific JNDI name. If you are using JBoss only (or Glassfish only), this is fine. But if you want to deploy the same app to multiple servers and not change the client code at all, it simplifies things to use a common JNDI name via @Stateless(mappedName="JNDI-Name")

Clients for Stateless Session Beans

Idea

- **Clients find the bean via JNDI**
  - Client Java code doesn't even know the machine on which the bean resides
- **Clients use the bean like a normal POJO**
  - But arguments and return values are sent across network
  - So, custom classes should be Serializable
- **Core code**
  - InitialContext context = new InitialContext();
  - InterfaceName bean =
    (InterfaceName)context.lookup("JNDI-Name");
- **jndi.properties**
  - Text file in classpath; gives remote URL and other info

Remote Client Project

- **Made new Dynamic Web Project**
  - Chose lucky-numbers-client as name
  - Chose JBoss 5 as target runtime
    - Can be deployed to Glassfish or other Java EE 5 server
- **jndi.properties**
  - Created text file called "jndi.properties" in src folder
  - Must give remote URL. Often other info. Server specific!
  - Details shown in upcoming slide
- **Deploying**
  - Does not need to run on same machine as EJB Project.
  - Standalone (desktop) clients don't need to be deployed to *any* app server; they can just have a "main" method.
  - Web apps should be deployed to a Java EE 5 app server

Remote Standalone (Desktop) Client

```
package coreservlets.client;
import javax.naming.*;
import coreservlets.bean.*;

public class LuckyNumberClient {
  public static void main(String[] args) throws Exception {
    InitialContext context = new InitialContext();
    NumberService service =
      (NumberService)context.lookup("NumberCreator");
    System.out.printf("Small lucky number:  %6.2f.%n",
                      service.getNumber(10));
    System.out.printf("Medium lucky number: %6.2f.%n",
                      service.getNumber(100));
    System.out.printf("Big lucky number:    %6.2f.%n",
                      service.getNumber(1000));
  }
}
```

mappedName given in @Stateless annotation

Used like normal object. Behind the scenes, however, arguments and return value are sent across network.

Note on Name in context.lookup

- **Issue**
  - You pass a name to context.lookup. If you just use @Remote with no mappedName, default name is different for JBoss than for Glassfish (or other servers).
  - In JBoss, the default JNDI name would be "NumberServiceBean/remote"
  - In Glassfish, the default JNDI name would be "coreservlets.bean.NumberService"
- **Solution: use mappedName**
  - I use @Stateless(mappedName="NumberCreator") instead of just @Stateless
  - So, I can use the same name (NumberCreator) regardless of which server the EJB project is deployed to

Remote Client: jndi.properties

- **For JBoss**

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost:1099
```

- **For Glassfish**

```
org.omg.CORBA.ORBInitialHost=localhost
```

Change this hostname if app server is on different host than client.

- **Notes**
  - Put in "src" folder in Eclipse to be sure it is in classpath
  - The Eclipse project you can download has both versions. Just copy/rename jboss-jndi.properties or glassfish-jndi.properties.

Remote Standalone Client: Results

- **lucky-numbers (EJB Project)**
  - Deployed to JBoss or Glassfish. Output is same.
- **lucky-numbers-client (Dynamic Web Proj.)**
  - Not yet deployed to any server.
  - jndi.properties matches the server of EJB project

- **Output**

```
Small lucky number:    9.73.
Medium lucky number:  29.41.
Big lucky number:    114.37.
```

Remote Web Client (Servlet)

```
public class LuckyNumberServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    try {
      InitialContext context = new InitialContext();
      NumberService service =
        (NumberService)context.lookup("NumberCreator");
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String docType =
        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
        "Transitional//EN\">\n";
```

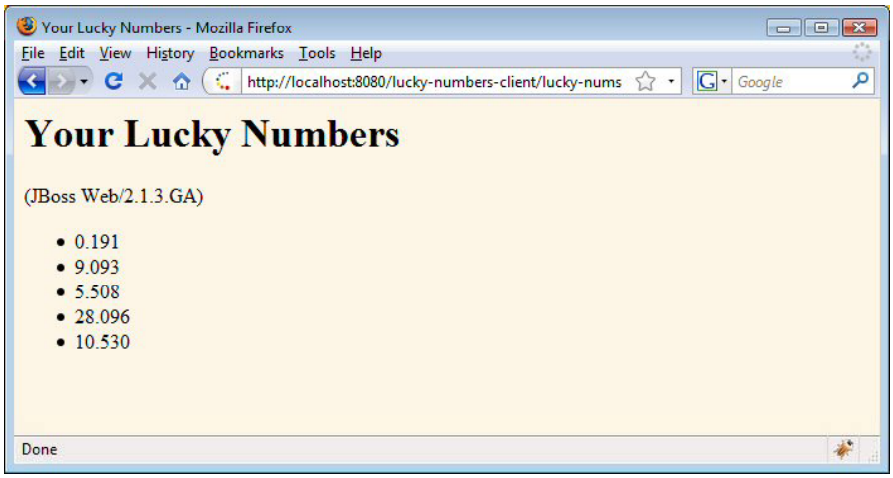

Remote Web Client (Servlet Continued)

```
      out.println(docType +
                  "<HTML>\n" +
                 "<HEAD><TITLE>Your Lucky Numbers</TITLE></HEAD>\n"+
                  "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                  "<H1>Your Lucky Numbers</H1>\n" +
                  "("+getServletContext().getServerInfo()+ ")\n"+
                  "<UL>");
      for(int i=0; i<5; i++) {
        out.printf("  <LI>%5.3f\n", service.getNumber((i+1)*10));
      }
      out.println("</UL>\n</BODY></HTML>");
    } catch(NamingException ne) {
      System.out.println("Error: " + ne);
    }
  }
}
```

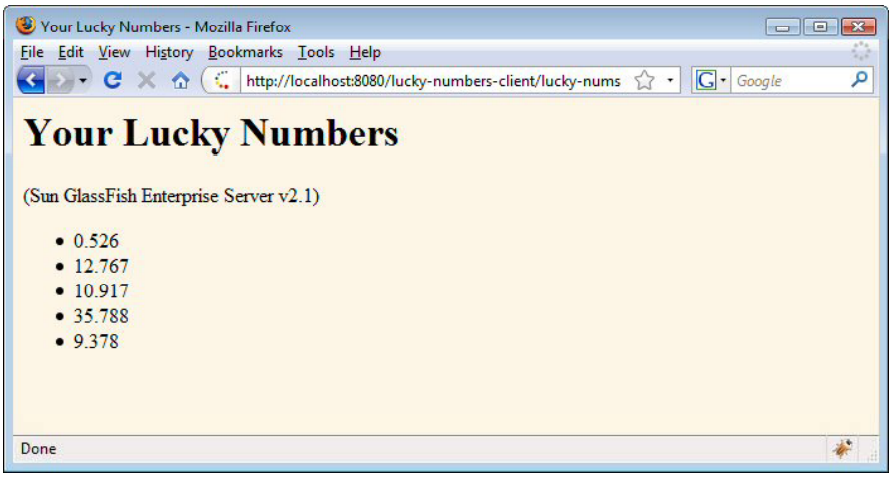
Remote Web Client (web.xml)

```
…
 <servlet>
    <servlet-name>
      Servlet with Individual Numbers from EJB
    </servlet-name>
    <servlet-class>
      coreservlets.client.LuckyNumberServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      Servlet with Individual Numbers from EJB
    </servlet-name>
    <url-pattern>/lucky-nums</url-pattern>
  </servlet-mapping>
…
```


Remote Web Client (Results JBoss 5)



Remote Web Client (Results Glassfish 2.1)



Using @EJB to Access Local Beans

Idea

- **Use @EJB before instance variable**
  - Bean will be automatically instantiated and assigned. Eg:
    - @EJB private SomeService myService;
- **Useful for**
  - One EJB that uses another EJB as part of its logic
    - Always good idea since EJBs are usually together
  - Multiple Web apps that use the same business logic
    - Simpler, but Web app can't move to remote server
- **Restrictions**
  - Before instance variables, not local variables.
  - Both classes must be part of same EAR on same server
    - In Eclipse, all classes in a single EJB project satisfy this
    - If you use an EJB project (EJBs) and Dynamic Web projects (classes that use the EJBs), you must choose "Add project to an EAR" and specify same one

Example: EJB Interface

```
package coreservlets.bean;

import java.util.List;
import javax.ejb.*;

@Remote
public interface NumberListService {
  public List<Double> getNumbers(int size,
                                 double range);
}
```

Example: EJB Class that Implements Interface

```
@Stateless(mappedName="NumberListCreator")
public class NumberListServiceBean
       implements NumberListService {
  @EJB private NumberService numService;

  public List<Double> getNumbers(int size, double range) {
    List<Double> nums = new ArrayList<Double>();
    for(int i=0; i<size; i++) {
      nums.add(numService.getNumber(range));
    }
    return(nums);
  }
}
```

Remote client will do context.lookup("NumberListCreator") and cast it to NumberListService. That remote client will never directly use the instance of NumberService.

Declare the interface type (NumberService), not the concrete type (NumberServiceBean).

Example: Remote Client (Servlet)

```
public class LuckyNumberServlet2 extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    try {
      InitialContext context = new InitialContext();
      NumberListService service =
        (NumberListService)context.lookup("NumberListCreator");
      List<Double> nums = service.getNumbers(10, 50);
```

Looks up remote EJB the normal way. That remote EJB uses @EJB to access a local bean. If this Web app were always part of the same EAR on the same app server, then this Web app could also use @EJB to access beans. But having the Web app use InitialContext is more flexible because it allows for the possibility of the Web app later moving to another server without changing any code.

The mappedName from the @Stateless declaration. If you did not use a mappedName, then this client code would need to use a server-specific name. For JBoss it would be "NumberServiceListBean/remote" and for Glassfish it would be "coreservlets.bean.NumberServiceList".

Example: Remote Client (Servlet Continued)

```
response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String docType =
       "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
       "Transitional//EN\">\n";
     out.println(docType +
                 "<HTML>\n" +
                 "<HEAD><TITLE>Your Lucky Numbers</TITLE></HEAD>\n" +
                 "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                 "<H1>Your Lucky Numbers</H1>\n" +
                 "(" + getServletContext().getServerInfo() + ")\n" +
                 "<UL>");
     for(double num: nums) {
       out.printf("  <LI>%5.3f\n", num);
     }
     out.println("</UL>\n</BODY></HTML>");
   } catch(NamingException ne) {
     System.out.println("Error: " + ne);
   }
 }
```


Remote Client: jndi.properties

- **For JBoss**

java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost:1099

- **For Glassfish**

org.omg.CORBA.ORBInitialHost=localhost

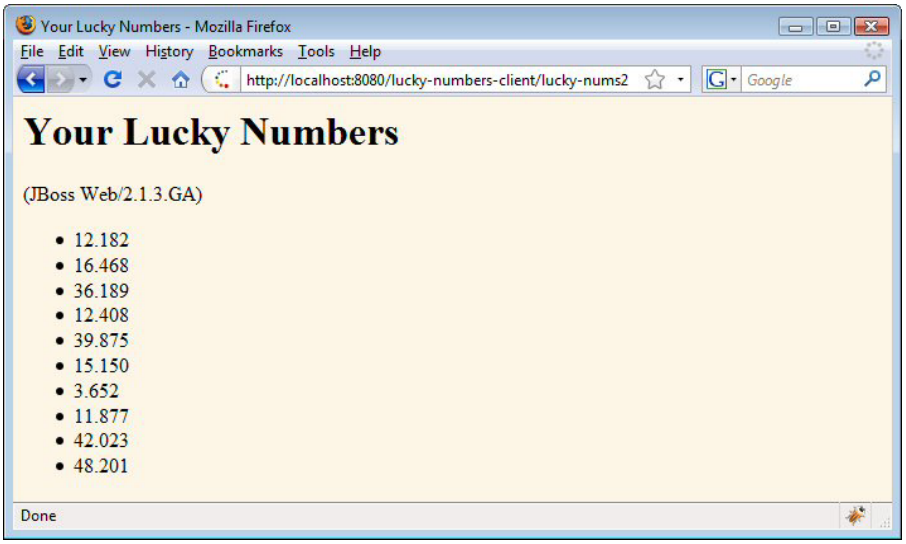
Change this hostname if app server is on different host than client.

- **Notes**
  - Put in “src” folder in Eclipse to be sure it is in classpath
  - The Eclipse project you can download has both versions. Just copy/rename jboss-jndi.properties or glassfish-jndi.properties.

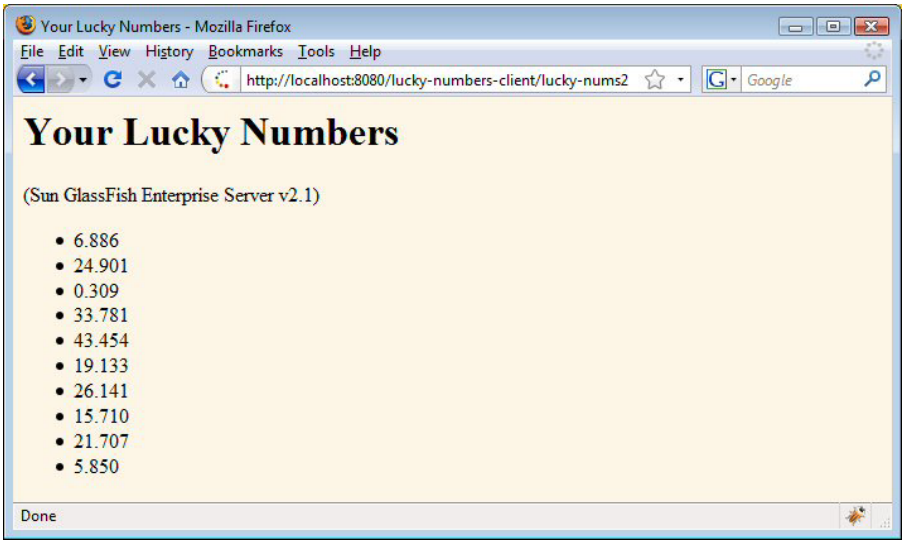
Example: Remote Client (web.xml)

```
…
 <servlet>
    <servlet-name>
       Servlet with Number List from EJB
    </servlet-name>
    <servlet-class>
      coreservlets.client.LuckyNumberServlet2
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      Servlet with Number List from EJB
    </servlet-name>
    <url-pattern>/lucky-nums2</url-pattern>
  </servlet-mapping>
…
```


Example: Results (JBoss 5)



Example: Results (Glassfish 2.1)



Stateful Session Beans

Stateful Session Beans: Idea

- **POJOs**
  - Ordinary Java classes; no special interfaces or parent classes. Often provide some sort of service that stores information (temporarily) as well as sending it.
- **Local or remote access**
  - Can be accessed either on local app server (same machine as app that uses the beans) or remote app server (different machine from app that uses the beans)
- **Maintain client state**
  - You can configure the bean with client-specific information, then access that info (without fear of race conditions) until you call special method marked with @Remove.

Stateful Session Beans: Approach

- **Make new EJB project**
  - File → New → EJB Project
- **Define an interface**
  - Mark with @Remote
    - For access from other servers or from projects on same server
  - Mark with @Local
    - To only allow access from projects on same server (default)
- **Create a class that implements interface**
  - Mark with @Stateful for server's default JNDI mapping
  - Mark with @Stateful(mappedName="SomeJndiName")
  - Mark a method with @Remove
    - When called, server terminates session and gc's instance
- **Deploy to app server**
  - R-click server, Add/Remove Projects, start server

Example: EJB Interface

```
package coreservlets.bean;

import java.util.List;
import javax.ejb.*;

@Remote
public interface FancyList {
  public void initializeNumbers(int size,
                                double range);
  public List<Double> getNumbers();
  public double getSum();
  public double getSmallest();
  public double getLargest();
  public void removeList();
}
```


Example: EJB Class that Implements Interface

```
@Stateful(mappedName="cool-number-list")
public class FancyListBean implements FancyList {
  @EJB private NumberListService service;
  private List<Double> nums = new ArrayList<Double>();

  public void initializeNumbers(int size, double range) {
    nums = service.getNumbers(size, range);
  }

  public List<Double> getNumbers() {
    return(nums);
  }

  public double getSum() {
    double sum = 0;
    for(double num: nums) {
      sum = sum + num;
    }
    return(sum);
  }
```

Remote client will do context.lookup("cool-number-list") and cast it to FancyList.

Client will call this method first to set up the state of the bean. Then the client will repeatedly access the get*Blah* methods to access information related to the state. When done, the client will call removeList (next slide).

Example: EJB Class that Implements Interface (Cont.)

```
 public double getSmallest() {
    return(Collections.min(nums));
  }

  public double getLargest() {
    return(Collections.max(nums));
  }

  @Remove
  public void removeList() {
    nums = new ArrayList<Double>();
  }
}
```

When client calls this method, session is broken and bean instance is eligible for garbage collection.

Example: Remote Client (Servlet)

```
public class FancyListServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    int size = 10;
    double range = 50.0;
    String address;
    try {
      size = Integer.parseInt(request.getParameter("size"));
      range = Double.parseDouble(request.getParameter("range"));
    } catch(Exception e) {}
    try {
      InitialContext context = new InitialContext();
      FancyList fancyList =
        (FancyList)context.lookup("cool-number-list");
      fancyList.initializeNumbers(size, range);
```

Sets up state that will be used in the JSP page.

Example: Remote Client (Servlet Continued)

```
      request.setAttribute("fancyList", fancyList);
      address = "/WEB-INF/results/show-fancy-list.jsp";
      RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
      dispatcher.forward(request, response);
      fancyList.removeList();
    } catch(NamingException ne) {
      address = "/WEB-INF/results/error-page.jsp";
      RequestDispatcher dispatcher =
        request.getRequestDispatcher(address);
      dispatcher.forward(request, response);
    }
  }
}
```

JSP page (MVC approach) uses state of the EJB.

Tells remote app server we are done with stateful EJB.

Example: Remote Client (JSP)

```
…
<h2>General Info</h2>
<ul>
  <li>Smallest: ${fancyList.smallest}</li>
  <li>Largest: ${fancyList.largest}</li>
  <li>Sum: ${fancyList.sum}</li>
  <li>Server: ${pageContext.servletContext.serverInfo}</li>
</ul>
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>
<h2>Specific Numbers</h2>
<ol>
<c:forEach var="number" items="${fancyList.numbers}">
  <li>${number}</li>
</c:forEach>
</ol>
</body></html>
```

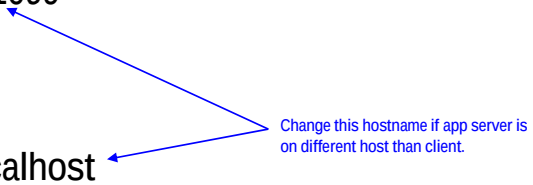
Example: Remote Client (jndi.properties)

- **For JBoss**

java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost:1099

- **For Glassfish**

org.omg.CORBA.ORBInitialHost=localhost



- **Notes**
  - Put in "src" folder in Eclipse to be sure it is in classpath
  - The Eclipse project you can download has both versions. Just copy/rename jboss-jndi.properties or glassfish-jndi.properties.

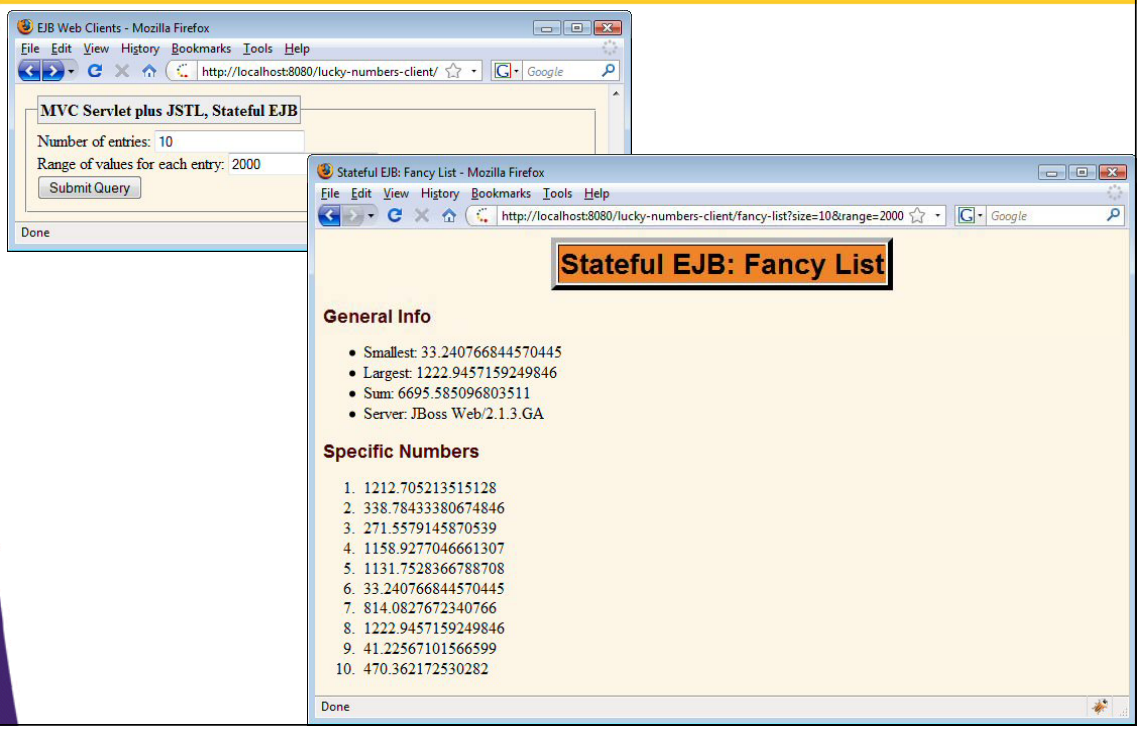
Example: Remote Client (web.xml)

```
…
 <servlet>
    <servlet-name>
      MVC Servlet with Stateful EJB
    </servlet-name>
    <servlet-class>
      coreservlets.client. FancyListServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      MVC Servlet with Stateful EJB
    </servlet-name>
    <url-pattern>/fancy-list</url-pattern>
  </servlet-mapping>
…
```

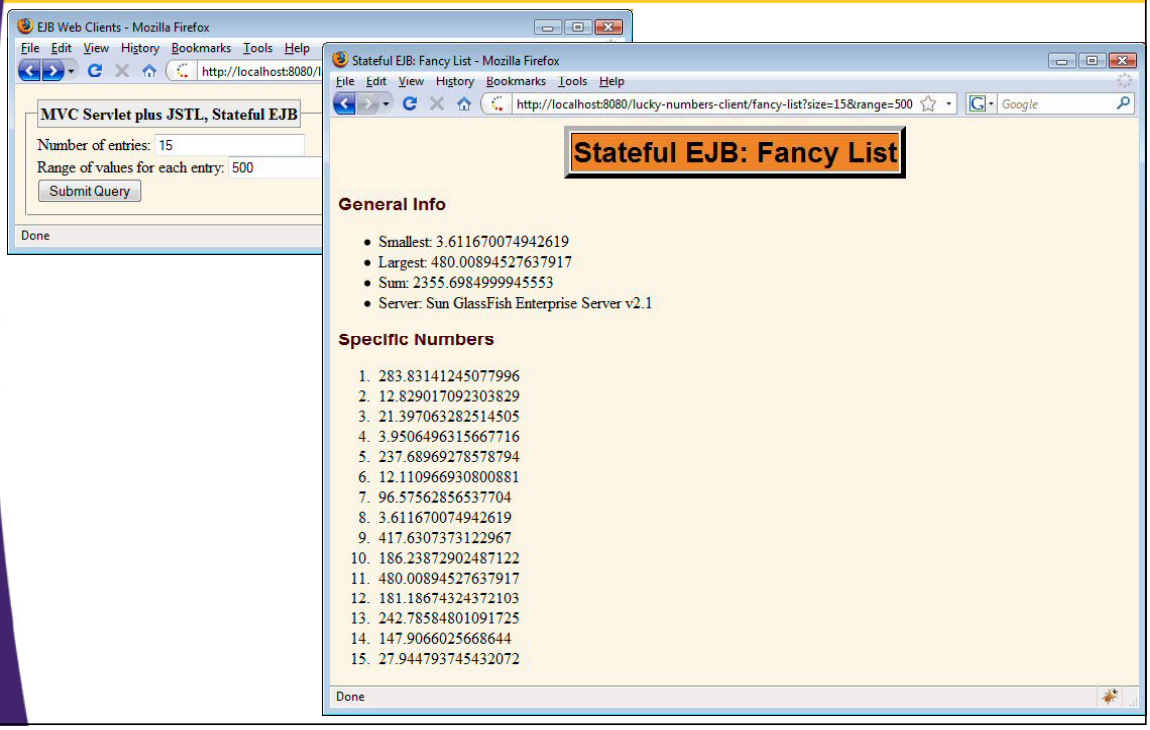
Example: Remote Client (Input Form)

```
…
<fieldset>
  <legend>MVC Servlet plus JSTL, Stateful EJB</legend>
  <form action="fancy-list">
    <label>Number of entries:
           <input type="text" name="size"/></label><br/>
    <label>Range of values for each entry:
           <input type="text" name="range"/></label><br/>
    <input type="submit"/>
  </form>
</fieldset>
…
```


Example: Remote Client Results (JBoss 5)



Example: Remote Client Results (Glassfish 2.1)



Wrap-up

Books and References

- ***EJB 3 in Action***
  - Debu Panda, Reza Rahman, and Derek Lane (Manning)
- ***Enterprise JavaBeans 3.0* (5th Edition)**
  - Bill Burke and Richard Monson-Haefel (O'Reilly)
- **Java EE 5 Tutorial (EJB section)**
  - http://java.sun.com/javaee/5/docs/tutorial/doc/bnblr.html
- **JBoss EJB3 Tutorial**
  - http://docs.jboss.com/ejb3/app-server/tutorial/
- **Glassfish EJB FAQ**
  - https://glassfish.dev.java.net/javaee5/ejb/EJB_FAQ.html
- **Cay Horstmann's EJB Tutorial**
  - http://www.horstmann.com/elvis/elvis-meets-glassfish.html

Summary

- **Stateless session beans**
  - Interface: mark with @Remote
  - Class: mark with @Stateless(mappedName="blah")
- **Stateful session beans**
  - Mark class with @Stateful instead of @Stateless
  - Mark a method with @Remove
- **Session bean clients**

```
InitialContext context = new InitialContext();
InterfaceType var = (InterfaceType)context.lookup("blah");
var.someMethod(args);
```

  - For stateful beans, call specially marked method when done
  - Need jndi.properties specific to server type
- **Local access to beans**

```
@EJB private InterfaceType var;
```

Questions?