Core Servlets and JavaServer Pages / 2e

Volume 1: Core Technologies

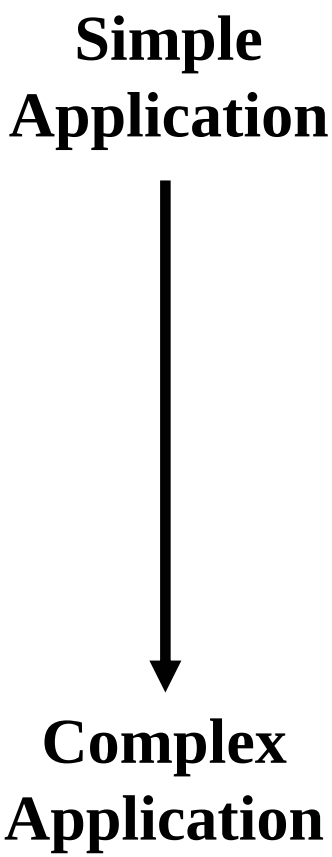
Marty Hall •Larry Brown

Using JavaBeans Components in JSP Documents

Agenda

- Understanding the benefits of beans
- Creating beans
- Installing bean classes on your server
- Accessing bean properties
- Explicitly setting bean properties
- Automatically setting bean properties from request parameters
- Sharing beans among multiple servlets and JSP pages

Uses of JSP Constructs



- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC)
- MVC with JSP expression language
- Custom tags

Background: What Are Beans?

- **Java classes that follow certain conventions**
  - ***Must have a zero-argument (empty) constructor***
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
  - ***Should have no public instance variables (fields)***
    - Always good advice to follow this practice and use accessor methods instead of allowing direct access to fields
  - ***Persistent values should be accessed through methods called getXxx and setXxx***
    - If class has method getTitle that returns a String, class is said to have a String *property* named title
    - Boolean properties use is*Xxx* instead of get*Xxx*
  - For more on beans, see http://java.sun.com/beans/docs/

Why You Should Use Accessors, Not Public Fields

- To be a bean, you cannot have public fields
- So, you should replace

```
public double speed;
```

- with

```
private double speed;

public double getSpeed() {
  return(speed);
}
public void setSpeed(double newSpeed) {
  speed = newSpeed;
}
```

- You should do this in *all* your Java code anyhow. Why?

Why You Should Use Accessors, Not Public Fields

- **1) You can put constraints on values**

```
public void setSpeed(double newSpeed) {
  if (newSpeed < 0) {
    sendErrorMessage(...);
    newSpeed = Math.abs(newSpeed);
  }
  speed = newSpeed;
}
```

  - If users of your class accessed the fields directly, then they would each be responsible for checking constraints.

Why You Should Use Accessors, Not Public Fields

- **2) You can change your internal representation without changing interface**

```
// Now using metric units (kph, not mph)

public void setSpeed(double newSpeed) {
  setSpeedInKPH = convert(newSpeed);
}

public void setSpeedInKPH(double newSpeed) {
  speedInKPH = newSpeed;
}
```

Why You Should Use Accessors, Not Public Fields

- **3) You can perform arbitrary side effects**

```
public double setSpeed(double newSpeed) {
  speed = newSpeed;
  updateSpeedometerDisplay();
}
```

  - If users of your class accessed the fields directly, then they would each be responsible for executing side effects. Too much work and runs huge risk of having display inconsistent from actual values.

Using Beans: Basic Tasks

- **jsp:useBean**
  - In the simplest case, this element builds a new bean.

```
<jsp:useBean id="beanName" class="package.Class" />
```

- **jsp:getProperty**
  - This element reads and outputs the value of a bean property i.e., it calls a method of the form getXxx

```
<jsp:getProperty name="beanName" property="propertyName"/>
```

- **jsp:setProperty**
  - This element modifies a bean property (i.e., calls a method of the form set*Xxx*).

```
<jsp:setProperty name="beanName“
    property="propertyName" value="propertyValue" />
```

Building Beans: jsp:useBean

- **Format**

```
<jsp:useBean id="name" class="package.Class" />
```

- **Purpose**
  - Allow instantiation of Java classes without explicit Java programming
    - Instantiate an object of the class package.Class and bind it to a variable in _jspService with the name specified by id
    - You must use the fully qualified class name even if you use a <%page import …%>
- **Notes**
  - Simple interpretation:

```
<jsp:useBean id="book1" class="coreservlets.Book" />
```

    can be thought of as equivalent to the scriptlet

```
<% coreservlets.Book book1 = new coreservlets.Book(); %>
```

  - But jsp:useBean has two additional advantages:
    - It is easier to derive object values from request parameters
    - It is easier to share objects among pages or servlets

Accessing Bean Properties: jsp:getProperty

- **Format**

```
<jsp:getProperty name="name" property="property" />
```

- **Purpose**
  - Allow access to bean properties (i.e., calls to get*Xxx* methods) without explicit Java programming
- **Notes**
  - `<jsp:getProperty name="book1" property="title" />`

    is equivalent to the following JSP expression

```
<%= book1.getTitle() %>
```

Setting Simple Bean Properties: jsp:setProperty

- **Format**

```
<jsp:setProperty name="name"
     property="property"  value="value" />
```

- **Purpose**
  - Allow setting of bean properties (i.e., calls to set*Xxx* methods) without explicit Java programming
- **Notes**
  -
```
<jsp:setProperty name="book1"
     property="title"
     value="Core Servlets and JavaServer Pages" />
```
    is equivalent to the following scriptlet
```
<% book1.setTitle("Core Servlets and JavaServer Pages"); %>
```

Example: StringBean

```
package coreservlets;
public class StringBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }

  public void setMessage(String message) {
    this.message = message;
  }
}
```

- Since `StringBean` has a method called `getmessage` that returns a `String` and a method `setMessage` that takes a `String` as an argument, the class is said to have a `String` property called message
- Beans installed in normal Java directory
  - …/WEB-INF/classes/*directoryMatchingPackageName*
- Beans (and utility classes) must *always* be in packages!

JSP Page That Uses StringBean

```
<jsp:useBean id="stringBean"
             class="coreservlets.StringBean" />
<OL>
<LI>Initial value (from jsp:getProperty):
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>
<LI>Initial value (from JSP expression):
    <I><%= stringBean.getMessage() %></I>
<LI><jsp:setProperty name="stringBean"
                     property="message"
                     value="Best string bean: Fortex" />
    Value after setting property with jsp:setProperty:
    <I><jsp:getProperty name="stringBean"
                        property="message" /></I>
<LI>
<% stringBean.setMessage("My favorite:Kentucky Wonder"); %>
    Value after setting property with scriptlet:
    <I><%= stringBean.getMessage() %></I>
</OL>
```

JSP Page That Uses StringBean

Setting Bean Properties
Case 1:Explicit Conversion & Assignment

- **The value attribute of `jsp:setProperty` is allowed to be a request time expression**

```
<!DOCTYPE ...>
...
<jsp:useBean id="entry"
    class="coreservlets.SaleEntry" />

<%-- setItemID expects a String --%>

<jsp:setProperty
    name="entry"
    property="itemID"
    value='<%= request.getParameter("itemID") %>' />
```

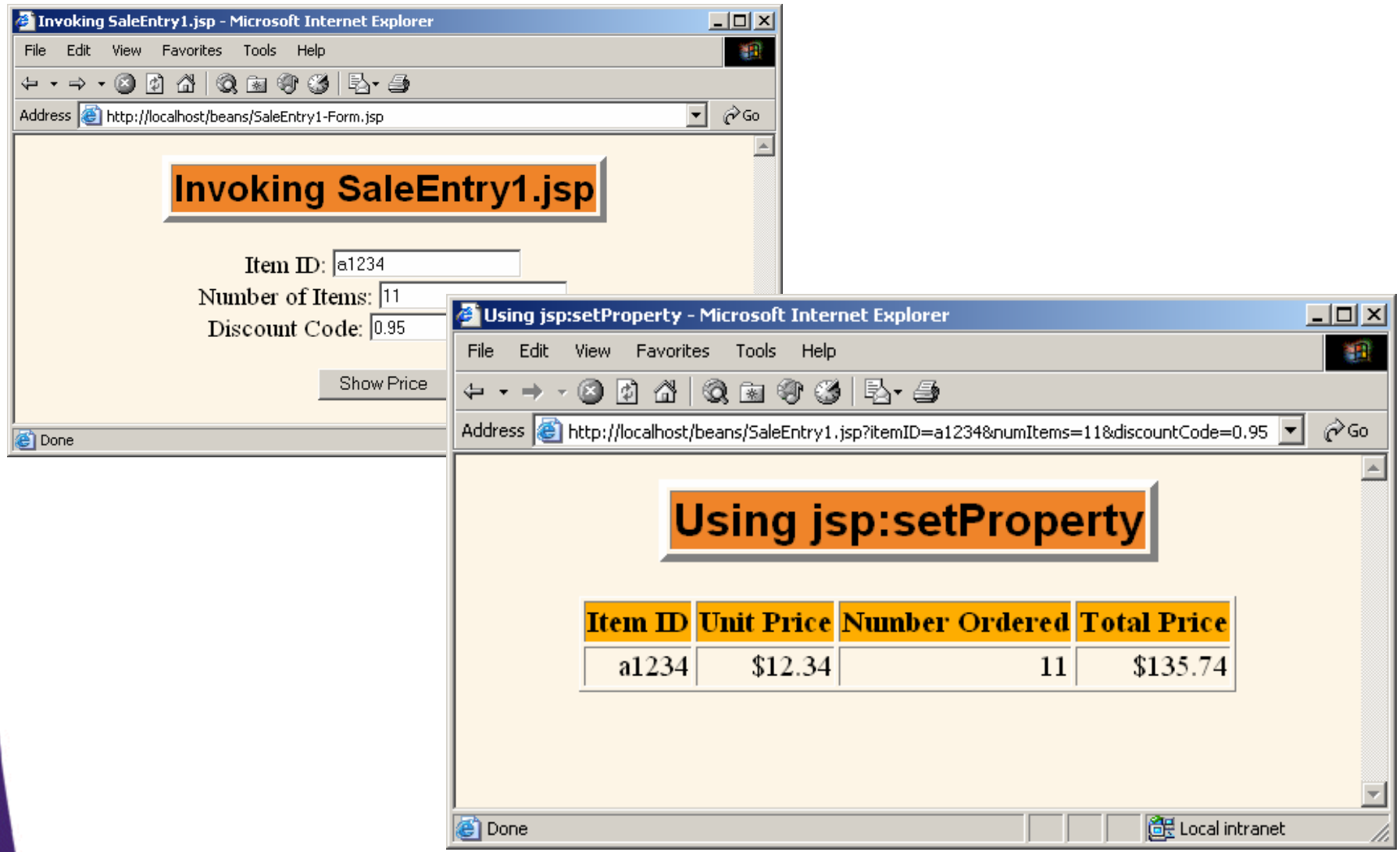
Setting Bean Properties
Case 1:Explicit Conversion & Assignment

```
<%
int numItemsOrdered = 1;
try {
  numItemsOrdered =
    Integer.parseInt(request.getParameter("numItems"));
} catch(NumberFormatException nfe) {}
%>

<%-- setNumItems expects an int --%>

<jsp:setProperty
    name="entry"
    property="numItems"
    value="<%= numItemsOrdered %>" />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment

```
<%
double discountCode = 1.0;
try {
  String discountString =
    request.getParameter("discountCode");
  discountCode =
    Double.parseDouble(discountString);
} catch(NumberFormatException nfe) {}
%>

<%-- setDiscountCode expects a double --%>

<jsp:setProperty
    name="entry"
    property="discountCode"
    value="<%= discountCode %>" />
```

Setting Bean Properties Case 1: Explicit Conversion & Assignment



Case 2: Associating Individual Properties with Input Parameters

- **Use the param (not the value) attribute of jsp:setProperty to indicate that**
  - Value should come from specified request parameter
  - Simple automatic type conversion should be performed for properties that expect values of standard types
    - boolean, Boolean, byte, Byte, char, Character, double, Double, int, Integer, float, Float, long, or Long
    - If value entered is of the wrong type an exception is raised and program terminates

Case 2: Associating Individual Properties with Input Parameters

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />

<jsp:setProperty
 name="entry" property="itemID" param="itemID" />

<jsp:setProperty
  name="entry" property="numItems"  param="numItems" />

<jsp:setProperty name="entry"
    property="discountCode" param="discountCode" />
```

If the request parameter name and the bean property name are the same, you can omit param

```
<jsp:setProperty name="entry" property="numItems" />
```

Case 3: Associating All Properties with Input Parameters

- **Use "*" for the value of the property attribute of jsp:setProperty to indicate that**
  - Value should come from request parameter whose name matches property name
  - Simple automatic type conversion should be performed

Case 3: Associating All Properties with Input Parameters

```
<jsp:useBean id="entry"
             class="coreservlets.SaleEntry" />
<jsp:setProperty name="entry" property="*" />
```

- **This is extremely convenient for making "form beans" -- objects whose properties are filled in from a form submission.**
  - You can even divide the process up across multiple forms, where each submission fills in part of the object.
- **Caveats:**
  - No action is taken when an input parameter is missing
  - Automatic type conversion does not guard against illegal values
  - Bean property names and request parameters are case sensitive

Sharing Beans

- **You can use the `scope` attribute to specify additional places where bean is stored**
  - Still also bound to local variable in _jspService
  - `<jsp:useBean id="…" class="…" scope="…" />`
- **Lets multiple servlets or JSP pages share data**
- **Also permits conditional bean creation**
  - Creates new object *only* if it can't find existing one

Values of the scope Attribute

- **`page`: (`<jsp:useBean … scope="page"/>` or `<jsp:useBean…>`)**
  - Default value. Bean object is bound to a local variable and should be placed in the **`PageContext`** object for the duration of the current request. Servlet code accesses it by calling **`getAttribute`** on the predefined **`pageContext`** variable. ***Lets methods in same servlet access the same bean***
- **`request`:(`<jsp:useBean … scope="request"/>`)**
  - Bean object should be placed in the **`ServletRequest`** object for the duration of the current request, where it is available by means of **`getAttribute`**
  - Two JSP pages or a JSP page and servlet will share request objects when use jsp:include

Values of the scope Attribute

- **`session: (<jsp:useBean … scope="session"/>)`**
  - Bean will be stored in the HttpSession object associated with the current request, where it can be accessed from regular servlet code with getAttribute and setAttribute, as with normal session objects.

- **`application:(<jsp:useBean …scope="application"/>)`**
  - Bean will be stored in ServletContext (available through the application variable or by call to getServletContext()). ServletContext is ***shared by all servlets in the same Web application*** (or all servlets on server if no explicit Web applications are defined).

Conditional Bean Operations

- **Bean conditionally created**
  - **`jsp:useBean`** results in new bean being instantiated only if no bean with **same id and scope** can be found.
  - If a bean with same id and scope is found, the preexisting bean is simply bound to variable referenced by id.
- **Bean properties conditionally set**
  - **`<jsp:useBean ... />`** replaced by **`<jsp:useBean...>statements</jsp:useBean>`**
  - The statements (jsp:setProperty elements) are executed *only* if a new bean is created, not if an existing bean is found.

Conditional Bean Creation: AccessCountBean

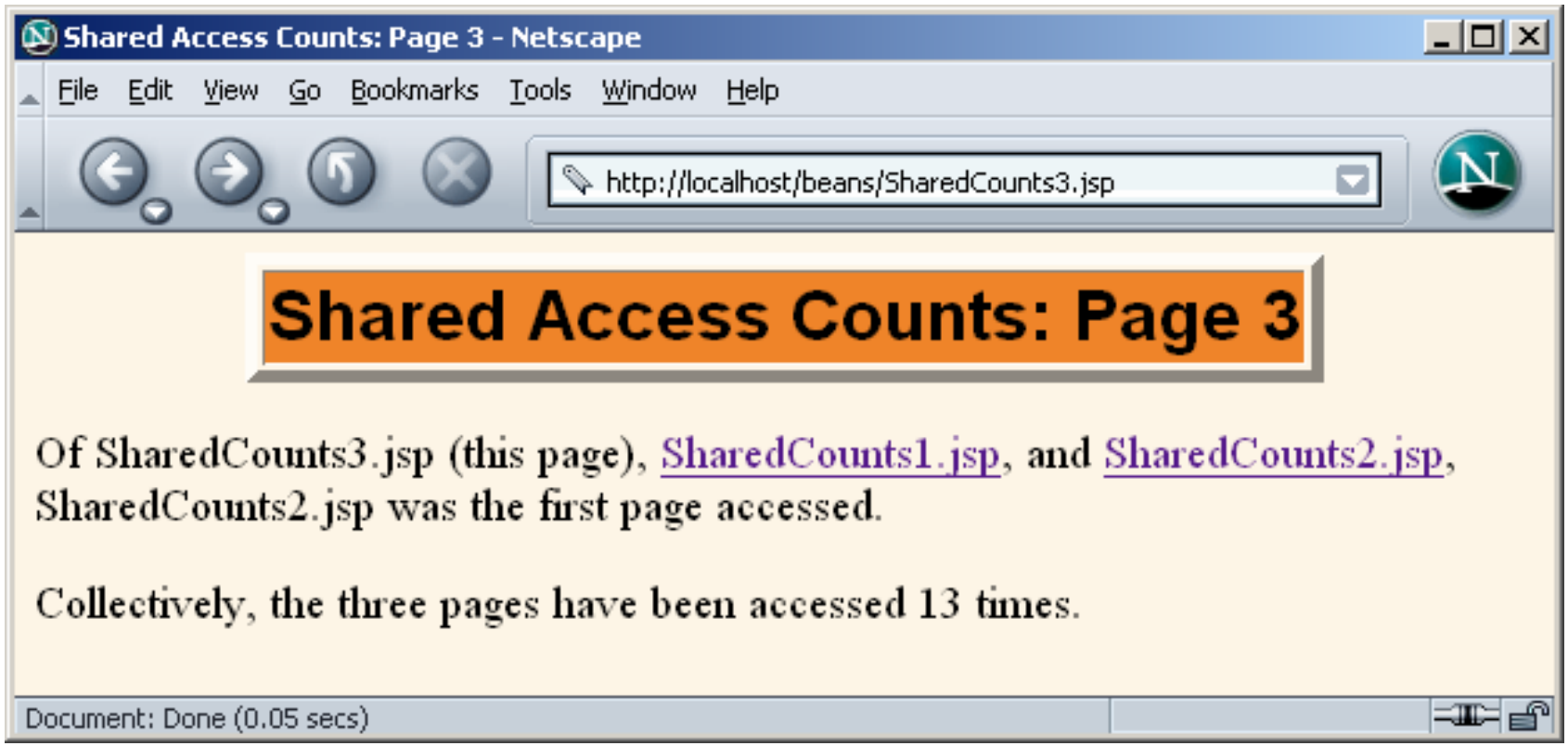
```
public class AccessCountBean {
  private String firstPage;
  private int accessCount = 1;

  public String getFirstPage() {
    return(firstPage);
  }

  public void setFirstPage(String firstPage) {
    this.firstPage = firstPage;
  }

  public int getAccessCount() {
    return(accessCount);
  }

  public void setAccessCountIncrement(int increment) {
    accessCount = accessCount + increment;
  }
}
```

Conditional Bean Creation: SharedCounts1.jsp

```
<jsp:useBean id="counter"
             class="coreservlets.AccessCountBean"
             scope="application">
<jsp:setProperty name="counter"
             property="firstPage"
             value="SharedCounts1.jsp" />
</jsp:useBean>
Of SharedCounts1.jsp (this page),
<A HREF="SharedCounts2.jsp">SharedCounts2.jsp</A>, and
<A HREF="SharedCounts3.jsp">SharedCounts3.jsp</A>,
<jsp:getProperty name="counter" property="firstPage" />
was the first page accessed.
<P>
Collectively, the three pages have been accessed
<jsp:getProperty name="counter" property="accessCount" />
times.
<jsp:setProperty name="counter"
                 property="accessCountIncrement"
                 value="1" />
```

Accessing SharedCounts1, SharedCounts2, SharedCounts3

- SharedCounts2.jsp was accessed first.
- Pages have been accessed twelve previous times by an arbitrary number of clients



Sharing Beans in Four Different Ways

- **Using unshared (page-scoped) beans.**
- **Sharing request-scoped beans**
  - Servlet stores data in the HttServletRequest where they are accessible only to the destination JSP page
- **Sharing session-scoped beans**
  - Data is stored for each client in HttpSession
- **Sharing application-scoped beans**
  - Data is shared among multiple clients via ServletContext
  - Data is accessible to the same client in the destination page or in other pages

- **Note:**
  - Use different names (i.e., id in jsp:useBean) for different beans

Sharing Beans Four Ways: Bean Code

```
package coreservlets;

public class BakedBean {
  private String level = "half-baked";
  private String goesWith = "hot dogs";

  public String getLevel() {
    return(level);
  }
  public void setLevel(String newLevel) {
    level = newLevel;
  }
  public String getGoesWith() {
    return(goesWith);
  }
  public void setGoesWith(String dish) {
    goesWith = dish;
  }
}
```

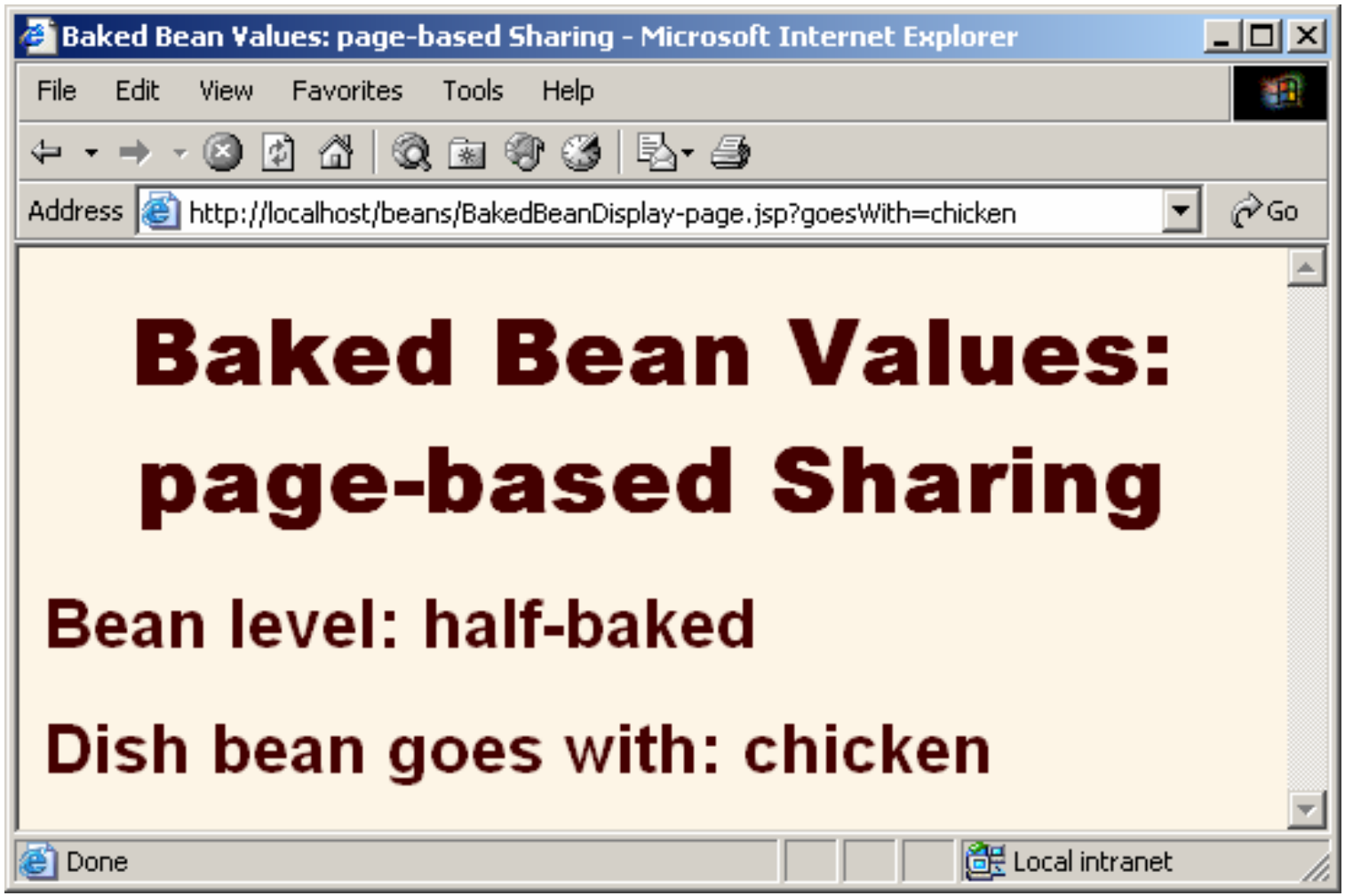
Sharing Beans Example 1: Page-Scoped (Unshared)

- **Create the bean**
  - Use jsp:useBean with scope="page" (or no scope at all, since page is the default).
- **Modify the bean**
  - Use jsp:setProperty with property="*".
  - Then, supply request parameters that match the bean property names.
- **Access the bean**
  - Use jsp:getProperty.

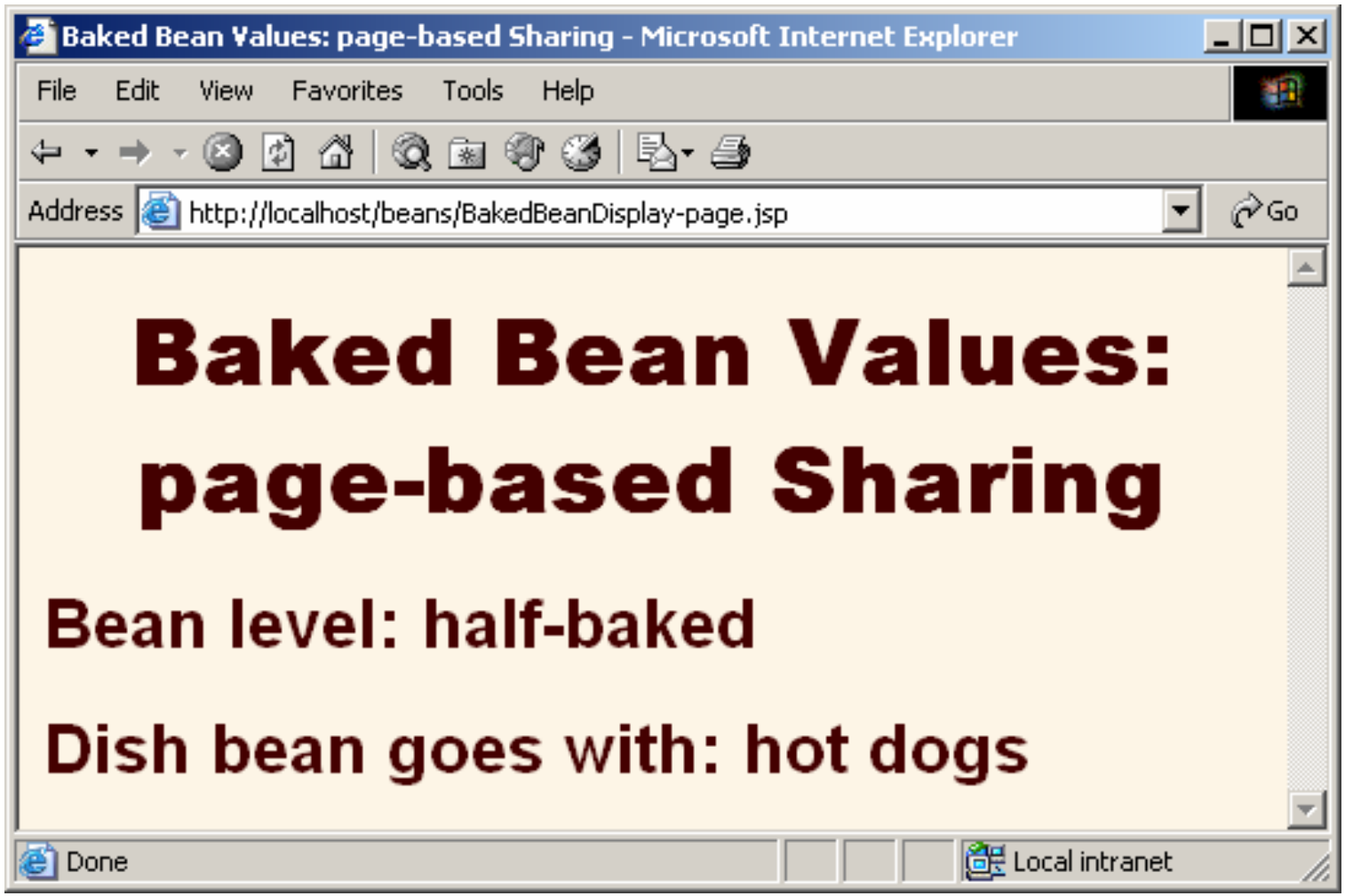
Sharing Beans Example 1: Page-Scoped (Unshared)

```
…
<BODY>
<H1>Baked Bean Values: page-based Sharing</H1>
<jsp:useBean id="pageBean"
             class="coreservlets.BakedBean" />
<jsp:setProperty name="pageBean" property="*" />
<H2>Bean level:
<jsp:getProperty name="pageBean"
                 property="level" />
</H2>
<H2>Dish bean goes with:
<jsp:getProperty name="pageBean"
                 property="goesWith" />
</H2>
</BODY></HTML>
```

Sharing Beans Example 1: Result (Initial Request)



Sharing Beans Example 1: Result (Later Request)



Sharing Beans Example 2: Request-Based Sharing

- **Create the bean**
  - Use jsp:useBean with scope="request".
- **Modify the bean**
  - Use jsp:setProperty with property="*".
  - Then, supply request parameters that match the bean property names.
- **Access the bean in the 1st (main) page**
  - Use jsp:getProperty.
  - Then, use jsp:include to invoke the second page.
- **Access the bean in the 2nd (included) page**
  - Use jsp:useBean with the same id as on the first page, again with scope="request".
  - Then, use jsp:getProperty.

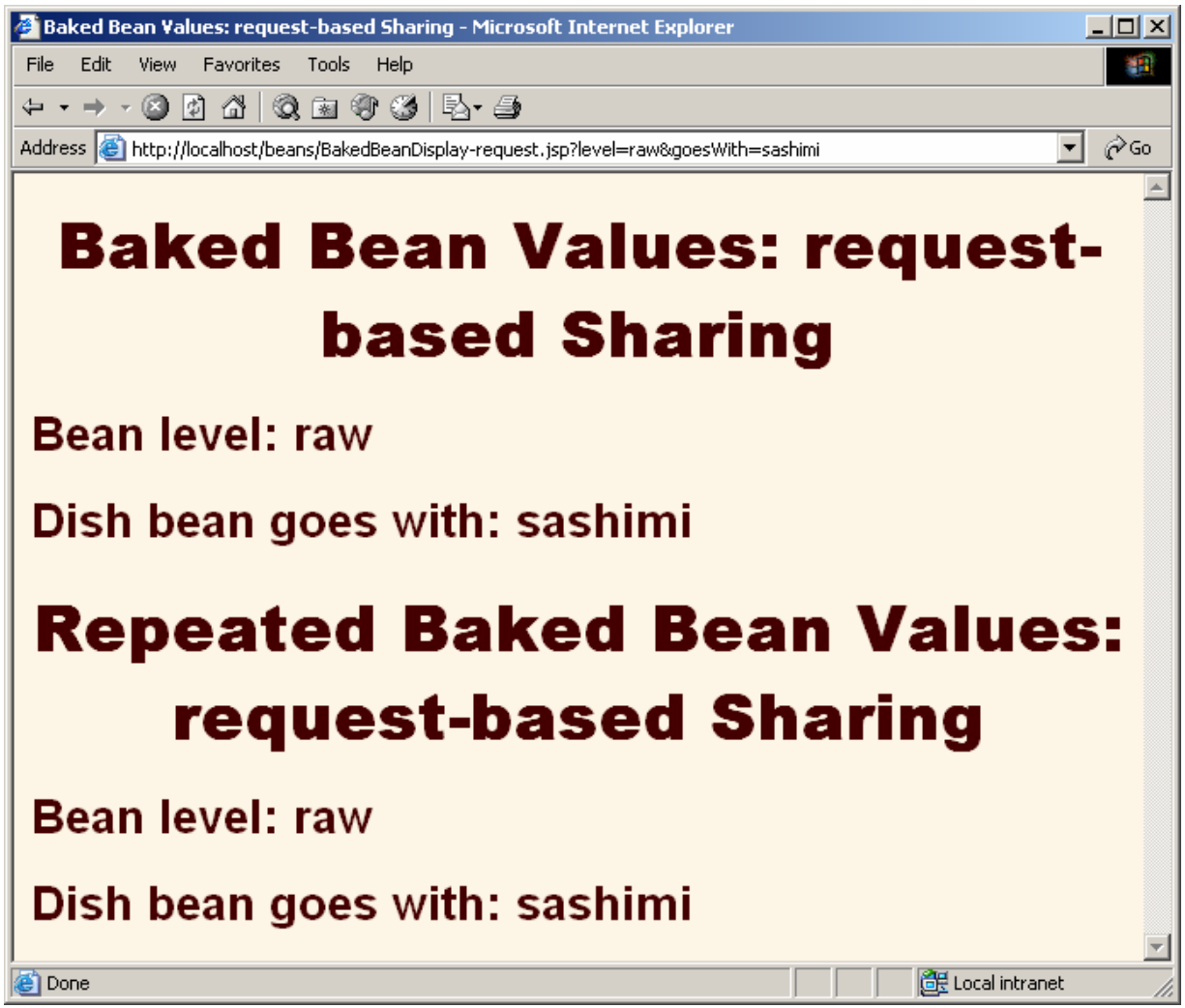
Request-Based Sharing: Code for Main Page

```
…
<BODY>
<H1>Baked Bean Values: request-based Sharing</H1>
<jsp:useBean id="requestBean"
             class="coreservlets.BakedBean"
             scope="request" />
<jsp:setProperty name="requestBean"
                 property="*" />
<H2>Bean level:
<jsp:getProperty name="requestBean"
                 property="level" /></H2>
<H2>Dish bean goes with:
<jsp:getProperty name="requestBean"
                 property="goesWith" /></H2>
<jsp:include page="BakedBeanDisplay-snippet.jsp" />
</BODY></HTML>
```

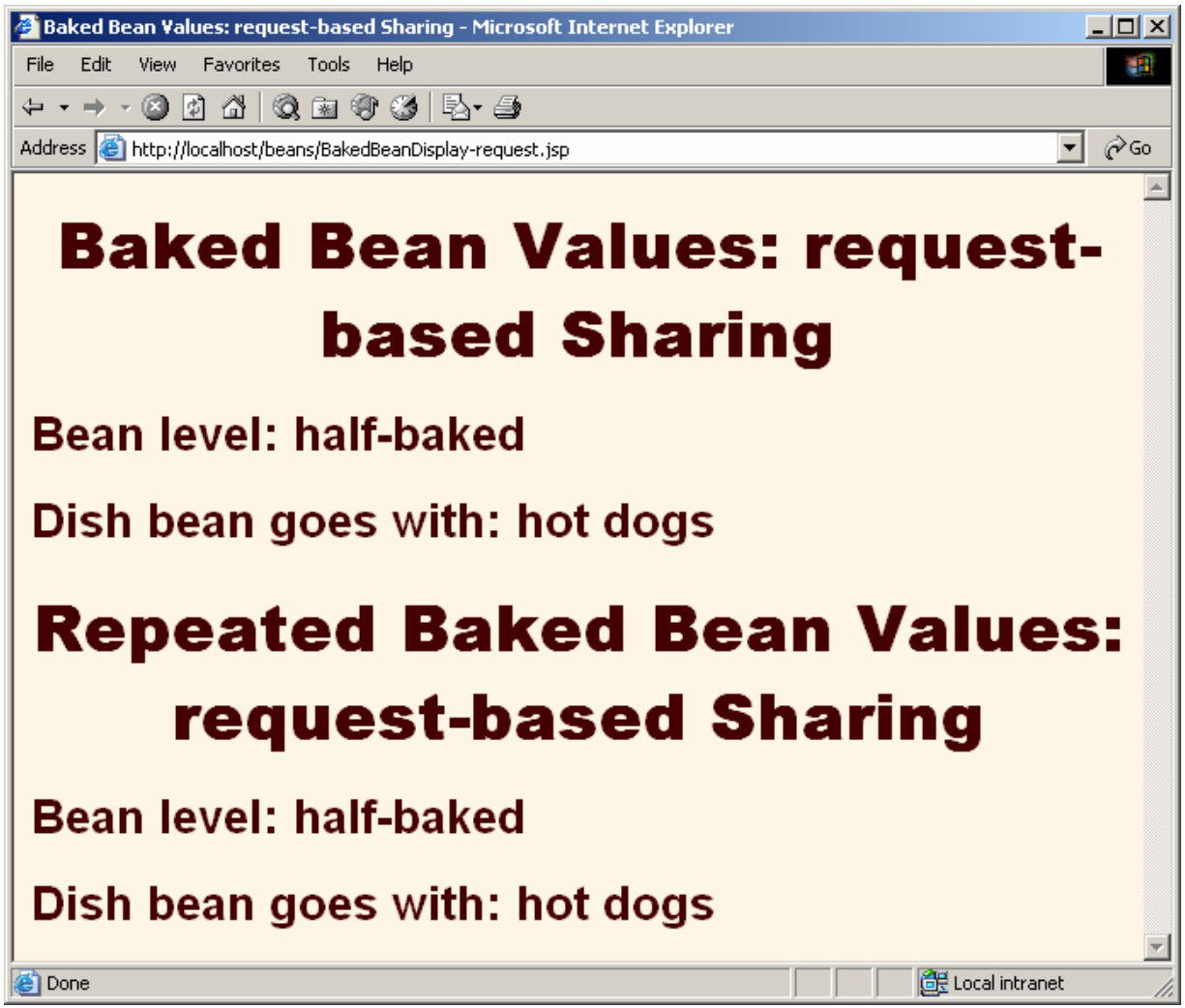
Request-Based Sharing: Code for Included Page

```
<H1>Repeated Baked Bean Values:
request-based Sharing</H1>
<jsp:useBean id="requestBean"
             class="coreservlets.BakedBean"
             scope="request" />
<H2>Bean level:
<jsp:getProperty name="requestBean"
                 property="level" />
</H2>
<H2>Dish bean goes with:
<jsp:getProperty name="requestBean"
                 property="goesWith" />
</H2>
```

Request-Based Sharing: Result (Initial Request)



Request-Based Sharing: Result (Later Request)



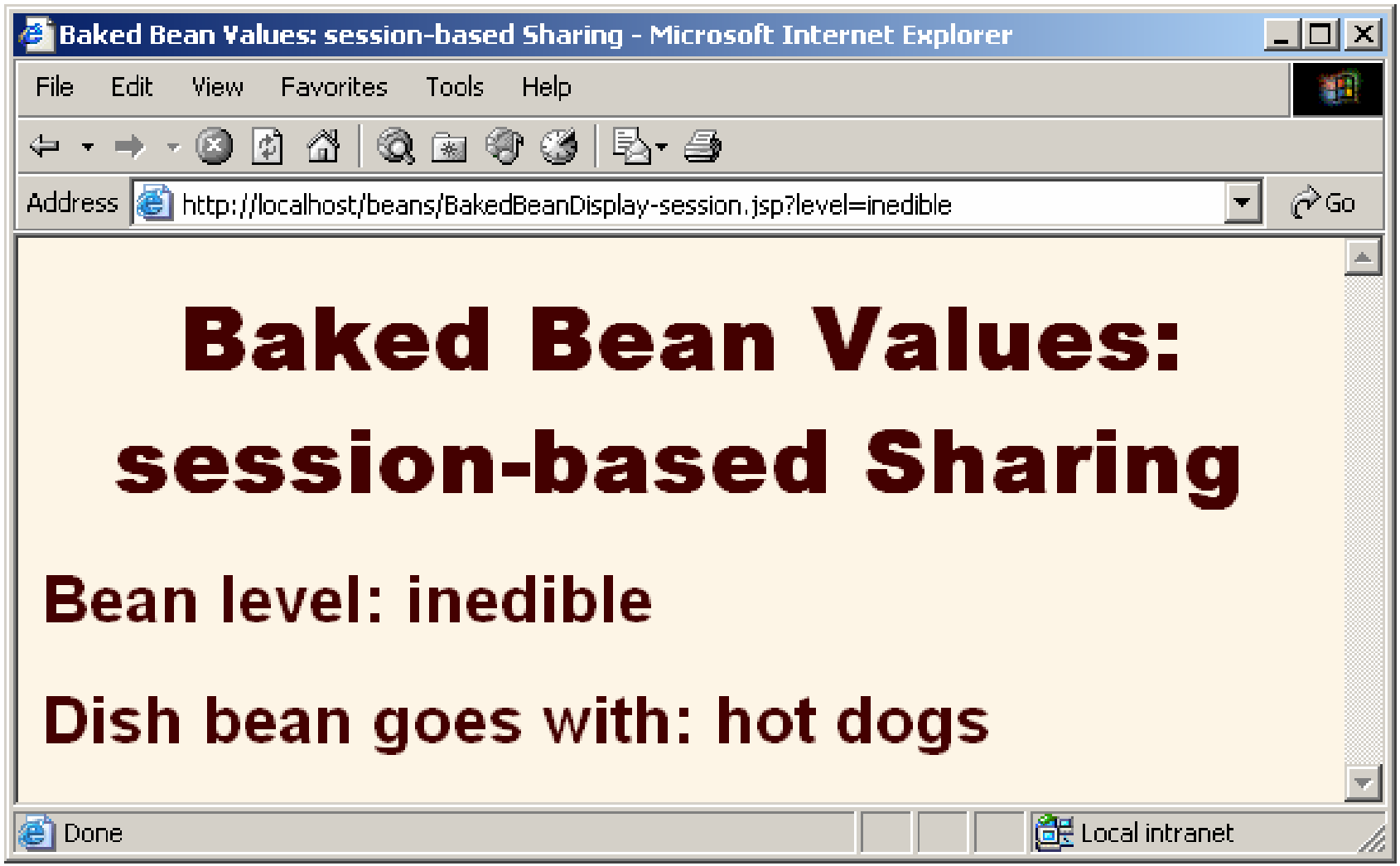
Sharing Beans Example 3: Session-Based Sharing

- **Create the bean**
  - Use jsp:useBean with scope="session".
- **Modify the bean**
  - Use jsp:setProperty with property="*".
  - Then, supply request parameters that match the bean property names.
- **Access the bean in the initial request**
  - Use jsp:getProperty in the request in which jsp:setProperty is invoked.
- **Access the bean later**
  - Use jsp:getProperty in a request that does not include request parameters and thus does not invoke jsp:setProperty. If this request is from the same client (within the session timeout), the previously modified value is seen. If this request is from a different client (or after the session timeout), a newly created bean is seen.

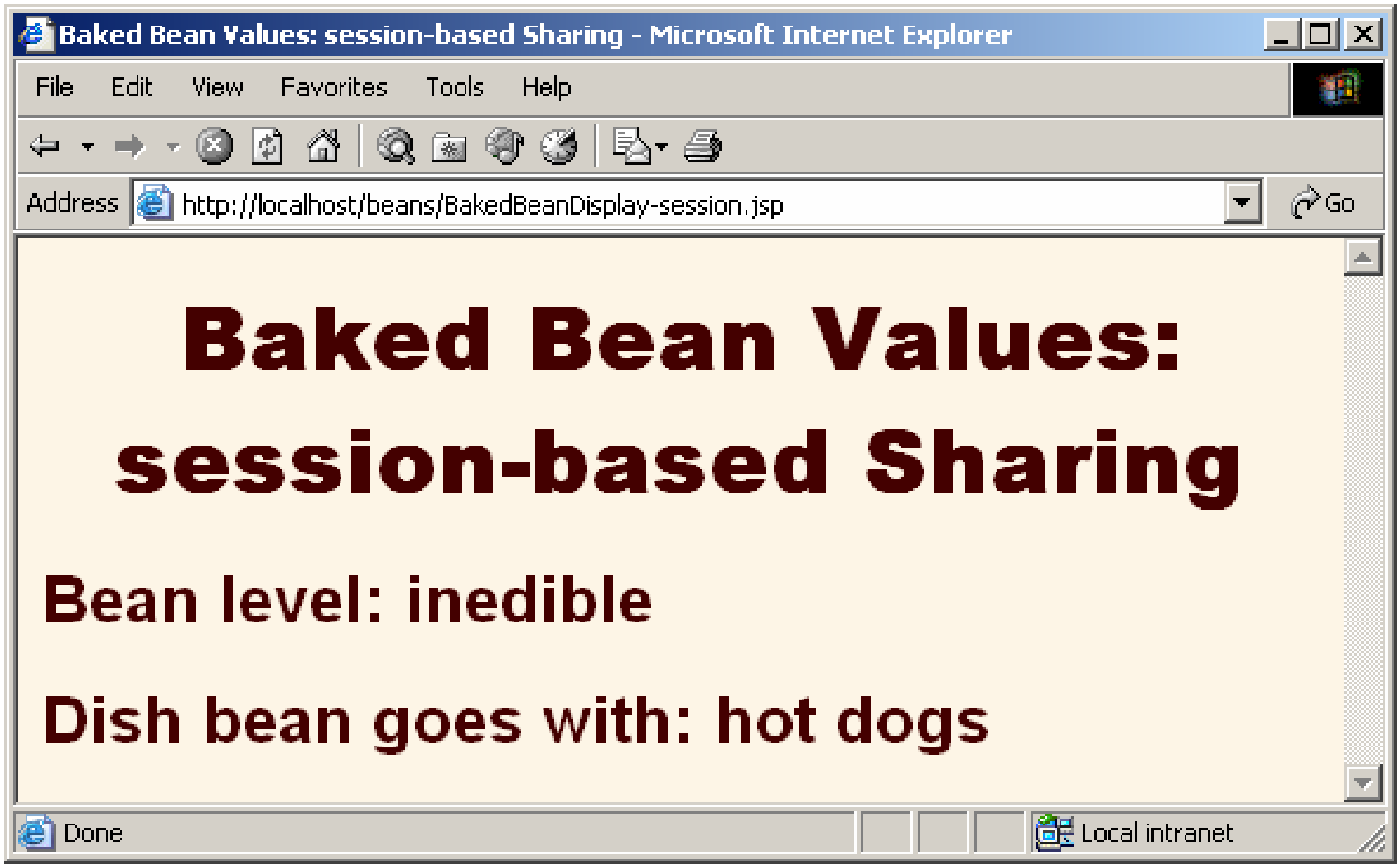
Session-Based Sharing: Code

```
…
<BODY>
<H1>Baked Bean Values: session-based Sharing</H1>
<jsp:useBean id="sessionBean"
             class="coreservlets.BakedBean"
             scope="session" />
<jsp:setProperty name="sessionBean"
                 property="*" />
<H2>Bean level:
<jsp:getProperty name="sessionBean"
                 property="level" />
</H2>
<H2>Dish bean goes with:
<jsp:getProperty name="sessionBean"
                 property="goesWith" />
</H2></BODY></HTML>
```

Session-Based Sharing: Result (Initial Request)



Session-Based Sharing: Result (Later Request -- Same Client)



Session-Based Sharing: Result (Later Request -- New Client)

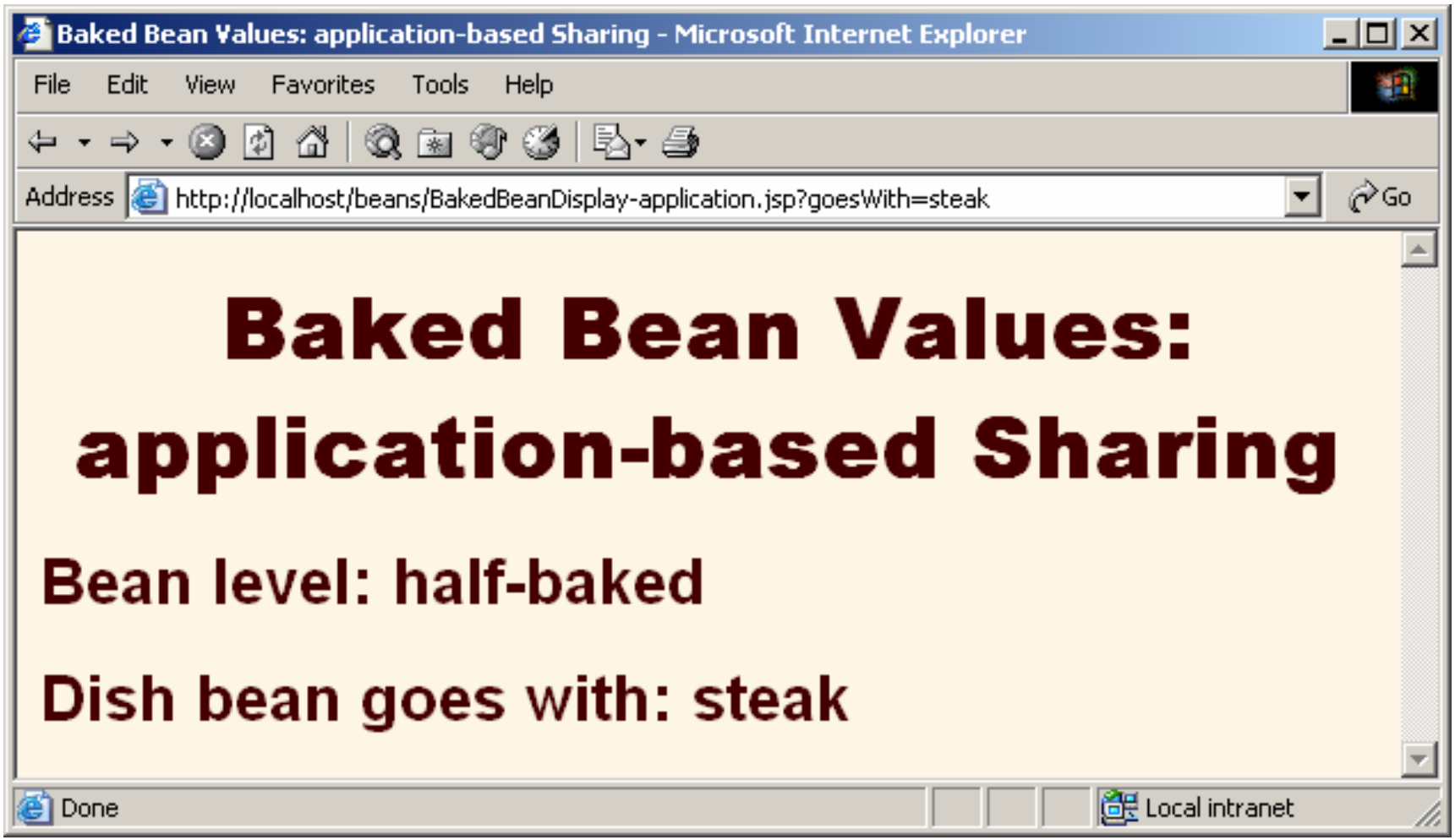
Sharing Beans Example 4: Application-Based Sharing

- **Create the bean**
  - Use jsp:useBean with scope="application".
- **Modify the bean**
  - Use jsp:setProperty with property="*".
  - Then, supply request parameters that match the bean property names.
- **Access the bean in the initial request**
  - Use jsp:getProperty in the request in which jsp:setProperty is invoked.
- **Access the bean later**
  - Use jsp:getProperty in a request that does not include request parameters and thus does not invoke jsp:setProperty. Whether this request is from the *same client or a different client* (regardless of the session timeout), the previously modified value is seen.

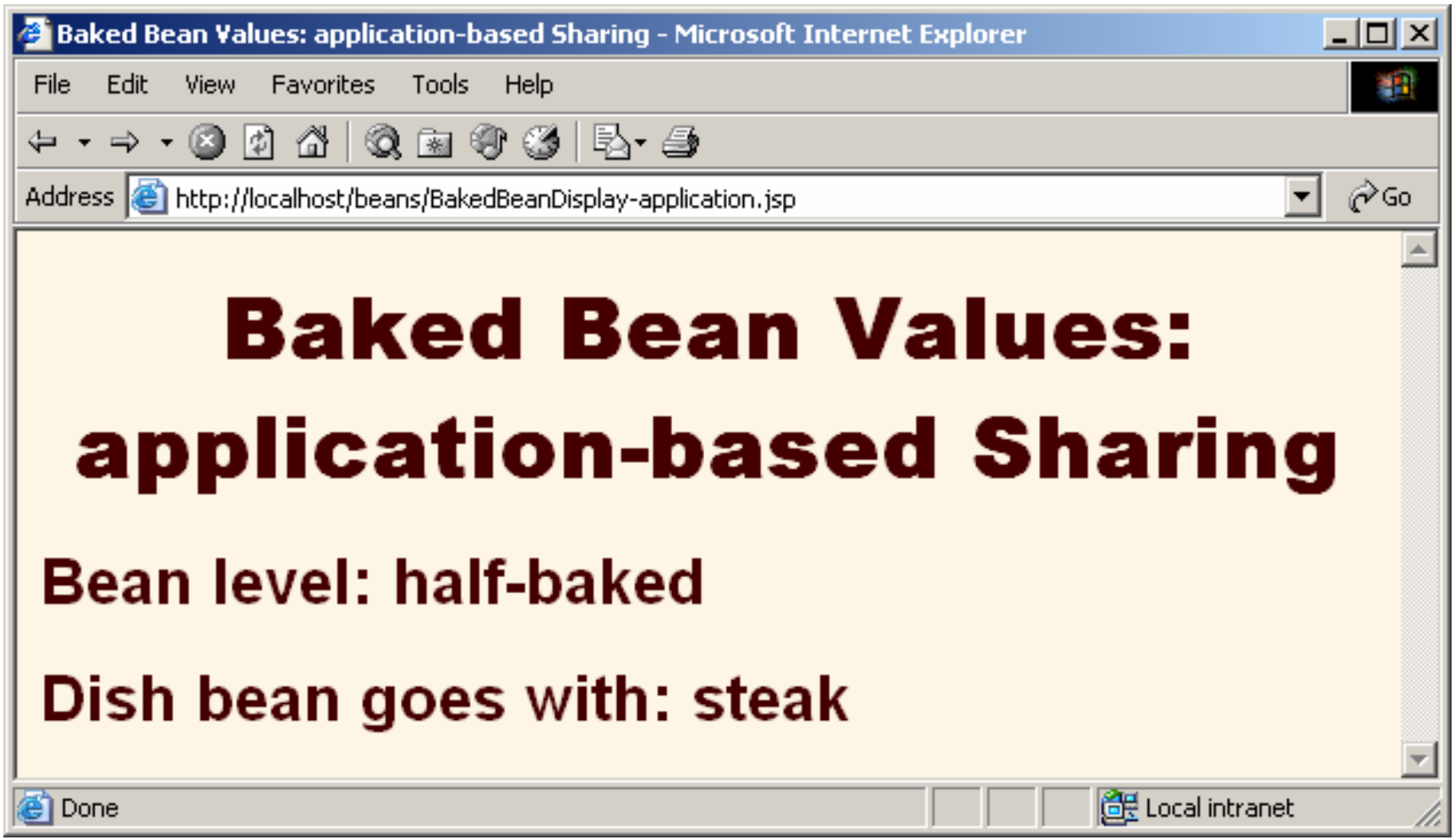
Application-Based Sharing: Code

```
<BODY>
<H1>Baked Bean Values:
application-based Sharing</H1>
<jsp:useBean id="applicationBean"
             class="coreservlets.BakedBean"
             scope="application" />
<jsp:setProperty name="applicationBean"
                 property="*" />
<H2>Bean level:
<jsp:getProperty name="applicationBean"
                 property="level" />
</H2>
<H2>Dish bean goes with:
<jsp:getProperty name="applicationBean"
                 property="goesWith"/>
</H2></BODY></HTML>
```

Application-Based Sharing: Result (Initial Request)



Application-Based Sharing: Result (Later Request -- Same Client)



Application-Based Sharing: Result (Later Request -- New Client)