



# JSF 2.0: Integrated Ajax Support

Originals of Slides and Source Code for Examples:  
<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live training on JSF 2.x, please see  
courses at <http://courses.coreservlets.com/>.**



**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.**

- Courses developed and taught by Marty Hall
    - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
    - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
  - Courses developed and taught by [coreservlets.com](http://coreservlets.com) experts (edited by Marty)
    - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details**

# Topics in This Section

- **Motivation**
  - Web apps in general
  - Ajax in general
  - Ajax integrated with JSF 2.0
- **Using f:ajax**
  - Overview
  - render: specifying elements to update on client
  - execute: specifying elements to process on server
  - event: specifying user events to respond to
  - onevent: specifying JavaScript side effects
  - Limitations on the use of h:outputText with Ajax

5

© 2012 Marty Hall



## Motivation

**Customized Java EE Training:** <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Why Web Apps?

- **Downsides to browser-based apps**

- GUI is poor
  - HTML is OK for static documents, but lousy for programs
- Communication is inefficient
  - HTTP is poor protocol for the way we now use Web apps



# Why Web Apps? (Continued)

- **So why does everyone want Web apps?**

- Universal access
  - Everyone already has a browser installed
  - Any computer on the network can access content
- Automatic “updates”
  - Content comes from server, so is never out of date





# Why Ajax?

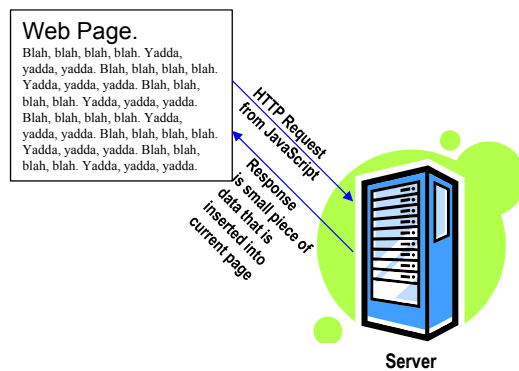
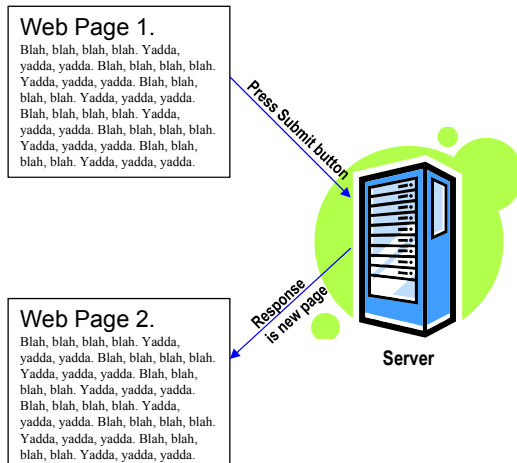
- 
- Superman, this is the iPad.  
It's the future of comics.  
It's going to save you.
- Spider-Man, this is the iPad.  
It's the future of comics.  
It's going to save you.
- Hulk, this is the iPad.  
It's the future of comics.  
It's going to save you.
- What's about me?
- Sorry, Flash, you're out of luck.
- So I'm guessing you don't hope to see your comics on the iPad.
- I'm sure Apple has a sense of humor about this stuff... Right? Right? Right?
- © 2011 by Dave Coverly. All rights reserved. www.dilbert.com

From Bill Amend and foxtrot.com. © Universal Press Syndicate

9

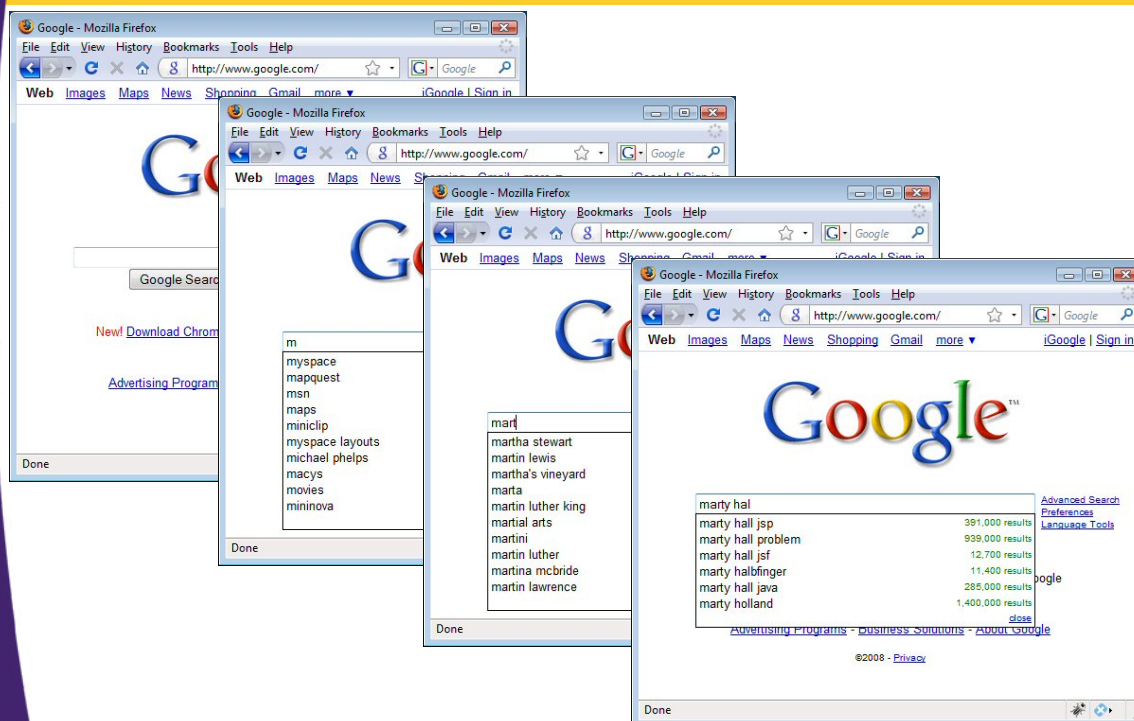
# Traditional Web Apps vs. Ajax Apps

- **Traditional Web Apps:**  
Infrequent Large Updates
- **Ajax Apps:**  
Frequent Small Updates



10

# Google Home Page (formerly Google Suggest)



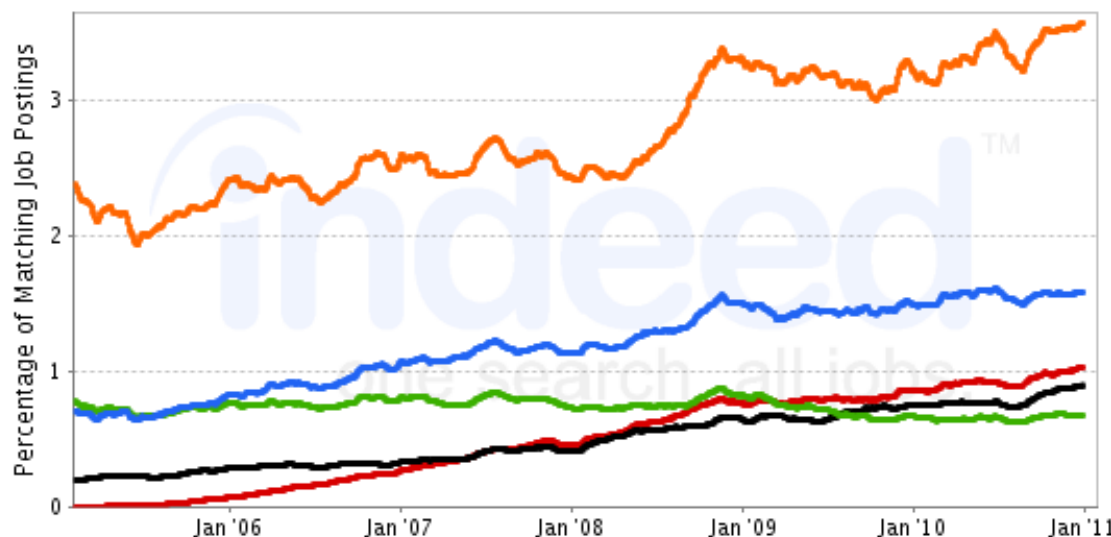
11

# Ajax Jobs

Indeed.com compiles data from multiple jobs sites

**Job Trends from Indeed.com**

— java — c# — vb — php — ajax



12

# Why Ajax in JSF?

- **Why a JSF-specific Ajax library?**
  - There are tons of Ajax libraries already (jQuery, DWR, GWT, etc.). Why invent a new one for JSF?
- **Advantages of a JSF-specific Ajax approach**
  - Client side
    - You can update *JSF* elements (h:outputText, h:inputText, h:selectOneMenu, etc.)
      - It would be very unwieldy if Ajax updates were entirely separate elements from the Ajax UI elements
    - You don't have to write JavaScript
  - Server side
    - Ajax calls know about JSF managed beans
      - Including reading form fields and setting bean properties
    - You don't have to write servlets and parse parameters

13

© 2012 Marty Hall



## f:ajax – Overview

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Simplest Form

- **Code**

- `<h:commandButton ... action="...">`  
    `<f:ajax render="id1"/>`  
    `</h:commandButton>`  
    `<h:outputText ... value="#{...}" id="id1"/>`

- **Interpretation**

- When the pushbutton is pressed, go to server, run the action, compute the value of the JSF element whose id is “id1”, send that value back to the client, then replace that element in the DOM with the new value.
  - If the “value” attribute computes a new result each time, then the “action” of the button can be a dummy value  
    `<h:commandButton value="Update Time" action="nothing">`  
        `<f:ajax render="timeResult"/>`  
    `</h:commandButton>`  
    `<h:outputText value="#{dateBean.time}" id="timeResult"/>`

15

# General Form

- **Code**

- `<h:commandButton ... action="...">`  
    `<f:ajax render="id1 id2" execute="id3 id4"`  
        `event="blah" onevent="javaScriptHandler"/>`  
    `</h:commandButton>`

- **Attributes**

- render
  - The elements to redisplay on the page. Often `h:outputText`
    - The target of the render must be inside the same `h:form`
- execute
  - The elements to send to server to process. Generally input elements such as `h:inputText` or `h:selectOneMenu`.
- event
  - The DOM event to respond to (e.g., `keyup`, `blur`)
- onevent
  - A JavaScript function to run when event is fired

16

# Structure of Facelets Pages

- **Declare the f:namespace**

- In the `<html ...>` start tag, you must declare the namespace for the f: tags
  - We saw the same requirement in other tutorial sections where we used f: tags (e.g., f:selectItems and f:param)

- **Use h:head**

- When you use f:ajax, the system automatically inserts a `<script>` tag into the `<head>` section of the page. It cannot find the head section unless you use h:head
- As discussed in early tutorial sections, it is a good standard practice to *always* use h:head and h:body.
- If browser has JavaScript disabled, Ajax buttons will automatically become normal buttons with form submissions and then with the same page redisplayed

17

# Structure of Facelets Pages

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    ...
  </h:head>
  <h:body>
    ...
  </h:body>
</html>
```

18



# Signature for Action Controller Methods

- **Non-Ajax**

```
public String myActionControllerMethod() {  
    ...  
    return("some outcome");  
}
```

- **Ajax: permitted**

```
public void myActionControllerMethod() { ... }
```

- **Ajax: preferred**

```
public String myActionControllerMethod() {  
    ...  
    return(null); // In non-Ajax apps, means to redisplay form  
}
```

- Method will work for *both* Ajax and non-Ajax forms

19

# Ajax Testing

- **JavaScript is notoriously inconsistent**

- You hope that the JSF implementation took this into account, and hid the browser differences. Nevertheless, JSF 2.0 is a specification, not an implementation, so different implementations could be better or worse at this.

- **Test on multiple browsers**

- If you field an internal application, test on all officially sanctioned browsers on all supported operating systems.
- If you field an external application, test on as many browsers as possible. Preferably: IE 6-9, a recent Firefox implementation, Chrome, and Safari.
  - Test regularly on IE and Firefox. Test on a wider set of browsers before deploying.
    - Usage: [http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)
  - This is needed for any Ajax application, not just for JSF 2.

20



## render: Specifying Elements to Update on Client

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## The render attribute of f:ajax

- **Code summary**
  - `<f:ajax render="elementId" ... />`
- **Idea**
  - Id or space-separated list of ids of JSF elements whose values should be returned from server and replaced in DOM
    - These JSF elements should be in same h:form as f:ajax
- **Details**
  - There are four special values: `@this`, `@form`, `@none`, and `@all`. However, these are more often used for the execute attribute than the render attribute. See execute section.
  - Values for render (and execute and event) can be JSF expressions instead of literal strings

# Facelets Code: Non-Ajax Version

```
<h:form>
<fieldset>
  <legend>Random Number: No Ajax</legend>
  <h:commandButton value="Show Number"
                    action="#{numberGenerator.randomize}"/>
  <h2><h:outputText value="#{numberGenerator.number}"
                    id="numField1"/></h2>
</fieldset>
</h:form>
```

When the pushbutton is pressed, submit form and on server, get bean whose name is numberGenerator (since it is session scoped, use existing instance if available, otherwise instantiate it). Then, run the randomize method. Then, compute the value of the getNumber method and insert it into the page where the h:outputText is. Note that the "id" attribute is not needed here; it is just planning ahead for the next slide. In this specific example, we could have just replaced the entire h:outputText with #{numberGenerator.number}.

23

# Facelets Code: Ajax Version

```
<h:form>
<fieldset>
  <legend>Random Number</legend>
  <h:commandButton value="Show Number"
                    action="#{numberGenerator.randomize}">
    <f:ajax render="numField1"/>
  </h:commandButton><br/>
  <h2><h:outputText value="#{numberGenerator.number}"
                    id="numField1"/></h2>
</fieldset>
</h:form>
```

When the pushbutton is pressed, have JavaScript make a request to the server without submitting the form. On the server, get bean whose name is numberGenerator (since it is session scoped, use existing instance if available, otherwise instantiate it). Then, run the randomize method. Then, compute the value of the getNumber method. Send that value back to the client and insert it into the DOM in the place where the h:outputText is. Note that the thing being updated (i.e., the target of render) must be inside the same h:form as the f:ajax tag that refers to it.

If JavaScript is disabled in the browser, then this form will work exactly like the one on the previous slide (i.e., with a normal form submission and then page redisplay).

24

# Bean Code

```
@ManagedBean
public class NumberGenerator {
    private double number = Math.random();
    private double range = 1.0;

    public double getRange() { return(range); }

    public void setRange(double range) {
        this.range = range;
    }
    public double getNumber() { return(number); }

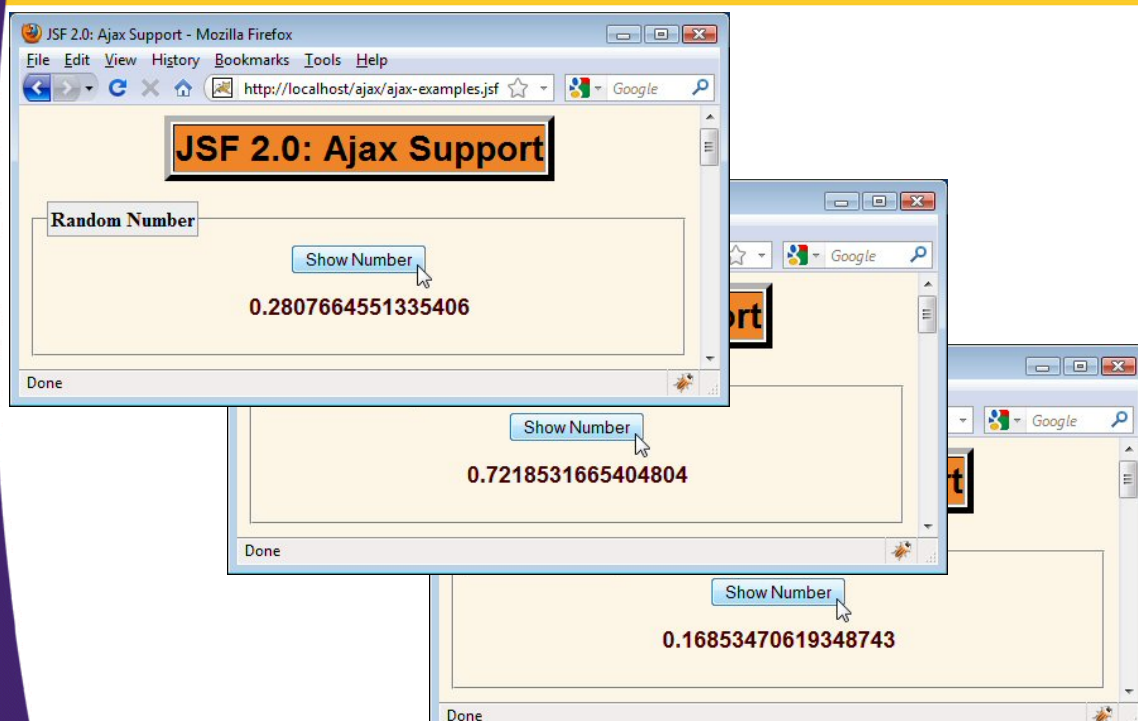
    public String randomize() {
        number = range * Math.random();
        return(null);
    }
}
```

getRange and setRange aren't used in this example, but are used in an upcoming one.

If randomize were used only in the Ajax example, it would have been legal to declare it  
`public void randomize() { ... }`  
But, better to return null so that it can be used in non-Ajax examples as well.

25

# Results



26





## execute: Specifying Elements to Process on Server

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## The execute attribute of f:ajax

- **Code summary**
  - `<f:ajax render="..." execute="..." ... />`
- **Idea**
  - An id or space-separated list of ids of JSF elements that should be sent to the server for execution.
    - `h:inputText` processed normally (setters, validation, etc.)
- **Details**
  - There are 4 special values: `@this`, `@form`, `@none`, `@all`
    - `@this`. The element enclosing `f:ajax`. Default.
    - `@form`. The `h:form` enclosing `f:ajax`. Very convenient if you have multiple fields to send. Shown in later example.
    - `@none`. Nothing sent. Useful if the element you render changes values each time you evaluate it.
    - `@all`. All JSF UI elements on page.

# Facelets Code

```
<h:form>
<fieldset>
  <legend>Random Number (with execute="someId")</legend>
  Range:
  <h:inputText value="#{numberGenerator.range}"
               id="rangeField"/><br/>
  <h:commandButton value="Show Number"
                   action="#{numberGenerator.randomize}">
    <f:ajax execute="rangeField"
            render="numField3"/>
  </h:commandButton><br/>
  <h2><h:outputText value="#{numberGenerator.number}"
                    id="numField3"/></h2>
</fieldset>
</h:form>
```

29

# Bean Code

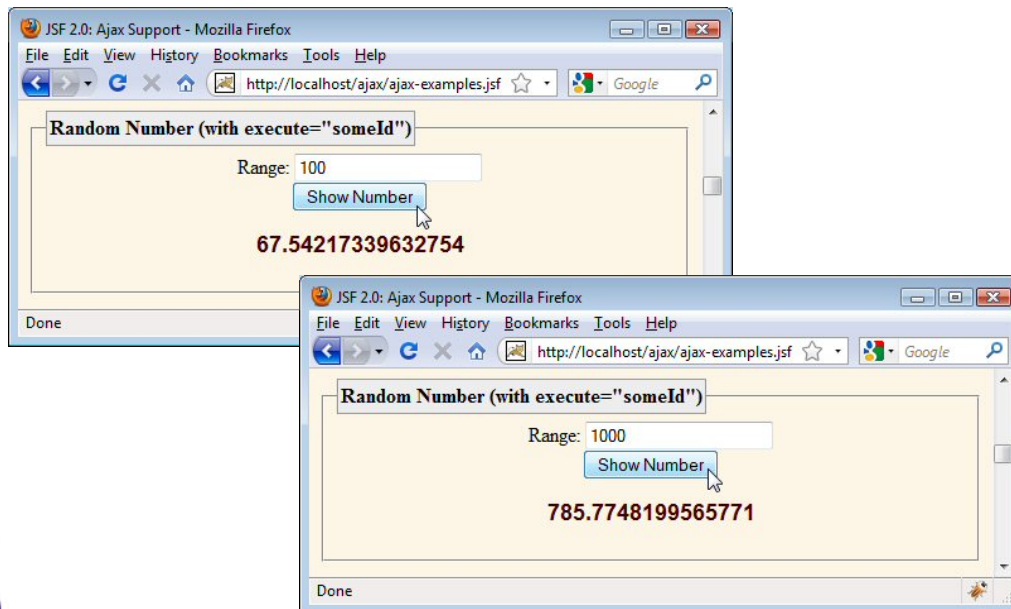
```
@ManagedBean
public class NumberGenerator {
  private double number = Math.random();
  private double range = 1.0;

  public double getRange() {
    return(range);
  }
  public void setRange(double range) {
    this.range = range;
  }
  public double getNumber() {
    return(number);
  }
  public String randomize() {
    number = range * Math.random();
    return(null);
  }
}
```

This is the same bean code as in previous examples. But shown again to emphasize that setRange (corresponding to the h:inputText element specified with execute) will be called before randomize (the action of the h:commandButton that surrounds f:ajax).

30

# Results



31

## Using execute="@form"

- **Code summary**

- `<h:form />`

- ...
    - `<f:ajax render="elementId" execute="@form" />`
    - ...

- `</h:form>`

- **Idea**

- Send all elements of current form to server to process.
    - Again, processes form elements in normal JSF manner (performs validation, calls setter methods, etc.)

- **Details**

- Convenient if you have several input fields and don't want to list each one in "render". Also, input fields don't need explicit ids when you use `@form`.

32

# Facelets Code

```
<h:form>
<fieldset>
  <legend>Bank Customer Lookup (with execute="@form")</legend>
  Customer ID:
  <h:inputText value="#{bankingBeanAjax.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBeanAjax.password}"/><br/>
  <h:commandButton value="Show Current Balance"
    action="#{bankingBeanAjax.showBalance}"
    <f:ajax execute="@form"
      render="ajaxMessage1"/>
  </h:commandButton>
  <br/>
  <h2><h:outputText value="#{bankingBeanAjax.message}"
    id="ajaxMessage1"/></h2>
</fieldset>
</h:form>
```

I didn't need to give explicit ids to the input fields. JSF generates ids automatically when no id specified.  
Since @form was given, JSF finds all elements in current form and sends them to server for processing.

33

# Bean Code

```
@ManagedBean
public class BankingBeanAjax extends BankingBeanBase {
  private String message = "";

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

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

34



## Bean Code (Continued)

```
public String showBalance() {
    if (!password.equals("secret")) {
        message = "Incorrect password";
    } else {
        CustomerLookupService service =
            new CustomerSimpleMap();
        customer = service.findCustomer(customerId);
        if (customer == null) {
            message = "Unknown customer";
        } else {
            message =
                String.format("Balance for %s %s is $%,.2f",
                    customer.getFirstName(),
                    customer.getLastName(),
                    customer.getBalance());
        }
    }
    return (null);
}
```

The message starts off as an empty String. Once the button is pressed, showBalance changes the message to either an error message or a string showing the balance. Then JSF replaces the h:outputText value in the DOM with the new message. This is same supporting class as shown in previous section on annotations. CustomerSimpleMap is just a lookup table of a few customers, and is shown in the previous section (plus is in the downloadable source code).

35

## BankingBeanBase.java

```
...
public abstract class BankingBeanBase {
    protected String customerId, password;
    protected Customer customer;

    public String getCustomerId() { return(customerId); }

    public void setCustomerId(String customerId) {
        this.customerId = customerId;
    }

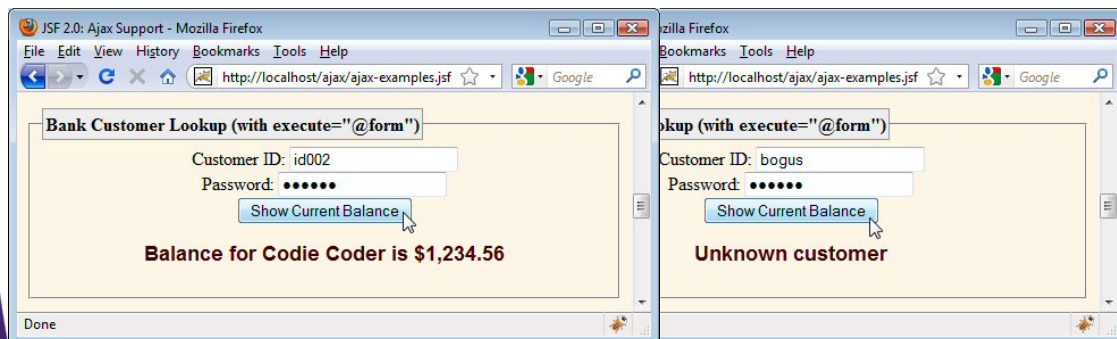
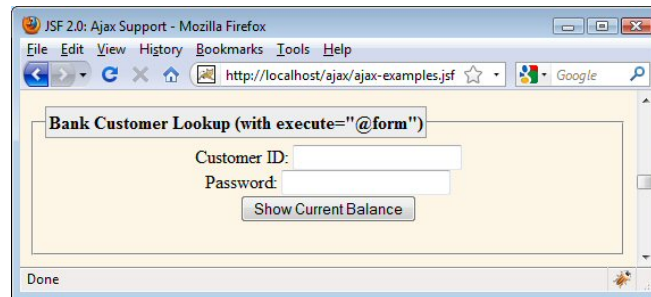
    public String getPassword() { return(password); }

    public void setPassword(String password) {
        this.password = password;
    } ... }
}
```

These will be automatically called by JSF because the corresponding h:input elements were processed due to the execute="@form" attribute.

36

# Results



© 2012 Marty Hall



## event: Specifying User Events to Respond To

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# The event attribute of f:ajax

- **Code summary**

- `<f:ajax render="..." event="..." ... />`

- **Idea**

- The name of the DOM event to respond to. You don't include "on", so it is mouseover, keyup, blur, etc.

- **Details**

- Defaults
  - If unspecified, default event used. See next slide.
- High-level events
  - JSF adds 2 extras: action & valueChange. See next slide.
- Wrapping f:ajax around elements
  - `<f:ajax render="...">a bunch of components</f:ajax>`
    - Adds Ajax behavior on default event for each wrapped component

39

# Default Events

- **action**

- h:commandButton, h:commandLink
  - Note that "action" is part of JSF, and not a native JavaScript/DOM event name. It means button has been invoked in any way (clicking on it, ENTER if it has focus, keyboard shortcut, etc.)

- **valueChange**

- h:inputText, h:inputSecret, h:inputTextarea, all radio button, checkbox, & menu items (h:selectOneMenu, etc.)
  - Again, this event is added by JSF and is not a native DOM event name. Different browsers handle "change" differently, so this unifies the behavior.
  - Also note that it is "valueChange", not "valuechange". The native DOM events are all lower case (mouseover, keyup, etc.)

40

# Example 1: Chained Combo Boxes (Select Menus)

- **Idea**

- Use h:selectOneMenu to make a list of US states
- When the user selects a state, a list of corresponding cities is shown (again, using h:selectOneMenu)
- When city selected, population of that city is displayed

- **Approach**

- State list
  - <f:ajax render="cityList"/> (valueChange is default event)
- City List
  - <f:ajax render="population"/>
- Bean
  - Make managed bean session scoped so that city entry gets sent to same bean that already stored state

41

## Facelets Code

```
<h:form>
<fieldset>
  <legend>Population Lookup (with chained comboboxes)</legend>
  State:
  <h:selectOneMenu value="#{locationBean.state}">
    <f:selectItems value="#{locationBean.states}"/>
    <f:ajax render="cityList"/>
  </h:selectOneMenu>
  <br/>City:
  <h:selectOneMenu value="#{locationBean.city}"
    disabled="#{locationBean.cityListDisabled}"
    id="cityList">
    <f:selectItems value="#{locationBean.cities}"/>
    <f:ajax render="population"/>
  </h:selectOneMenu>
  <br/>Population:
  <h:outputText value="#{locationBean.city}"
    id="population"/>
</fieldset></h:form>
```

42



# Bean Code

```
@ManagedBean
@SessionScoped
public class LocationBean implements Serializable {
    private String state, city;
    private boolean isCityListDisabled = true;

    public String getState() {
        return (state);
    }

    public void setState(String state) {
        this.state = state;
        isCityListDisabled = false;
    }
}
```

Make city list disabled  
(grayed out) initially. Enable  
it when the state is selected.

43

# Bean Code (Continued)

```
public String getCity() {
    return(city);
}

public void setCity(String city) {
    this.city = city;
}

public boolean isCityListDisabled() {
    return(isCityListDisabled);
}
```

Although this is called setCity, the String passed to setCity and returned from getCity really represents the population. That is because, for combo boxes and list boxes, JSF lets you have one value displayed to the end user and a different value passed to the setter method. See the SelectItem array in nearbyStates in upcoming slide.

44

## Bean Code (Continued)

```
public List<SelectedItem> getStates() {  
    List<SelectedItem> states = new ArrayList<SelectedItem>();  
    states.add(new SelectedItem("--- Select State ---"));  
    for(StateInfo stateData: StateInfo.getNearbyStates()) {  
        states.add(new SelectedItem(stateData.getStateName()));  
    }  
    return(states);  
}
```

Put dummy value at the top of the list  
so that any real user selection is  
considered a change

This part uses the one-argument version of the SelectedItem constructor. In JSF, that means that the value displayed to the end user and the value passed to the bean setter method are the same. The upcoming city example will use two different values per entry.

45

## Bean Code (Continued)

```
public SelectedItem[] getCities() {  
    SelectedItem[] cities =  
        { new SelectedItem("--- Choose City ---")};  
    if(!isCityListDisabled && (state != null)) {  
        for(StateInfo stateData: StateInfo.getNearbyStates()) {  
            if(state.equals(stateData.getStateName())) {  
                cities = stateData.getCities();  
                break;  
            }  
        }  
    }  
    return(cities);  
}
```

Result of first Ajax request: i.e., value  
from the first combo box (of states)

Value is a SelectedItem[] that will go in  
the second combo box (of cities).

46

## Supporting Class (StateInfo)

```
public class StateInfo {
    private String stateName;
    private SelectItem[] cities;

    public StateInfo(String stateName, SelectItem...cities) {
        this.stateName = stateName;
        this.cities = cities;
    }

    public String getStateName() {
        return(stateName);
    }

    public SelectItem[] getCities() {
        return(cities);
    }
}
```

Value is a SelectItem array

47

## Supporting Class (StateInfo) Continued

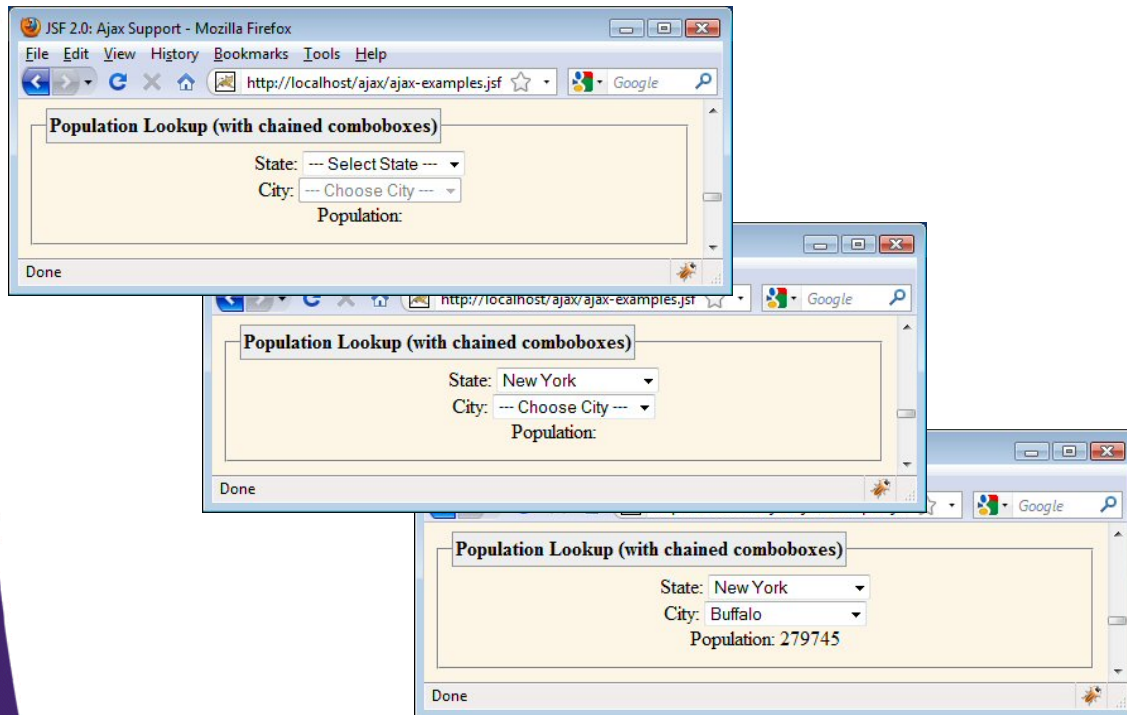
```
private static StateInfo[] nearbyStates =
    { new StateInfo("Maryland",
        new SelectItem("<i>unknown</i>",
            "--- Choose City ---"),
        new SelectItem("635815", "Baltimore"),
        new SelectItem("57907", "Frederick"),
        new SelectItem("57698", "Gaithersburg"),
        new SelectItem("57402", "Rockville")),
        ... // other states
    };

public static StateInfo[] getNearbyStates() {
    return(nearbyStates);
}
}
```

JSF lets you have dual-value entries in combo boxes and list boxes. The values on the right (city names) are displayed to the end user. But if you programmatically ask for the selected value, you get the value on the left (corresponding populations). That is, the value on the left is passed to the bean setter method. So, when the Ajax request fires for the valueChange event for the second combo box, the population gets passed to setCity.

48

# Results



49

## Example 2: On-the-Fly Temperature Converter

- **Idea**
  - The user types a temperature in Fahrenheit into textfield
  - As the value is being entered, the corresponding values in Celsius and Kelvin are displayed
- **Approach**
  - Temperature field
    - `<f:ajax event="keyup" render="cField kField"/>`
    - `keyup` is not the default event, so needs to be specified explicitly in “event”
    - We want to update two output fields, so list both for “render”
  - Bean
    - Temp (in F) passed each time, so use request scope

50



# Facelets Code

```
<h:form>
<fieldset>
  <legend>On-the-Fly Temperature Converter ...</legend>
  Temperature in Fahrenheit:
  <h:inputText value="#{temperatureConverter.fTemp}">
    <f:ajax event="keyup"
      render="cField kField"/>
  </h:inputText><br/>
  <h2>
    Temperature in Celsius:
    <h:outputText value="#{temperatureConverter.cTemp}"
      id="cField"/><br/>
    Temperature in Kelvin:
    <h:outputText value="#{temperatureConverter.kTemp}"
      id="kField"/><br/>
  </h2>
</fieldset></h:form>
```

51

# Bean Code

```
@ManagedBean
public class TemperatureConverter {
  private String cTemp, kTemp;

  public String getcTemp() {
    return(cTemp);
  }

  public String getkTemp() {
    return(kTemp);
  }

  public String getfTemp() {
    return("");
  }
}
```

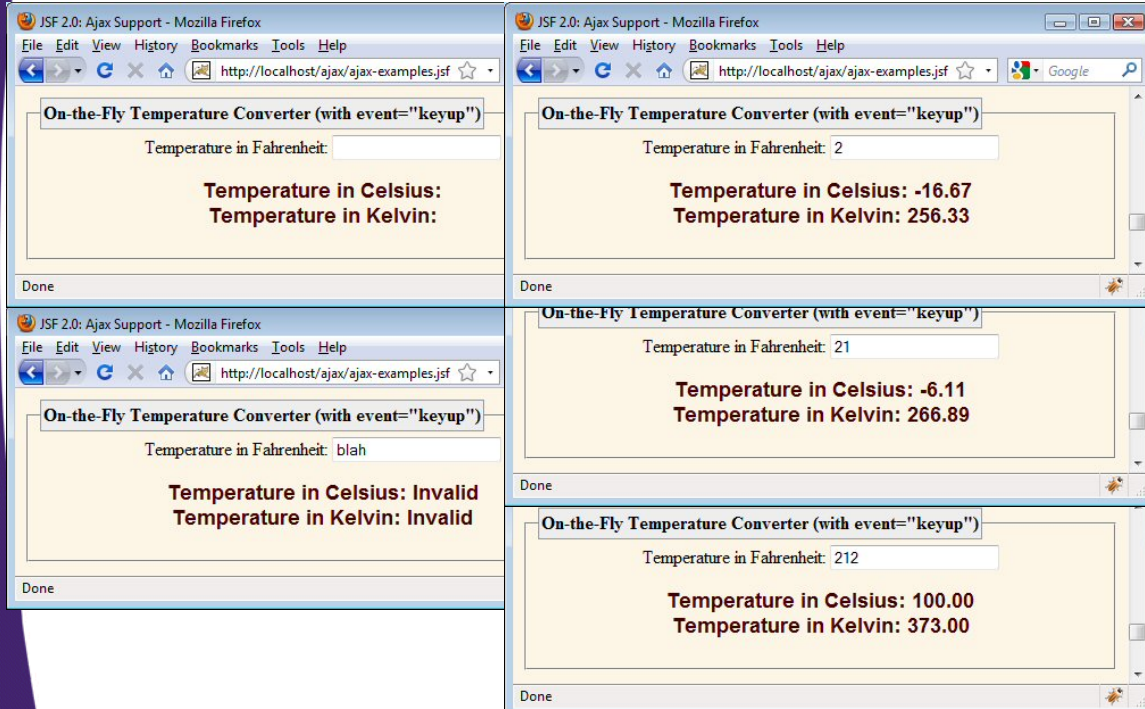
52

# Bean Code (Continued)

```
public void setfTemp(String fTemp) {  
    double f = -500;  
    try {  
        f = Double.parseDouble(fTemp);  
    } catch (NumberFormatException nfe) {  
        cTemp = "Invalid";  
        kTemp = "Invalid";  
    }  
    if (f >= -459.4) {  
        double c = (f - 32)*(5.0/9.0);  
        double k = c + 273;  
        cTemp = String.format("%.2f", c);  
        kTemp = String.format("%.2f", k);  
    }  
}
```

53

# Results



54



# onevent:

## Specifying JavaScript Side Effects to Run Before/After Ajax Request

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## The onevent attribute of f:ajax

- **Code summary**
  - `<f:ajax render="..." onevent="functionName" ... />`
- **Idea**
  - The name of a JavaScript function to call at various stages of the Ajax submission and response. Function should take one argument (e.g., function blah(data) {...})
- **Details on argument to JavaScript function**
  - The status property (e.g., data.status in example above)
    - Either "begin", "complete", or "success" (in that order)
  - The source property (e.g., data.source)
    - The DOM event that triggered the Ajax request
  - responseCode, responseText, responseXML
    - The values in XHR object. Omitted for "begin" event.

# Example: Showing “Getting Data...” Message While Waiting

- **Idea**

- You have slow server operation
- Display animated GIF (& message) when request sent
- Hide GIF/message when response completes

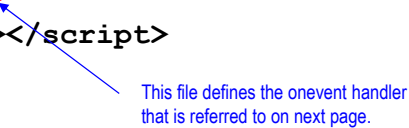
- **Approach**

- Get animated GIF
  - <http://ajaxload.info/> lets you build your own
- Display image plus message in region with display: none
  - So it is hidden initially
- When request begins, change to display: inline
  - Use onevent handler and “begin” status
- When request finishes successfully, make display: none
  - Use onevent handler and “success” status

57

## Facelets Code

```
...
<h:head><title>JSF 2.0: Ajax Support</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
<script src="./scripts/utils.js"
        type="text/javascript"></script>
</h:head>
...
```



This file defines the onevent handler that is referred to on next page.

In this simple example, we load style sheets and JavaScript files the normal XHTML way. However, if you have pages in several folders that use the same resources (especially with `ui:composition`), this standard approach is inconvenient since the relative URL to the resources could be different for each of the pages. So, JSF 2.0 adds `h:outputScript`, `h:outputStyleSheet`, and an option for `h:graphicImage`. These let you load resources (scripts, style sheets, and images) with the same syntax, regardless of where the loading page is situated in your app. These approaches will be covered in the later section on page templating.

58

## Facelets Code (Continued)

```
<h:form>
<fieldset>
  <legend>Bank Customer Lookup (with onevent)</legend>
  Customer ID:
  <h:inputText value="#{bankingBeanSlow.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBeanSlow.password}"/><br/>
  <h:commandButton value="Show Current Balance"
    action="#{bankingBeanSlow.showBalance}"
    <f:ajax execute="@form"
      render="ajaxMessage2"
      onevent="showWorkingIndicator"/>
  </h:commandButton>
```

Calls the showWorkingIndicator JavaScript function (loaded on previous slide) with a data object at various stages in the Ajax process. The data object has a status property that lets the programmer decide whether it is before or after the Ajax request.

59

## Facelets Code (Continued)

```
<h2>
  <span id="workingIndicator"
    style="display: none; font-size: 60%">
    
    Loading data from server...
  </span>
  <h:outputText value="#{bankingBeanSlow.message}"
    id="ajaxMessage2"/></h2>
</fieldset>
</h:form>
```

This span is hidden initially. It is made visible in the first call to the showWorkingIndicator onevent handler (for the "begin" status) and hidden again in a later call to the onevent handler (for the "success" status).

60



# JavaScript Code

```
function showWorkingIndicator(data) {  
    showIndicatorRegion(data, "workingIndicator");  
}
```

```
function showIndicatorRegion(data, regionId) {  
    if (data.status == "begin") {  
        showElement(regionId);  
    } else if (data.status == "success") {  
        hideElement(regionId);  
    }  
}
```

```
function showElement(id) {  
    document.getElementById(id).style.display  
        = "inline";  
}
```

```
function hideElement(id) {  
    document.getElementById(id).style.display  
        = "none";  
}
```

This is the specific function referred to in onevent. It is not reusable since it refers to the specific id of the region. Experienced JavaScript developers could also make an anonymous function on the main page that captures the id, and use that for the onevent handler.

This is the reusable function called above.

These make the hidden region visible and invisible. Experienced JavaScript developers might use a library like jQuery or Prototype and then use shortcut methods like `$(id).show()` and `$(id).hide()`.

61

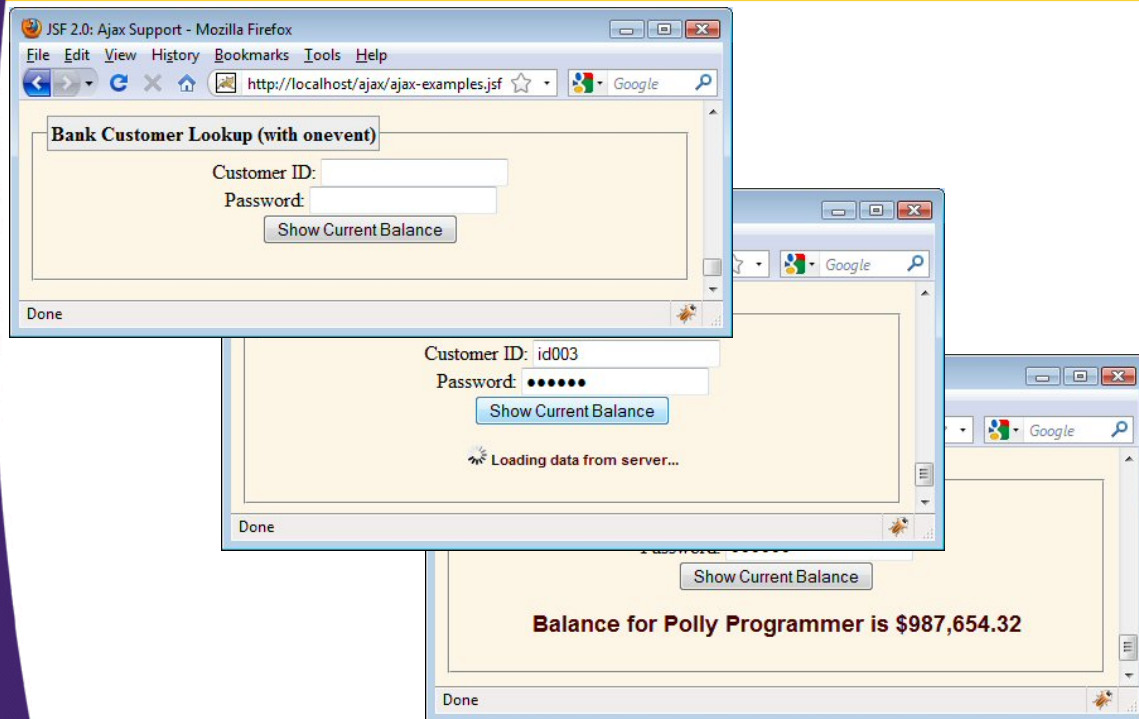
# Bean Code

```
@ManagedBean  
public class BankingBeanSlow extends BankingBeanAjax {  
    public String showBalance() {  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException ie) {}  
        return (super.showBalance());  
    }  
}
```

Works the same as the previous banking example, except for the extra five-second delay.

62

# Results



63

© 2012 Marty Hall



## Limitations on Use of h:outputText with Ajax

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Main Point

- **General point**

- Not all uses of `<h:outputText value="#{bean.prop}"/>` or `#{bean.prop}` can be updated with Ajax
  - Once you add an id, output becomes a span, which limits the places it can be used

- **Specifics**

- You can use `<f:ajax render="..." />` to dynamically build content that could go inside an existing HTML element
  - E.g., between `<div>` and `</div>`
- You cannot directly use `<f:ajax render="..." />` to dynamically change attributes of HTML elements
  - Doing so would require some explicit JavaScript programming, as with the last example where we changed the “display” property

65

# Quick Example

- **Before Ajax (legal)**

- JSF code: `<h1>#{cust.balance}</h1>`
- Resultant HTML: `<h1>$123.45</h1>`

- **After Ajax (legal)**

- JSF code:  
`<h1><h:outputText value="#{cust.balance}" id="id1"/></h1>`
- Resultant HTML: `<h1><span id="id1">$123.45</span></h1>`

- **Before Ajax (legal)**

- JSF code: `<font color="#{userPrefs.color}"/>`
- Resultant HTML: `<font color="blue"/>`

- **After Ajax (illegal)**

- JSF code: `<font color=`  
`"<h:outputText value='#{userPrefs.color}' id='id2'/>">`
- Resultant HTML: `<font color="<span id="id2">blue</span>">`

66

# **`#{bean.prop}`**

- **Overview of usage**

- Can be used to generate
  - HTML content
  - Complete HTML tags
  - HTML attributes

- **Examples**

- `<h1>#{someBean.someProperty}</h1>`
- `#{bean.startTag} blah, blah #{bean.endTag}`
- `<body bgcolor="#{someBean.someProperty}">`

- **Preview**

- The last two examples cannot be done once you use `h:inputText` with an `id`, because this results in a `span` containing a string, not just a simple string.

67

# **`<h:outputText value="#{bean.prop}"/>`**

- **Overview of usage**

- Can be used to generate
  - HTML content
  - Complete HTML tags

- **Examples**

- `<h1><h:outputText value="#{bean.prop}"/></h1>`
- `<h:outputText value="#{bean.startTag}"/>blah, blah`  
`<h:outputText value="#{bean.endTag}"/>`

- **Illegal**

- `<body bgcolor='<h:outputText value="#{bean.prop}"/>'>`
  - Violates XML syntax rules

68

## **<h:outputText value="#{bean.prop}" id="foo"/>**

- **Overview of usage**

- Can be used to generate
  - HTML content (outputs `<span id="foo">content</span>`)

- **Example**

- `<h1>`  
`<h:outputText value="#{bean.prop}" id="foo"/></h1>`

- **Illegal**

- `<h:outputText value="#{bean.startTag}" id="foo"/>`  
`blah, blah`  
`<h:outputText value="#{bean.endTag}" id="foo"/>`
  - `<span...><h1></span>blah, blah<span...></h1></span>`
- `<body bgcolor='<h:outputText value="#{bean.prop}"/>'>`
  - Violates XML syntax rules

69

## **Example: #{bean.prop} Generating Attributes**

- **Idea**

- Every time page is loaded, use different colors and different text for a specified region

- **Approach**

- To generate colors
  - `<th bgcolor="#{textGenerator.randomColor}">`
- To generate text
  - `<h1>#{textGenerator.randomText}</h1>`

- **Point**

- Color generation cannot be ajaxified
- Text generation can easily be ajaxified

70



# Facelets Code

```
<table>
  <tr><th bgcolor="#{textGenerator.randomColor}">
    <h2>#{textGenerator.randomText}</h2>
  </th></tr>
</table>
```

71

# Bean Code

```
@ManagedBean
@ApplicationScoped
public class TextGenerator {
  private String[] colors =
    { "red", "yellow", "blue", "green" };
  private String[] phrases =
    { "Blah, blah, blah. ",
      "Yadda, yadda, yadda. ",
      "Foo, bar, baz. "
    };
};
```

72

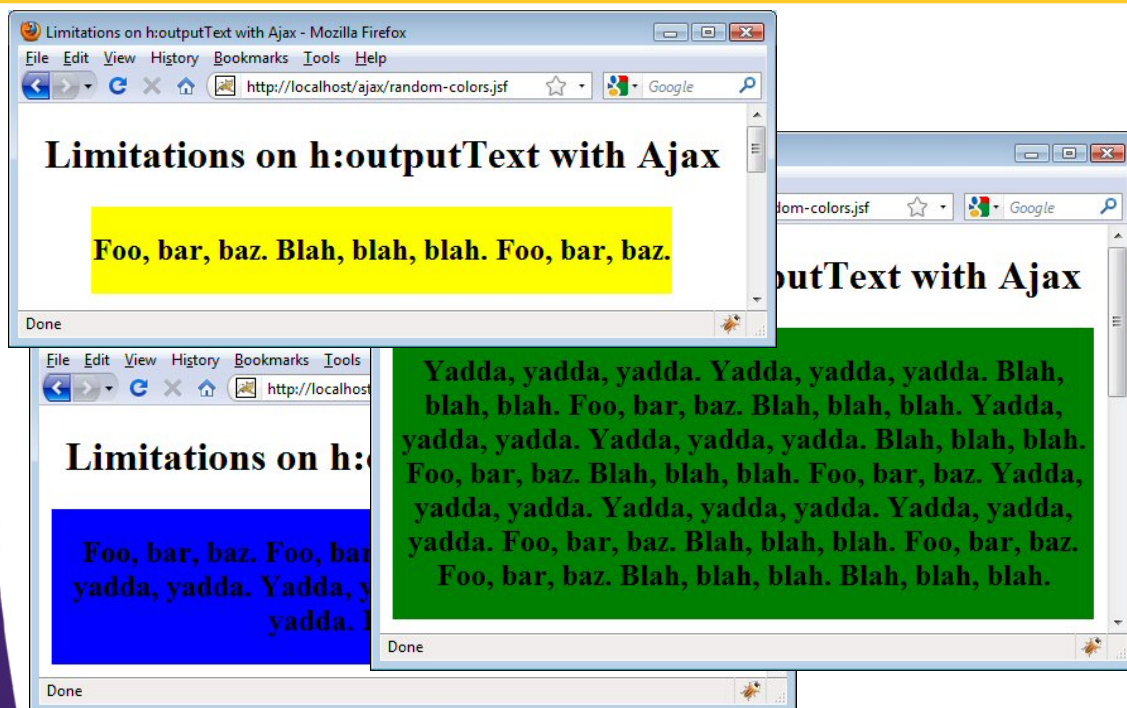
## Bean Code (Continued)

```
public String getRandomColor() {
    return(RandomUtils.randomElement(colors));
}

public String getRandomText() {
    int numPhrases = 1+ RandomUtils.randomInt(20);
    String text = "";
    for(int i=0; i<numPhrases; i++) {
        text += RandomUtils.randomElement(phrases);
    }
    return(text);
}
```

73

## Results



74

## Example: <h:outputText.../> Generating Entire Tags

- **Idea**

- Every time page is loaded, use different colors and different text for a specified region

- **Approach**

- To generate colors
  - <h:outputText value="#{textGenerator.startCell}" escape="false"/>
    - Similar for endCell to output </th>
- To generate text
  - <h:outputText value="#{textGenerator.randomText}"/>

- **Point**

- Color generation still cannot be ajaxified
  - Even though colors built with h:outputText, once you add id, it will become a span, so cannot output a start tag

75

## Facelets Code

```
<table>
  <tr><h:outputText value="#{textGenerator.startCell}"
                    escape="false"/>

    <h2>
      <h:outputText value="#{textGenerator.randomText}"/>
    </h2>
    <h:outputText value="#{textGenerator.endCell}"
                  escape="false"/>

  </tr>
</table>
```

76

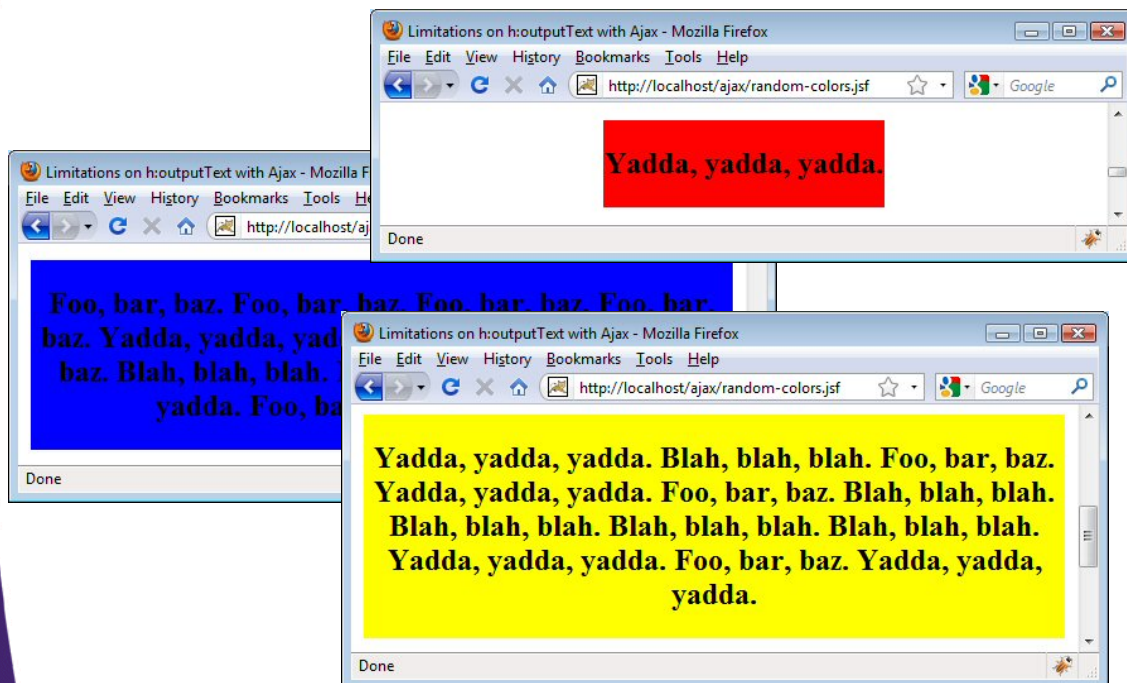
# Bean Code

```
public String getStartCell() {
    String text =
        String.format("<th bgcolor='%s'>",
                      getRandomColor());
    return(text);
}

public String getEndCell() {
    return("</th>");
}
```

77

# Results



78

## Example: Ajaxifying

- **Idea**

- When you press the button, change the text (without reloading page). Color does not change until page reloads

- **Approach**

- To generate colors
    - `<th bgcolor="#{textGenerator.randomColor}">`
      - Cannot be ajaxified, so go back to simpler approach of 1<sup>st</sup> example
  - To generate text
    - `<h:outputText value="#{textGenerator.randomText}" id="foo"/>`

- **Point**

- Some uses of `#{...}` and `<h:outputText value="#{...}"/>` cannot be updated with Ajax

79

## Facelets Code

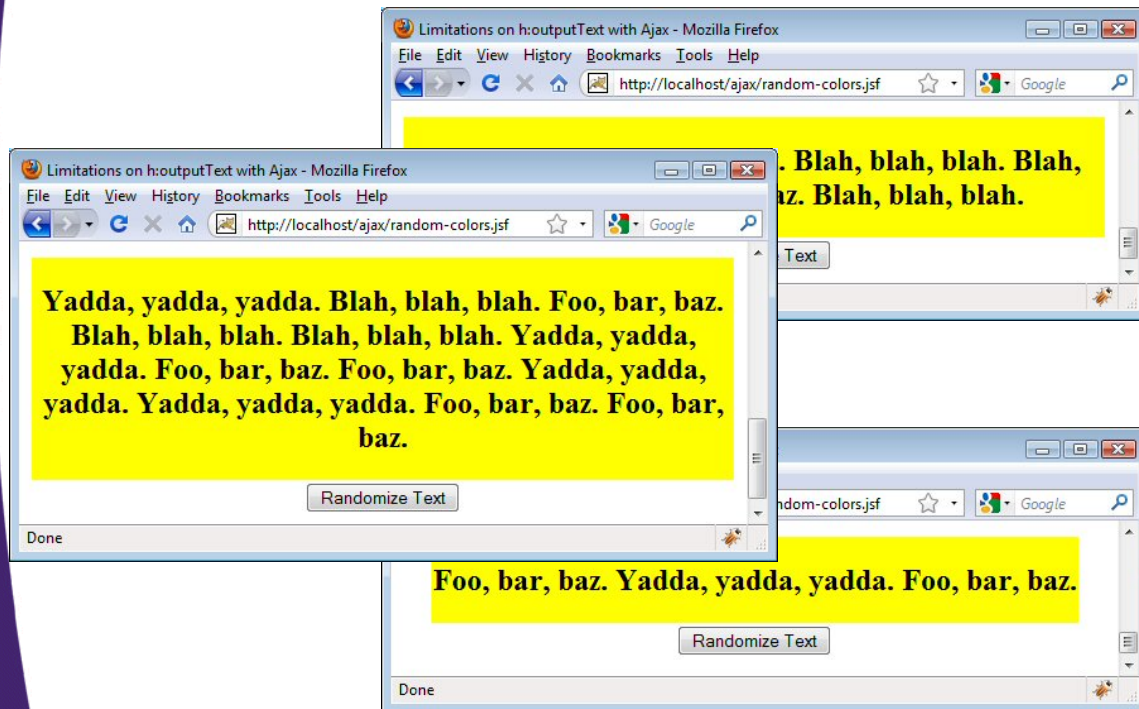
```
<h:form>
<table>
  <tr><th bgcolor="#{textGenerator.randomColor}">
    <h2><h:outputText value="#{textGenerator.randomText}"
                      id="foo"/></h2>
  </th></tr>
</table>
<h:commandButton value="Randomize Text">
  <f:ajax render="foo"/>
</h:commandButton>
</h:form>
```

Uses same bean code as first  
dynamic-color example.

80



# Results



81

© 2012 Marty Hall



# Wrap-Up

**Customized Java EE Training:** <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Summary

- Simple example

```
<h:form>
  <h:commandButton value="Show Number"
                    action="#{numberGenerator.randomize}">
    <f:ajax render="numField1"/>
  </h:commandButton><br/>
  <h2><h:outputText value="#{numberGenerator.number}"
                    id="numField1"/></h2>
</h:form>
```

- General format

- <f:ajax render="ids to update"
 execute="ids to send to server" (or use @form)
 event="event to respond to"
 onevent="javaScriptFunctionName"/>

83

© 2012 Marty Hall



## Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.