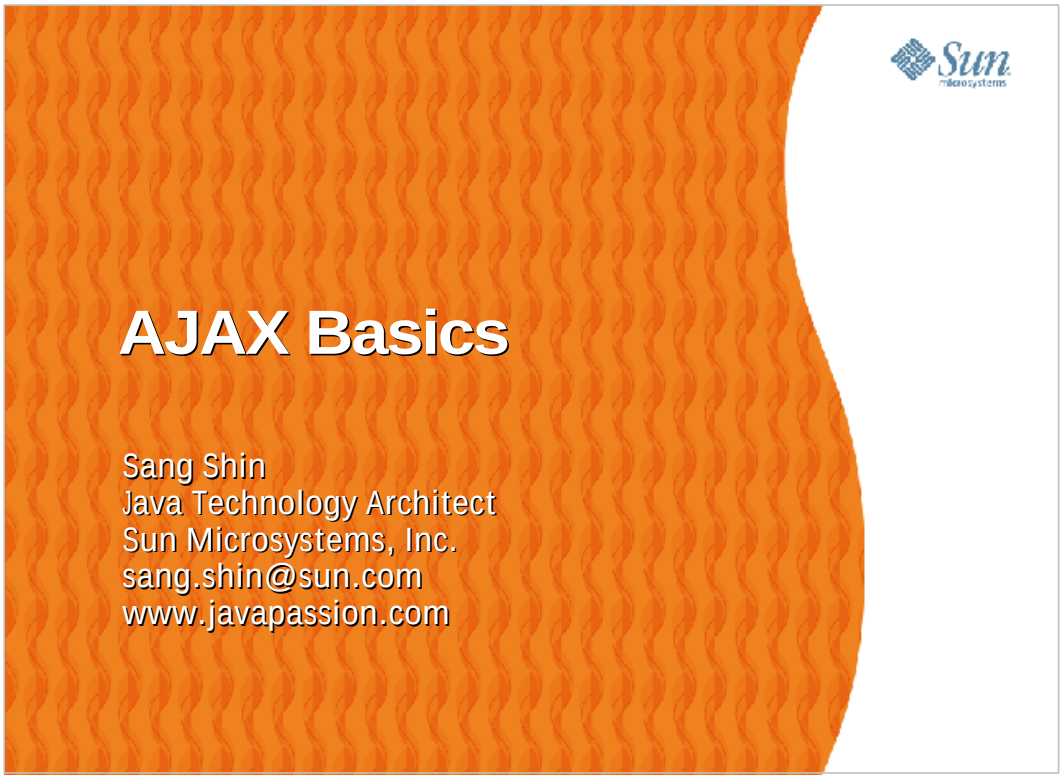


Welcome to “AJAX Basics” presentation. My name is Sang Shin. I am Java technology architect and evangelist from Sun Microsystems.

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems on any particular technology

Just to let you know, all the demos I will do in this presentations are from the “AJAX basics and development tools” hands-on lab. The hands-on lab is available from the link mentioned in the slide.

This is the agenda of this presentation. First, I will briefly talk about what we mean by “Rich User Experience” from an end-user standpoint. I will then go over several Rich Internet Application technologies that make rich user experience possible when using Web applications.

Then I will spend the rest of the presentation talking about AJAX in a bit more detail. First, I will show you the real-life examples and usage cases of AJAX to give you a sense of how AJAX is currently being used.

Then I will talk about a little bit on what is and Why AJAX describing differences between conventional and AJAX-based web applications. Then I will spend sometime talking about the technologies that are used under AJAX-based applications - mainly JavaScript, CSS, DOM, and XMLHttpRequest JavaScript object.

We will then spend some time talking about the anatomy of an AJAX operation using an example code. Here we will go over the sequence of things that occur in a typical AJAX-based interaction between a browser and the server. We will then look into the methods and properties of the XMLHttpRequest JavaScript object. We will also learn more about the XMLHttpRequest object.

In the latter part of this presentation, I will touch briefly on AJAX security issues. Then I will talk about some JavaScript debugging tools for testing and debugging AJAX applications.

I will end this presentations briefly mentioning the current issues of AJAX as a technology and things that will be coming in the future.

Topics Covered in Other Presentations

- AJAX Toolkits & Frameworks
- JSON (JavaScript Object Notation)
- Dojo Toolkit
- DWR (Direct Web Remoting)
- AJAX-enabled JSF Components
- Google Web Toolkit (GWT)
- jMaki
- Wicket and Shale (as AJAX-aware Web applicaion frameworks)
- JavaScript Programming Best Practices

4

The topics in this slide are going to be covered in other presentations.

The “AJAX Toolkits and Frameworks” presentation covers various types of AJAX toolkits and frameworks that are available today.

Then there are presentations that talks about each of these toolkits and frameworks in detail.

Now let's talk about rich user experience for web applications.

Rich User Experience

- Take a look at a typical desktop application (Spreadsheet app, etc.)
- The program responses intuitively and quickly
- The program gives a user meaningful feedback's instantly
  - > A cell in a spreadsheet changes color when you hover your mouse over it
  - > Icons light up as mouse hovers them
- Things happen naturally
  - > No need to click a button or a link to trigger an event

6

In order to understand what we mean by rich user experience, we can take a look at a typical desktop application we use, for example, spreadsheet.

When we use standalone desktop application, we expect the application to respond to what we do at the moment intuitively and quickly, giving us meaning feed-backs. For example, if you are hovering your mouse over a cell in a spreadsheet, the color of the cell might change.

In using the desktop application, things happen more naturally. For example, you don't have to click a button or link to make things happen. Instead, you move your mouse around and things are responding accordingly.

Characteristics of Conventional Web Applications

- "Click, wait, and refresh" user interaction
- Page refreshes from the server needed for all events, data submissions, and navigation
- The user has to wait for the response
- Synchronous "request/response" communication model
- Page-driven: Workflow is based on pages
- Page-navigation logic is determined by the server

7

So these are characteristics, rather limitations of the conventional web applications.

First, in a conventional web application, a user interaction is basically a repeated sequence of "click, wait and refresh" cycles. In fact, a total page refresh has to occur for anything you do with the server whether it is to send a small piece of data to the server, or retrieving a small piece of data from the server.

Then the user has to wait for the arrival of the response from the server. This interaction is considered synchronous in the sense that the user has to wait until a response from the server is received. Of course, we all know what it means when the network is down or slow - blank screen.

The flow of conventional web application is page-driven meaning the workflow of the application is based on pages. And which page to display next is determined by the server.

Issues of Conventional Web Application

- Slow response
- Loss of operational context during refresh
  - > Loss of information on the screen
  - > Loss of scrolled position
- No instant feedback's to user activities
  - > A user has to wait for the next page

These are the reasons why Rich Internet Application (RIA) technologies were born.

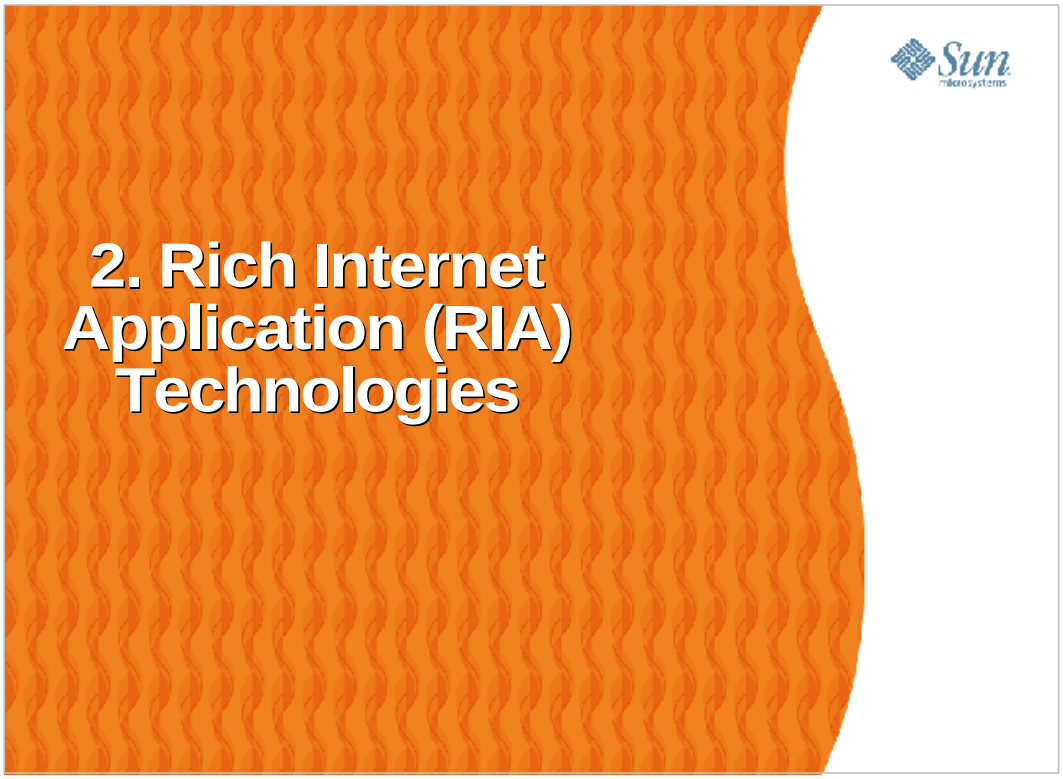
8

The conventional behavior of web application we talked in the previous slide has several limitations. First of all, it is slow both in terms of actual and especially in perceived responsiveness because the user simply cannot do anything until a response is received.

The second issue is loss of operational context during refresh. Because the total page has to be refreshed, in a sense your brain has to start from scratch in understanding each page. Another example is loss of scrolled position. If you have a page in which you have scrolled down, the next time the same page get refreshed, you have to scroll it down again.

There is no instant feedback to user activities since the user has to wait for the response from the server.

Now there have been several so called Rich Internet Application technologies that addressed some of these problems.



So let's talk about Rich Internet Application (RIA) technologies.

Rich Internet Application (RIA) Technologies

- Applet
- Macromedia Flash
- Java WebStart
- DHTML
- DHTML with Hidden IFrame
- AJAX

This is the list of RIA technologies including AJAX. So let's go over each of these in a bit more detail.

Applet

- Pros:
  - Can use full Java APIs
  - Custom data streaming, graphic manipulation, threading, and advanced GUIs
  - Well-established scheme
- Cons:
  - Code downloading time could be significant
- Use it if you are creating advanced UIs on the client and downloading time is not a major concern

11

First, let's talk about applet. Applet is Java application that gets downloaded and run within the context of a browser.

Because it is a Java application, you can use full Java APIs. And there are certain things you can do such as custom data streaming, graphic manipulation, and advanced UI operations which are not possible with other RIA technologies.

Applet technology has been around from the beginning of Java technology and is well understood. The downside is that code downloading time could be significant especially it has to be downloaded every time the user accesses it.

So the recommendation is “use applets if you are creating advanced UIs leveraging full Java APIs and code downloading time is not a major issue”.

Macromedia Flash

- Designed for playing interactive movies
- Programmed with ActionScript
- Implementation examples
  - > Macromedia Flex
  - > Laszlo suite (open source)
- Pros:
  - > Good for displaying vector graphics
- Cons:
  - > Browser needs a Flash plug-in
  - > ActionScript is proprietary

12

The next RIA technology is Macromedia Flash. Flash was designed for playing interactive movies. It does come with its own scripting language called ActionScript.

The implementation examples of flash technology is Macromedia Flex or Laszlo suite which is open source based.

The nice thing about Flash is that because of its origin, it is very good for displaying vector graphics. A downside is that a browser has to have a flash plug-in in order to run Flash. Another downside is that ActionScript is proprietary technology.

Java WebStart

- Desktop application delivered over the net
  - Leverages the strengths of desktop apps and applet
- Pros
  - Desktop experience once loaded
  - Leverages Java technology to its fullest extent
  - Disconnected operation is possible
  - Application can be digitally signed
  - Incremental redeployment
- Cons
  - Old JRE-based system do not work
  - First-time download time could be still significant

13

Java Web Start technology basically lets a website to deliver desktop Java application over the net. In that sense, it is leveraging the strengths of applet and desktop applications.

The strength of an applet is its distribution model in which a user automatically gets always up to date code. The weakness of the applet compared to desktop application, however, is that in order to use applet you have to have a network connection.

The strength of a desktop application, on the other hand, is that a user can use the application in disconnected mode while the major weakness of it is that it has to be manually installed on each client.

By using Java WebStart technology, you can install the desktop application as if it is an applet. Once installed, a Java application functions as if it is a standalone desktop application. Now each time you use the application, it checks automatically if there is a newer version or not. And only the pieces that are changed will be downloaded, which is called incremental redeployment.

The downside of Java Web Start is that the old JRE-base system, JDK 1.1 and below, do not work with it since it is introduced from JDK 1.2 and the first time download time could be significant.

DHTML (Dynamic HTML)

- DHTML = JavaScript + DOM + CSS
- Used for creating interactive applications
- No asynchronous communication, however
  - > Full page refresh still required
  - > Reason why it has only a limited success

14

Dynamic DHTML is basically combination of JavaScript, DOM and CSS. And it has been used for creating interactive and responsive Web applications. However, DHTML alone still does not provide asynchronous communication, which means the full page refresh is still required. And this is the reason why DHTML has only a limited success.

DHTML with Hidden IFrame

- IFrame was introduced as a programmable layout to a web page
  - > An IFrame is represented as an element of a DOM tree
  - > You can move it, resize it, even hide it while the page is visible
- An invisible IFrame can add asynchronous behavior
  - > The visible user experience is uninterrupted – operational context is not lost
- It is still a hack

15

IFrame was introduced a few years ago as a programmable layout to a web page. What this means is that an iframe is represented as an element of a DOM tree and you can move it, resize it, or even hide it while the parent page that includes it is still visible to a user.

Now what people found is that when you make the iframe as an hidden element, it can add asynchronous behavior to the web application meaning a user can continue to function while the iframe is retrieving the page in the background. This is basically providing AJAX-like behavior in which the visible end user experience is uninterrupted.

The downside is that iFrame is being used for the purpose it was not designed for. And there are in fact several limitations compared AJAX, which I am not going into detail here.

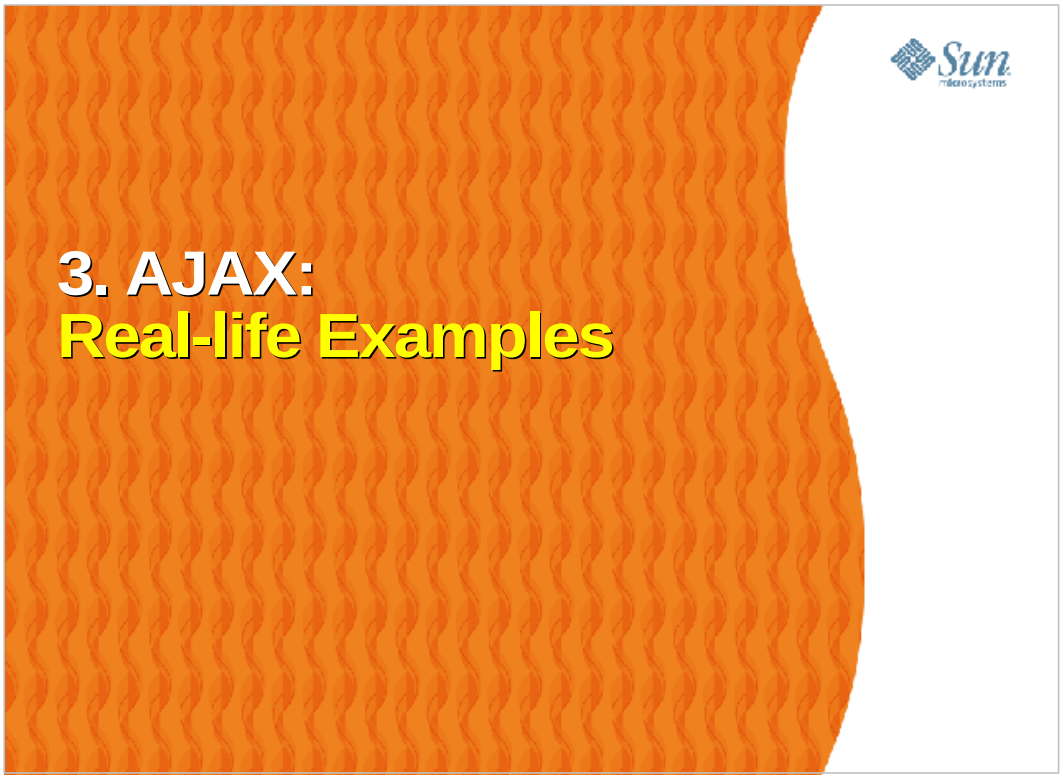
AJAX

- DHTML plus Asynchronous communication capability through XMLHttpRequest
- Pros
  - > Most viable RIA technology so far
  - > Tremendous industry momentum
  - > Several toolkits and frameworks are emerging
  - > No need to download code & no plug-in required
- Cons
  - > Still browser incompatibility
  - > JavaScript is hard to maintain and debug
- AJAX-enabled JSF components will help

16

You can think of AJAX as DHTML plus asynchronous communication capability through XMLHttpRequest JavaScript object.

AJAX is probably the most viable RIA technology so far. And there is a tremendous industry momentum behind AJAX. There are several toolkits and frameworks that can be used for developing production quality AJAX applications. The nice thing about AJAX compared to other RIA technologies is that you don't have to download or configure a plug-in to make it work.



Now let me show you some real life examples of AJAX for people who have not seen it or for people who have seen it but did not realize they were using AJAX underneath.

Real-Life Examples of AJAX Apps

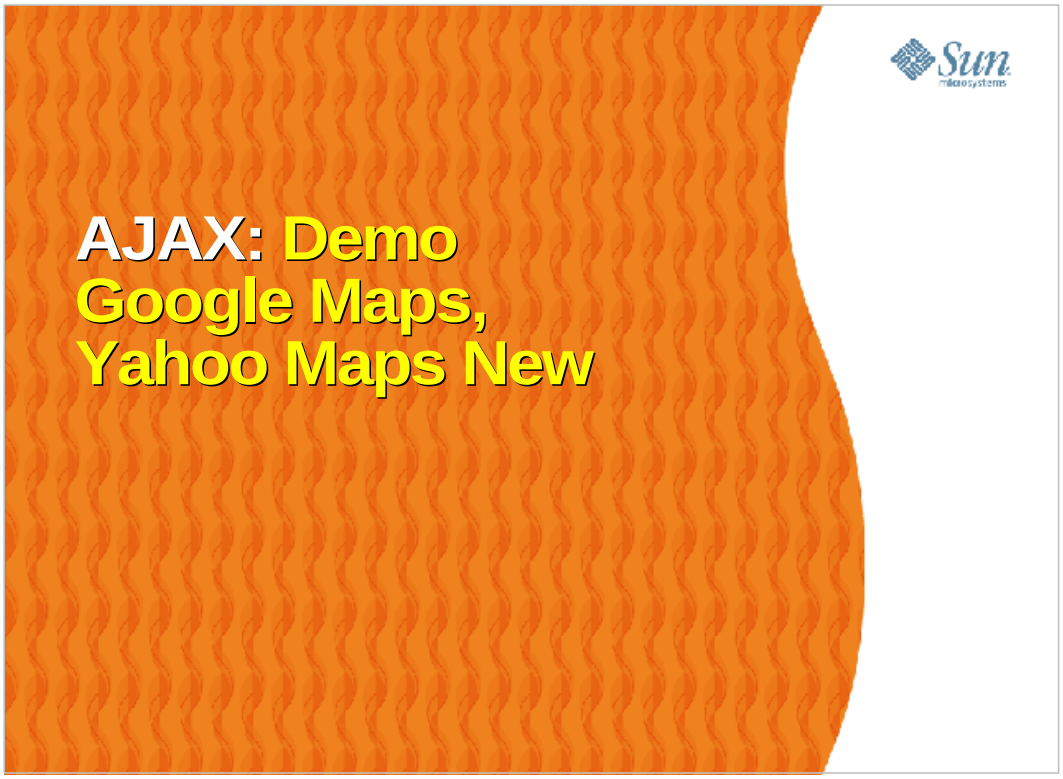
- Google maps
  - > http://maps.google.com/
- Goolgle Suggest
  - > http://www.google.com/webhp?complete=1&hl=en
- Gmail
  - > http://gmail.com/
- Yahoo Maps (new)
  - > http://maps.yahoo.com/
- Many more are popping everywhere

18

Everybody knows Google maps now. And it is highly likely everybody has used Google maps. Google maps is the one that really triggered the AJAX phenomenon.

Other applications that use AJAX underneath include Google Suggest which is basically auto-completion. And more comprehensive use of AJAX is Gmail application.

Yahoo is revamping their sites using AJAX. And more and more Web sites are beginning to use AJAX.



Now let me do a bit of demo of Google maps and Yahoo's new maps.

Key Aspects of Google Maps

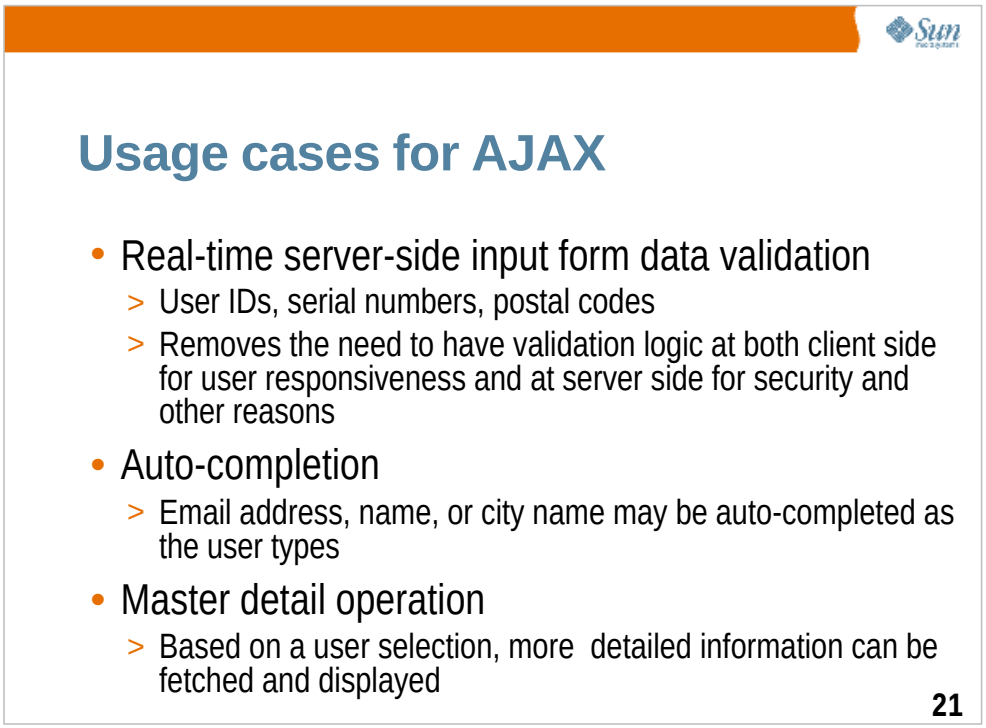
- A user can drag the entire map by using the mouse
  - Instead of clicking on a button or something
- The action that triggers the download of new map data is not a specific click on a link but a moving the map around
- Behind the scene - AJAX is used
  - The map data is requested and downloaded asynchronously in the background
- Other parts of the page remains the same
  - No loss of operational context

20

Now let me talk about key functional aspects of the Google maps.

First, you can drag the entire map by moving the mouse. In order to see other parts of the map, all you have to do is moving the mouse instead of clicking a button. What happens behind the scene is that the mouse movement triggers the downloading of the new map data in the background.

Please also note that the other parts of the page remain the same. This means there is no loss of operational context.



Some usage cases for AJAX interactions are the following:

* Real-Time Server side input form Data Validation: Form data such as user IDs, serial numbers, postal codes, or even special coupon codes that require server-side validation can be performed through AJAX. Now in general, applications have to have both client side validation and server side validation, the former for the purpose of user responsiveness and the latter due to security and other reasons such as the validation has to use backend data. Now having validation logic in two different places causes the maintenance problem of the application. Now using AJAX, you removes the need to have validation logic in two different places.

* Auto-completion: A specific portion of form data such as an email address, name, or city name may be autocompleted as a user types.

* Master Details Operations: Based on what a user selects, more detailed information can be retrieved and displayed. For example, if a user selects a product on a online shopping page, detailed information on that product can be retrieved and displayed without refreshing the page.

Usage cases for AJAX

- Advanced GUI widgets and controls
  - > Controls such as tree controls, menus, and progress bars may be provided that do not require page refreshes
- Refreshing data
  - > HTML pages may poll data from a server for up-to-date data such as scores, stock quotes, weather, or application-specific data
- Simulating server side notification
  - > An HTML page may simulate a server-side notification by polling the server in the background

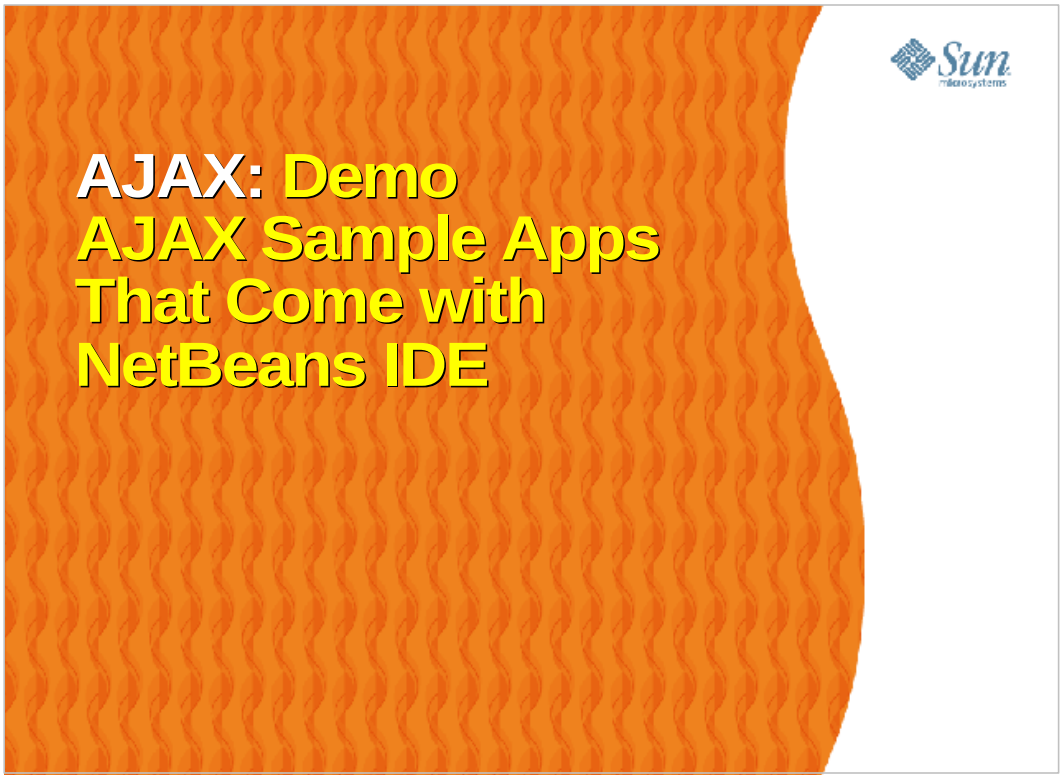
22

This is the more example usage cases of AJAX technology.

* Advanced User Interface Controls: Controls such as tree controls, menus, and progress bars may be provided that do not require page refreshes.

* Refreshing Data on the Page: HTML pages may poll data from a server for up-to-date data such as scores, stock quotes, weather, or application-specific data.

* Server-side Notifications: An HTML page may simulate a server-side notification by polling the server for event notifications that may notify the client with a message, refresh page data, or redirect the client to another page.

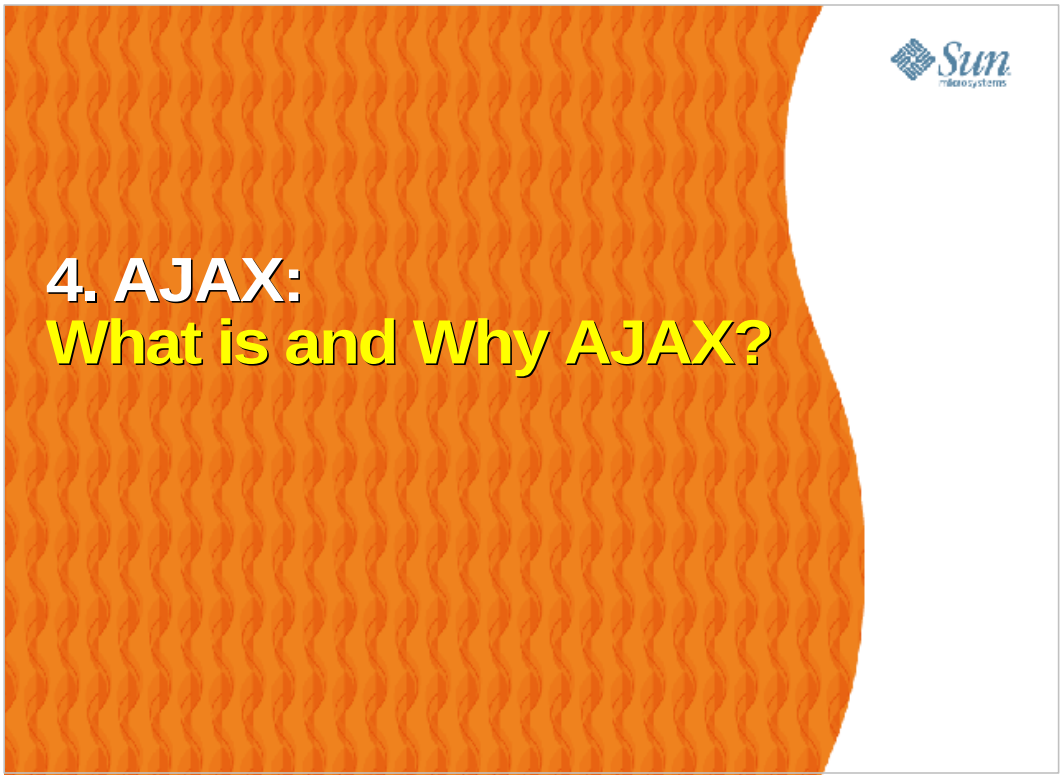


OK, let me give you demos on running AJAX sample applications that come with NetBeans IDE.

Demo Scenario

- Run sample AJAX applications within NetBeans IDE
  - Auto completion
  - Data validation
  - Progress bar
- You can try this demo yourself
  - These applications are provided as built-in sample applications in NetBeans

NetBeans comes with several AJAX sample applications. Here I am going to run “Auto completion”, “Data validation”, and “Progress bar” sample applications.



So far, we talked about issues of conventional applications and we have seen how AJAX can address these issues.

Now let's talk about what AJAX is from a bit more “under the hood” standpoint.

Why AJAX?

- Intuitive and natural user interaction
  - No clicking required
  - Mouse movement is a sufficient event trigger
- "Partial screen update" replaces the "click, wait, and refresh" user interaction model
  - Only user interface elements that contain new information are updated (fast response)
  - The rest of the user interface remains displayed without interruption (no loss of operational context)
- Data-driven (as opposed to page-driven)
  - UI is handled in the client while the server provides data

26

As you have seen, AJAX based web applications allow more intuitive and natural user interaction. For example, unlike conventional web application, you don't need to do a click a link or button to trigger an event. A mouse movement or typing a character into a input form field could be a sufficient event trigger.

Under AJAX-based application, partial screen update replaces the old “click, wait a refresh the whole page” model. Partial screen update means only the element that contain new information is updated while the rest of the page remains the same so that you do not lose the operational context.

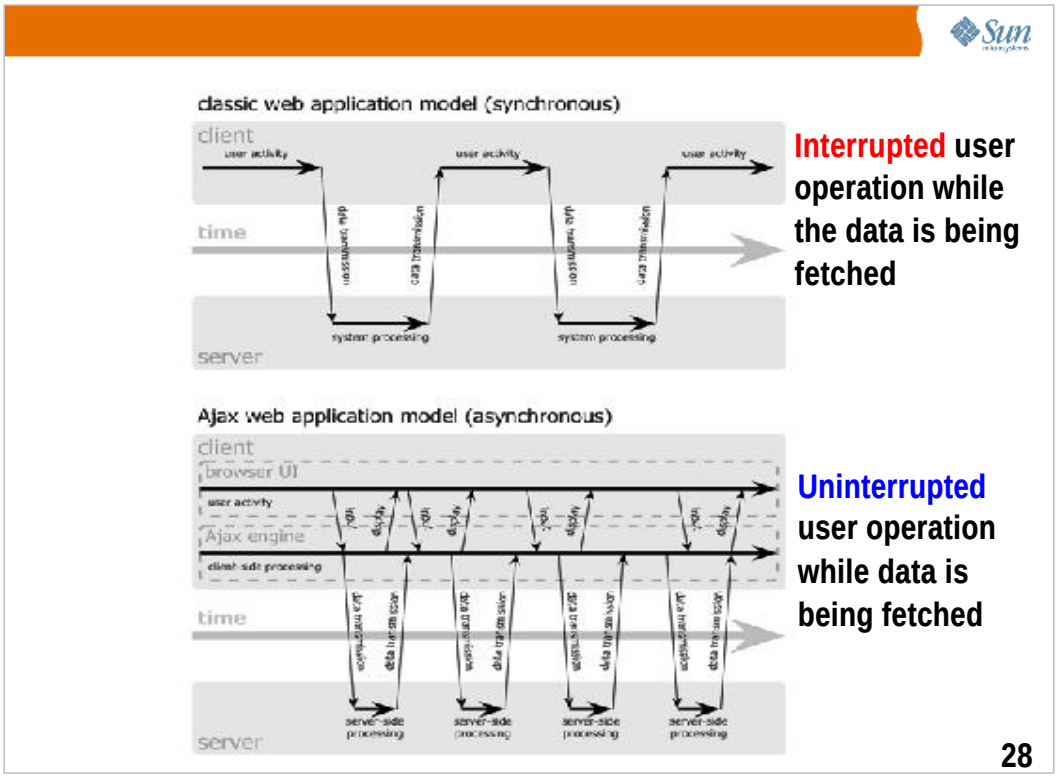
Another aspect of AJAX is AJAX enables the interaction between the client and server to be data-driven rather than page-driven. In other words, the UI is handled by the client while the data is provided by the server. This generally results in higher performance,reduced bandwidth, reduced work load on the server, and a better end user experience

Why AJAX?

- Asynchronous communication replaces "synchronous request/response model."
  - A user can continue to use the application while the client program requests information from the server in the background
  - Separation of displaying from data fetching

27

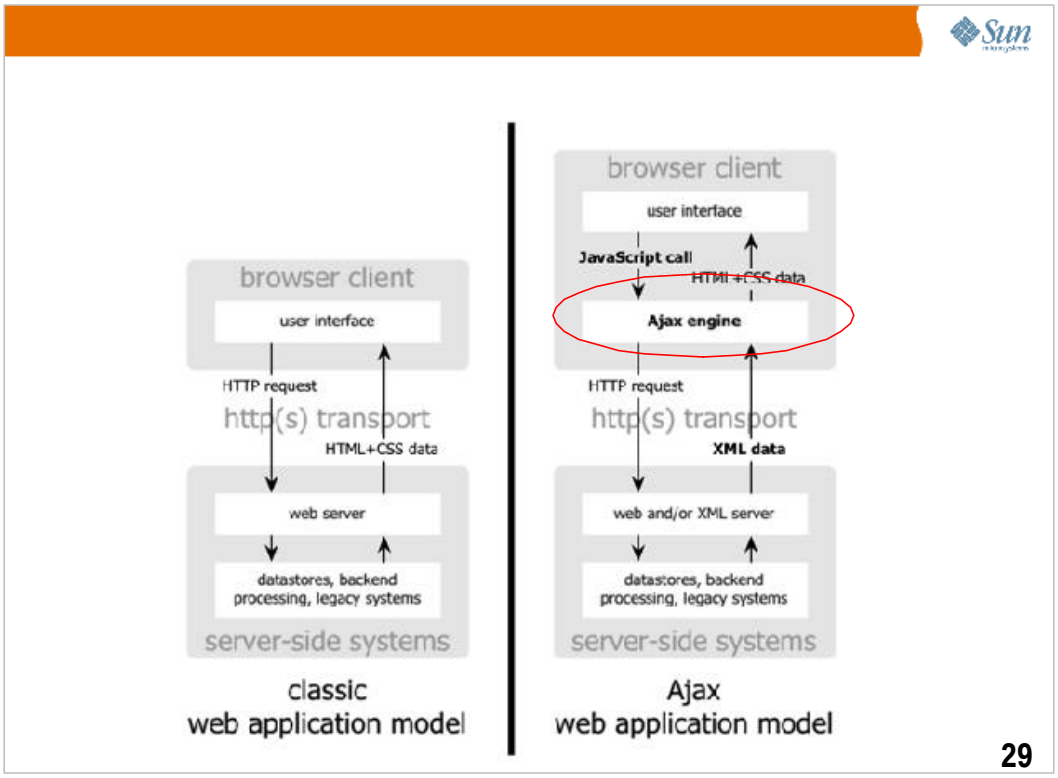
As its name implies, AJAX allows asynchronous communication between the browser and the backend server. What this means to a user experience is that a user can continue to function while the browser performs HTTP request/response interaction asynchronously in the background. In other words, the data fetching operation is done independently from displaying the information under AJAX-based interaction model.



I assume some of you have seen this picture before. This picture compares the user interaction models of conventional web application in the top half of the slide against AJAX-fied web application in the bottom part of the slide.

On the top, it is how a user interaction occurs when conventional web application is used. Here a user has to stop and wait until a response is received. What this means is that data fetching activity interrupts the user operation.

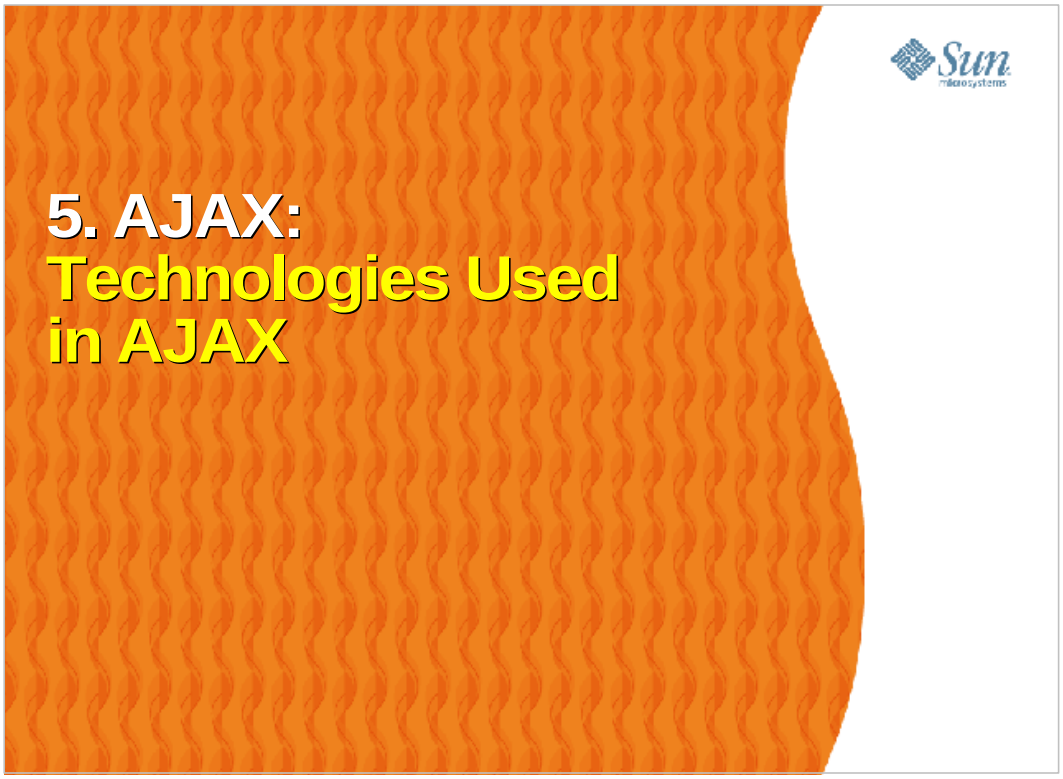
The bottom picture shows how a user interaction occurs for AJAX application. In this model, a user keeps operating since the data fetching occurs in the background.



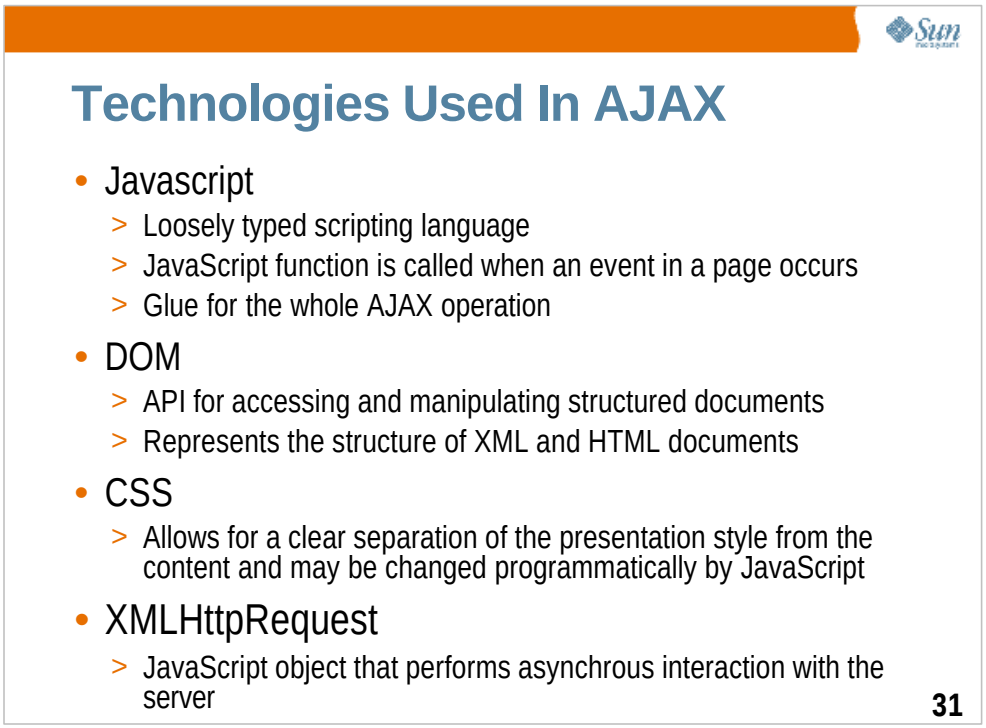
So how does AJAX application performs the asynchronous communication with the server?

This slide also compares conventional web application and AJAX application. In the left side, which shows the conventional web application, HTTP request/response interaction occurs directly between a browser and a backend web application.

In the right side, which shows AJAX based web application, within a browser, there is AJAX engine, which is actually represented by a JavaScript object called XMLHttpRequest which handles the HTTP request/response interaction in an asynchronous fashion.



Now let's take a look at the technologies that make up an AJAX application.



This slide shows the technologies used in AJAX.

* JavaScript - JavaScript is a loosely typed object based scripting language supported by all major browsers and essential for AJAX interactions. JavaScript functions in a page are invoked as event handlers when an event in a page occurs such as a page load, a mouse click, or a key press in a form element.

* DOM - is API for accessing and manipulating structured documents. In most cases DOM represent the structure of XML or HTML documents.

* CSS - Allows you to define the presentation of a page such as fonts, colors, sizes, and positioning. CSS allow for a clear separation of the style from the content and may be changed programatically by JavaScript.

* HTTP - Understanding the basic request/response interaction model of HTTP is important for a developer using AJAX. You will be exposed to the GET and PUT method when configuring an XMLHttpRequest and HTTP response codes when processing callback.

XMLHttpRequest

- JavaScript object
- Adopted by modern browsers
  - > Mozilla™, Firefox, Safari, and Opera
- Communicates with a server via standard HTTP GET/POST
- XMLHttpRequest object works in the background for performing asynchronous communication with the backend server
  - > Does not interrupt user operation

Now let's talk about XMLHttpRequest a bit. It is a JavaScript object, which means it gets created within a JavaScript code.

The XMLHttpRequest JavaScript object is supported in most modern browsers.

It is the AJAX engine that performs the HTTP request/response interaction with the backend web application in asynchronous fashion.

Server-Side AJAX Request Processing

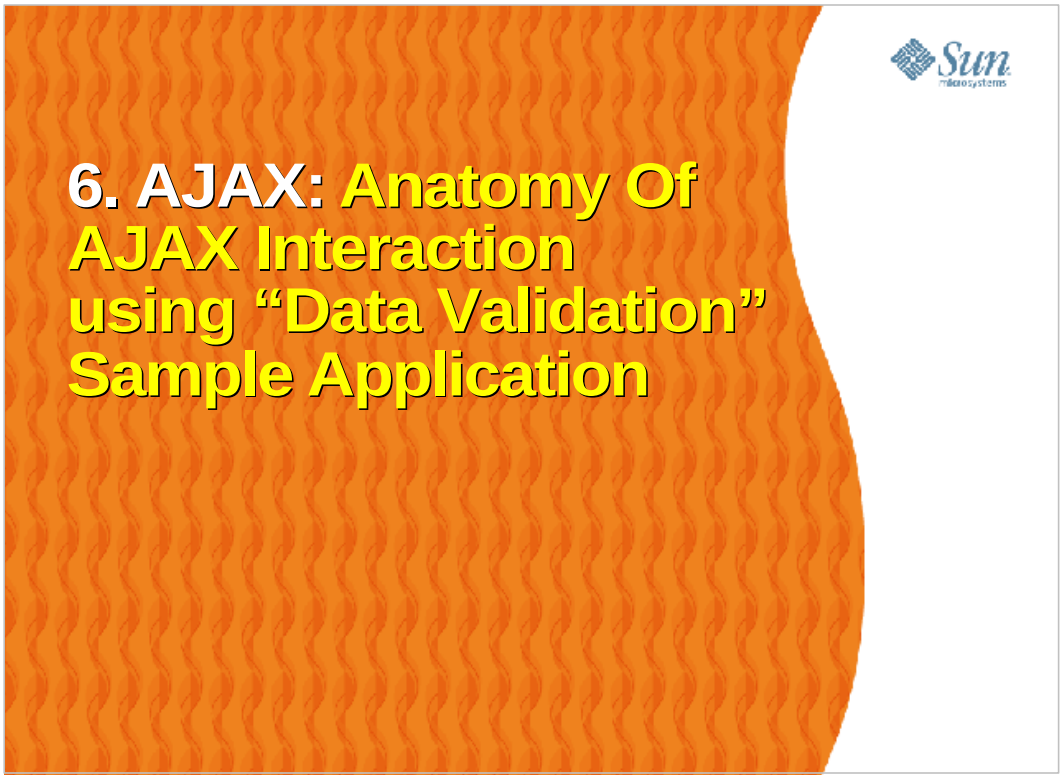
- Server programming model remains the same
  - > It receives standard HTTP GETs/POSTs
  - > Can use Servlet, JSP, JSF, ...
- With minor constraints
  - > More frequent and finer-grained requests from client
  - > Response content type can be
    - > text/xml
    - > text/plain
    - > text/json
    - > text/javascript

33

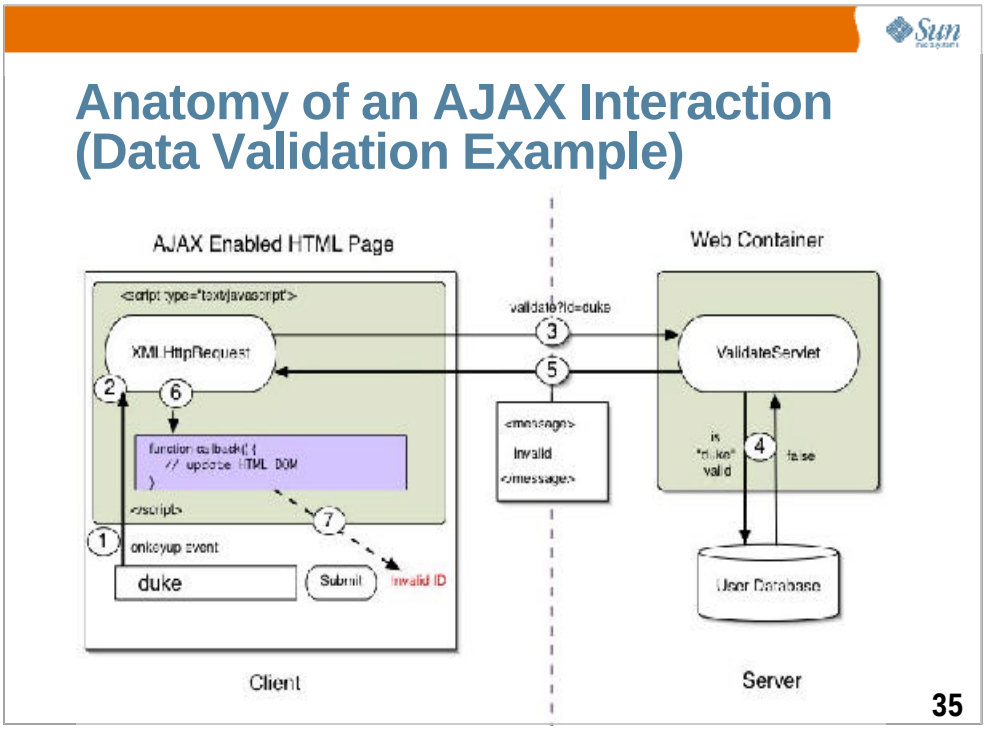
Now, how does server side web application handle the AJAX interaction with the browser? Well as far as Server side code is concerned, it is just another HTTP request that comes in. The server side does not even know the browser is sending the HTTP request in asynchronous fashion and it should not.

What this means is that the server side web application can use any server side technology of its choice such as servlet, JSP, JSF, Struts, or whatever.

The server side application however has minor constraints. First, it is possible the client might send more frequent and finer grained requests. The response type can be text/xml, text/plain, text/json, or text/javascript.



In the remaining part of the presentation, I would like to go over the anatomy of AJAX interaction using the data validation sample application.



Now that we have discussed what AJAX is and the technologies that make up the AJAX, let's put all these pieces together and see how things are working underneath.

The example code we are going to use is Data validation example.

Here a user types characters into a input form data field. That is denoted as number 1 in the slide. That will trigger an “onkeyup” event. The “onkeyup” event handler then gets called in which XMLHttpRequest object gets created and configured. That is noted as number 2 in the slide.

The XMLHttpRequest object then sends HTTP request along with what the user has typed in to the server. That is number 3 on the slide.

The server performs the validation of the data and returns its result. Those are denoted as number 4 and 5 in the slide. The callback function on the client side is then called. The callback function then changes a HTML element in the page to reflect the data that has been received from the server.

Steps of AJAX Operation

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an async. request
5. The ValidateServlet returns an XML document containing the result
6. The XMLHttpRequest object calls the callback() function and processes the result
7. The HTML DOM is updated

36

So this slide shows the steps of AJAX operation. Here I described it as 7 steps, which occur in the order specified. Now the steps in the blue color occur at the client while the one in the black color, step 5 occurs at the server.

Now let's go over each of these steps in a bit more detail.

1. A Client event occurs

- A JavaScript function is called as the result of an event
- Example: validateUserId() JavaScript function is mapped as a event handler to a onkeyup event on input form field whose id is set to “userid”

```
<input type="text"
        size="20"
         id="userid"
        name="id"
      onkeyup="validateUserId();">
```

37

The first event that triggers the whole AJAX operational sequence is that a user types a character into a input text form field, whose id is set to “userid”. This user triggered event is captured as JavaScript event called “onkeyup”. JavaScript event handler called validateUserId() is then called as a result.

2. An XMLHttpRequest object is created

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

38

The validateUserId() event handler creates an XMLHttpRequest JavaScript object through initRequest() function in this example. Please note that the way an XMLHttpRequest object gets created for Microsoft Internet Explorer is different from the way how it gets created for Mozilla type browsers. For Mozilla type browsers, you use “new XMLHttpRequest()” just like the way you create a new Java object while for Microsoft Internet Explorer, you have to create ActiveX object This is one of the browser incompatibilities that still exist.

3. An XMLHttpRequest object is configured with a callback function

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest; // callback function
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

39

Once XMLHttpRequest object is created, the next thing to do is to set the “onreadystatechange” property of the XMLHttpRequest object to a callback function. This callback function gets called asynchronously whenever state change occurred on the XMLHttpRequest. In this example, the name of the callback function is “processRequest”, which we will examine in the following slide. We will talk about possible state changes later on.

4. XMLHttpRequest object makes an async. request

```
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

- URL is set to validate?id=greg

40

Then the “open” method of the XMLHttpRequest gets called. The “open” method requires three parameters: the HTTP method, which is GET or POST; the URL of the server-side component that the object will interact with; and a boolean indicating whether or not the call will be made asynchronously or not. In this example, and in most cases, it is set to “true”, which means the HTTP request/response interaction occurs in asynchronous fashion.

5. The ValidateServlet returns an XML document containing the results (Server)

```
public  void doGet(HttpServletRequest request, HttpServletResponse  response)
    throws IOException, ServletException {

 String targetId = request.getParameter("id");

 if ((targetId != null) && !accounts.containsKey(targetId.trim())) {
       response.setContentType("text/xml");
       response.setHeader("Cache-Control", "no-cache");
       response.getWriter().write("<valid>true</valid>");
   } else {
       response.setContentType("text/xml");
       response.setHeader("Cache-Control", "no-cache");
       response.getWriter().write("<valid>false</valid>");
   }
}
```

41

Now HTTP response is created. First, the Content-Type is set to text/xml. Second, the Cache-Control is set to no-cache. Setting Cache-Control to no- cache will keep browsers from using locally caching responses. Finally it will return “true”or “false” XML fragment.

6. XMLHttpRequest object calls callback() function and processes the result

- The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the readyState of the XMLHttpRequest object

```
function processRequest() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            var message = ...;

...
```

42

Now the control is back to the client. We have seen in step 3, the processRequest function is set as a callback function whenever there isa state change in the readyState field of the XMLHttpRequest object.

The combination of two things - the readyState value of the XMLHttpRequest is 4 and the status value is 200 - indicates that the data from the server has been successfully received by the client.

7. The HTML DOM is updated

- JavaScript technology gets a reference to any element in a page using DOM API
- The recommended way to gain a reference to an element is to call
  - document.getElementById("userIdMessage"), where "userIdMessage" is the ID attribute of an element appearing in the HTML document
- JavaScript technology may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify child elements

43

Once the data has been successfully received, the browser is then ready to display a relevant message.

Now the location where you want to display a message is represented by element in the page. In your Javascript code, you get a reference to any element in a page using DOM API. And the recommended method to use is getElementById and you pass the id as a parameter. In this example, the id of the element is set as “userIdMessage”. Once you have a reference to an element, you can do pretty much anything you want to do, for example, changing the values of element attributes, element style or add/remove/modify the child elements.

In this example, you are going to add text node as a child element to the “userIdMessage” element.

```
1.  <script type="text/javascript">
2.  function setMessageUsingDOM(message) {
3.      var userMessageElement = document.getElementById("userIdMessage");
4.      var messageText;
5.      if (message == "false") {
6.          userMessageElement.style.color = "red";
7.          messageText = "Invalid User Id";
8.      } else {
9.          userMessageElement.style.color = "green";
10.         messageText = "Valid User Id";
11.     }
12.     var messageBody = document.createTextNode(messageText);
13.     // if the messageBody element has been created simple replace it otherwise
14.     // append the new element
15.     if (userMessageElement.childNodes[0]) {
16.         userMessageElement.replaceChild(messageBody,
17.                                         userMessageElement.childNodes[0]);
18.     } else {
19.         userMessageElement.appendChild(messageBody);
20.     }
21. }
22. </script>
23. <body>
24.     <div id="userIdMessage"></div>
25. </body>
```

44

Here you get a reference to element using "getElementById" method of the document object. And then depending on the data that is returned from the server - "false" or "true" -, the "messageText" variable is set to either "Invalid User Id" or "Valid User Id" text string.

A new text node is then created using the "messageText" variable. Then it is added as a child node to the "userIdMessage" element. The browser then immediately display the text on the page.

So in this page, the partial update occurs with only with userIdMessage element.

Now let's quickly go over the XMLHttpRequest methods and properties one more time.

XMLHttpRequest Methods

- open("HTTP method", "URL", syn/asyn)
  - > Assigns HTTP method, destination URL, mode
- send(content)
  - > Sends request including string or DOM object data
- abort()
  - > Terminates current request
- getAllResponseHeaders()
  - > Returns headers (labels + values) as a string
- getResponseHeader("header")
  - > Returns value of a given header
- setRequestHeader("label","value")
  - > Sets Request Headers before sending

This is the list of the methods of the XMLHttpRequest object. The most important methods are open() and send() methods. The open method configures the XMLHttpRequest object with HTTP method, destination URL, and calling mode. The send method send a request to the server. The abort terminates the current request. The rest of the methods are getting header information of the request or response message.

XMLHttpRequest Properties

- onreadystatechange
  - Set with an JavaScript event handler that fires at each state change
- readyState – current status of request
  - 0 = uninitialized
  - 1 = loading
  - 2 = loaded
  - 3 = interactive (some data has been returned)
  - 4 = complete
- status
  - HTTP Status returned from server: 200 = OK

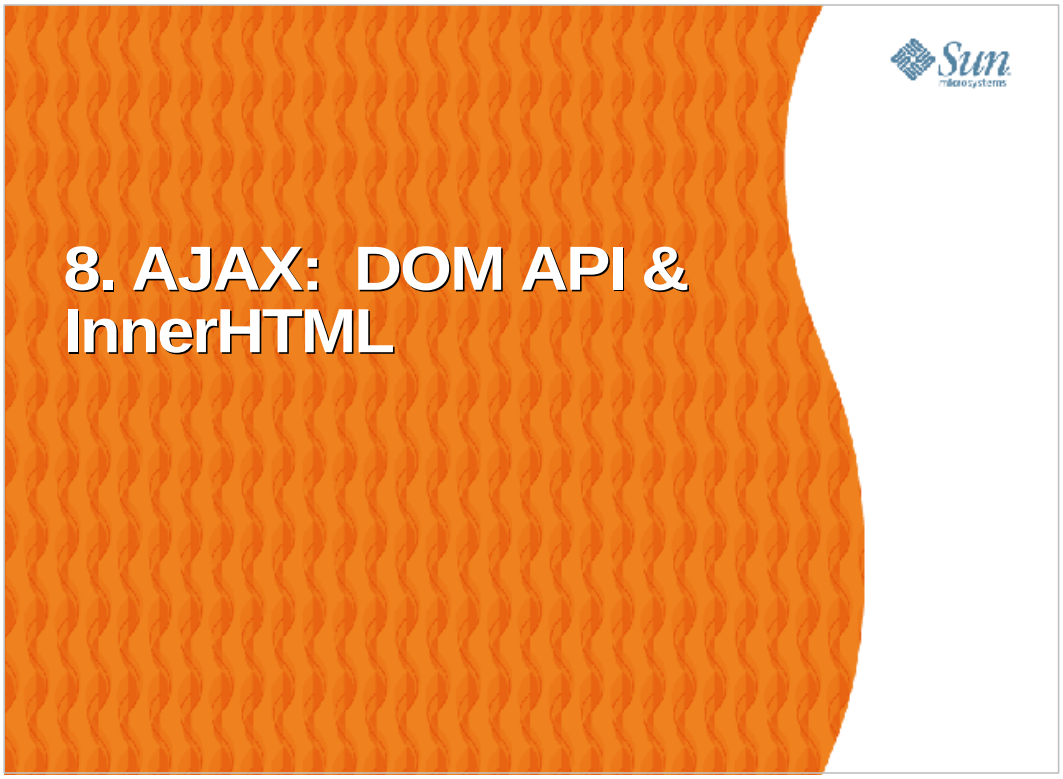
47

These are the properties of the XMLHttpRequest object. You have already seen “onreadystatechange” property, which is set with an callback event handler. The actual state is captured in the readyState property. The callback event handler set with the “onreadystatechange” property gets called every time “readyState” changes its value. The most important value is 4, which indicates that interaction with the server is complete. The status property reflects the HTTP status and the value 200 means it is a successful response.

XMLHttpRequest Properties

- responseText
  - String version of data returned from the server
- responseXML
  - XML document of data returned from the server
- statusText
  - Status text returned from server

This is a continuation of the properties of the XMLHttpRequest object. The properties on this slide reflect the data that is returned from the server. The responseText property is a string version of the data returned from the server while the responseXML property reflects the XML representation of the data.



Let's go over the DOM APIs a bit.

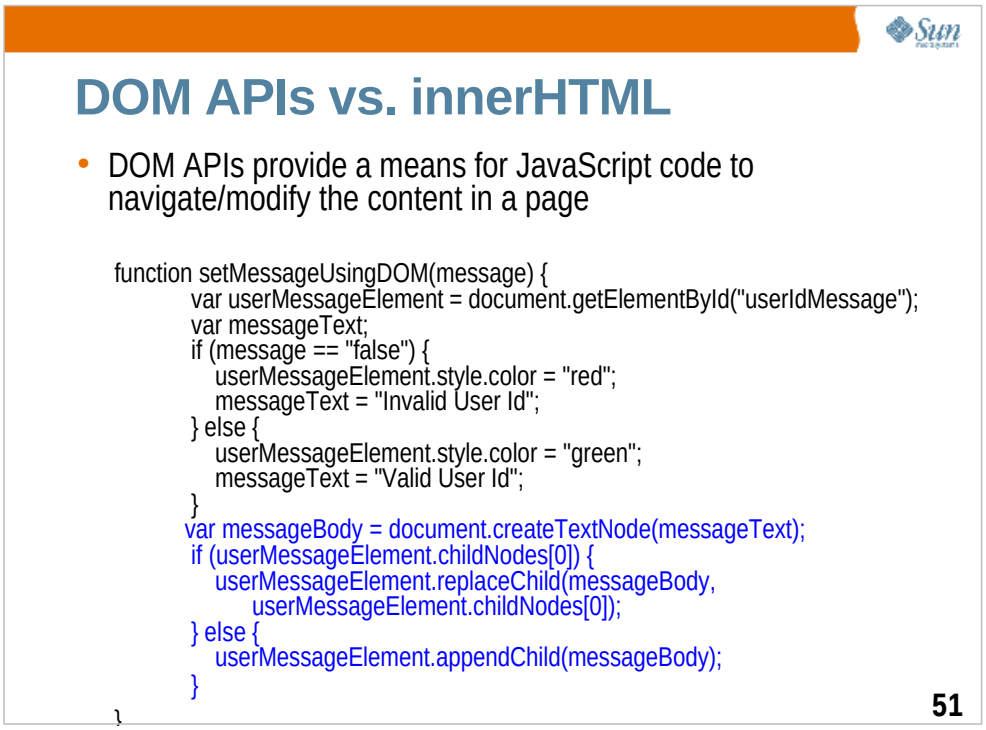
Browser and DOM

- Browsers maintain an object representation of the documents being displayed
  - In the form of Document Object Model (DOM)
  - It is readily available as document JavaScript object
- APIs are available that allow JavaScript code to modify the DOM programmatically

50

Browsers maintain an object representation of the HTML documents being displayed. And this object is in the form of DOM object.

As was mentioned, within the JavaScript code, you have a programmatic access to DOM object through DOM APIs. For example, you can access, modify or delete nodes through DOM APIs.



As we talked about, DOM APIs provide a means for you to navigate, modify the contents of a page.

You see examples of DOM APIs being used in the code of the slide.

DOM APIs vs. innerHTML

- Using innerHTML is easier: Sets or retrieves the HTML between the start and end tags of the object

```
function setMessageUsingDOM(message) {
        var userMessageElement = document.getElementById("userIdMessage");
        var messageText;
        if (message == "false") {
            userMessageElement.style.color = "red";
            messageText = "Invalid User Id";
        } else {
            userMessageElement.style.color = "green";
            messageText = "Valid User Id";
        }
        userMessageElement.innerHTML = messageText;
}
```

52

Usage of innerHTML is easier than using DOM APIs for setting or retrieving the html code fragment between the starting and ending tags of a node.

In this example, by setting the innerHTML property of the userMessageElement, the html code fragment between the starting and ending tag of the userMessageElement node, you are assigning the text node.

OK. What about Security?

AJAX Security: Server Side

- AJAX-based Web applications use the same server-side security schemes of regular Web applications
  - > You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programatic)
- AJAX-based Web applications are subject to the same security threats as regular Web applications
  - > Cross-site scripting
  - > Injection flaw

54

As for the security of AJAX web applications, they use the same security schemes of regular web applications. In other words, just like regular Web applications, you specify your authentication, authorization, and data protection requirements in the web.xml or write code that perform those security checks.

And any HTTP requests coming from the client, whether it is from regular Web application or from AJAX applications, it is constrained by the server side security schemes.

In the same vein, the AJAX-based applications are subject to the same security threats as regular web applications. For example, they are subject to such threats such as cross-site scripting or injection flaw.

AJAX Security: Client Side

- JavaScript code is visible to a user/hacker
  - > Hacker can use the JavaScript code for inferring server side weaknesses
- JavaScript code is downloaded from the server and executed ("eval") at the client
  - > Can compromise the client by mal-intended code
- Downloaded JavaScript code is constrained by sand-box security model
  - > Can be relaxed for signed JavaScript

55

Now a couple of things that are different between AJAX applications and regular applications are on the client side.

First, the JavaScript code is visible to a hacker, who could use it for inferring server side weaknesses.

Second, the JavaScript code is executed at the client. And mal-intended JavaScript code can compromise the client.

Now let's spend some time talking about development tools. These are development tools mostly for client side. The server side debugging is not that much different from what you already do.

First of all, I would like to say upfront that building AJAX applications over NetBeans is not that much different from building regular Web applications over it.

There is a JavaScript editor plug-in available for NetBeans.

Development Tools on Mozilla Browser

- Mozilla FireBug debugger (add-on)
  - > This is the most comprehensive and most useful JavaScript debugger
  - > This tool does things all other tools do and more
- Mozilla JavaScript console
- Mozilla DOM inspector (comes with Firefox package)
- Mozilla Venkman JavaScript debugger (add-on)
- Mozilla LiveHTTPHeaders HTTP monitor (similar to NetBeans HTTP monitor)

58

One of the most comprehensive and useful JavaScript debugger I found is Mozilla FireBug debugger. This tool provides features all the other tools combined and more. I will talk about the features of the FireBug in the following slide. Most of the time, FireBug debugger should be the one you want to use.

Mozilla class browsers come with a built-in JavaScript console. The DOM inspector can be installed when you install Firefox browser. You have to select custom installation.

You can install Venkman JavaScript debugger for source-code level debugging of your JavaScript code. There are are a few more development tools that are being introduced.

Another tool that you might want to consider, if you are not using NetBeans IDE, which already provides this capability, is LiveHttpHeaders HTTP monitor which captures HTTP traffic.

Mozilla FireBug Debugger

- JavaScript debugger for stepping through code one line at a time
- Status bar icon shows you when there is an error in a web page
- A console that shows errors from JavaScript and CSS
- Log messages from JavaScript in your web page to the console (bye bye "alert debugging")
- An JavaScript command line (no more "javascript:" in the URL bar)
- Spy on XMLHttpRequest traffic
- Inspect HTML source, computed style, events, layout and the DOM

59

This slide describes the features of the Mozilla FireBug debugger. First of all, you can do source code level step by step debugging.

The status bar shows whether you have a problem in your web page. The console shows syntax errors from your JavaScript or CSS code.

The Firebug lets you log messages and let them get displayed in the console.

One nice feature I liked a lot is spying on XMLHttpRequest traffic. Finally it lets you inspect HTML source, style, events, layout.

Let's go over some of the current issues in developing AJAX applications and future direction.

Current Issues of AJAX

- Complexity is increased
  - > Server side developers will need to understand that presentation logic will be required in the HTML client pages as well as in the server-side logic
  - > Page developers must have JavaScript technology skills
- AJAX-based applications can be difficult to debug, test, and maintain
  - > JavaScript is hard to test - automatic testing is hard
  - > Weak modularity in JavaScript
  - > Lack of design patterns or best practice guidelines yet
- Toolkits/Frameworks are not mature yet
  - > Most of them are in beta phase

61

Even though AJAX is picking a steam among web application developers, there are still some issues that need to be resolved.

First, complexity of the application could be increased. Server-side developers will need to understand that presentation logic will be required in the HTML client pages as well as in the server-side logic. HTML page developers must have JavaScript technology skills.

AJAX based applications can be difficult to debug, test and maintain. For example, JavaScript code is considered to be harder to do automatic testing. And design patterns and best practices still need to established.

Even though there are many toolkits and frameworks, many of them are still in version 1 or less phase.

Current Issues of AJAX

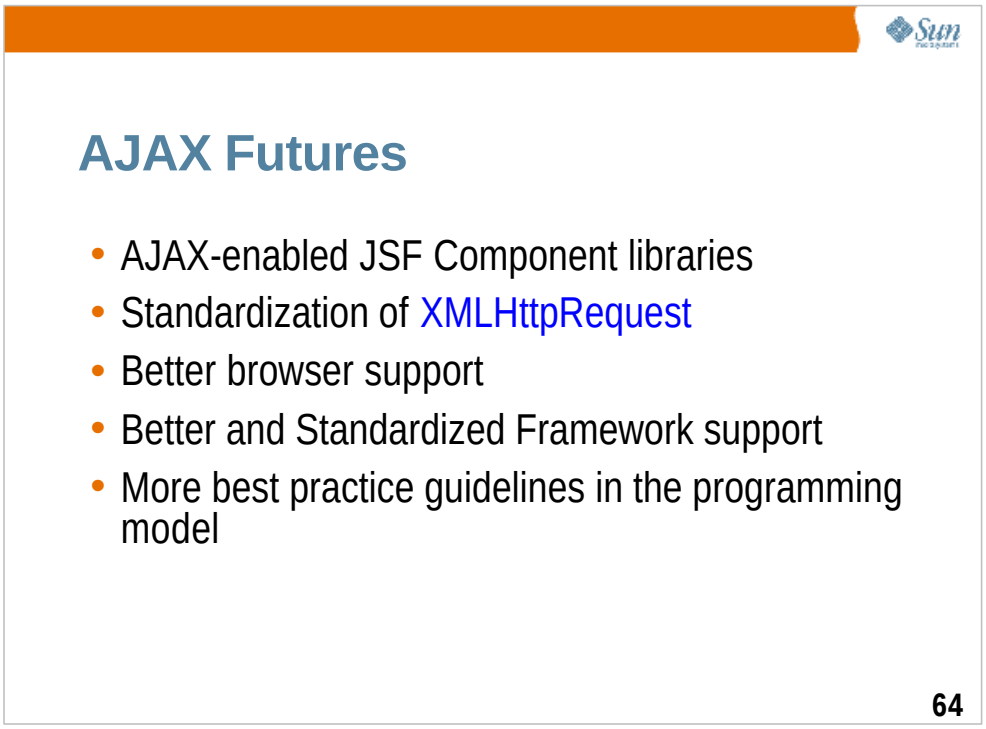
- No standardization of the XMLHttpRequest yet
  - > Future version of IE will address this
- No support of XMLHttpRequest in old browsers
  - > Iframe will help
- JavaScript technology dependency & incompatibility
  - > Must be enabled for applications to function
  - > Still some browser incompatibilities
- JavaScript code is visible to a hacker
  - > Poorly designed JavaScript code can invite security problem

The slide shows the current issues of the AJAX as a technology.

Browsers Which Support XMLHttpRequest

- Mozilla Firefox 1.0 and above
- Netscape version 7.1 and above
- Apple Safari 1.2 and above.
- Microsoft Internet Exporer 5 and above
- Konqueror
- Opera 7.6 and above

This is the list of browsers and their versions that support XMLHttpRequest, thus supporting AJAX behavior.



Now what could be the future of AJAX?

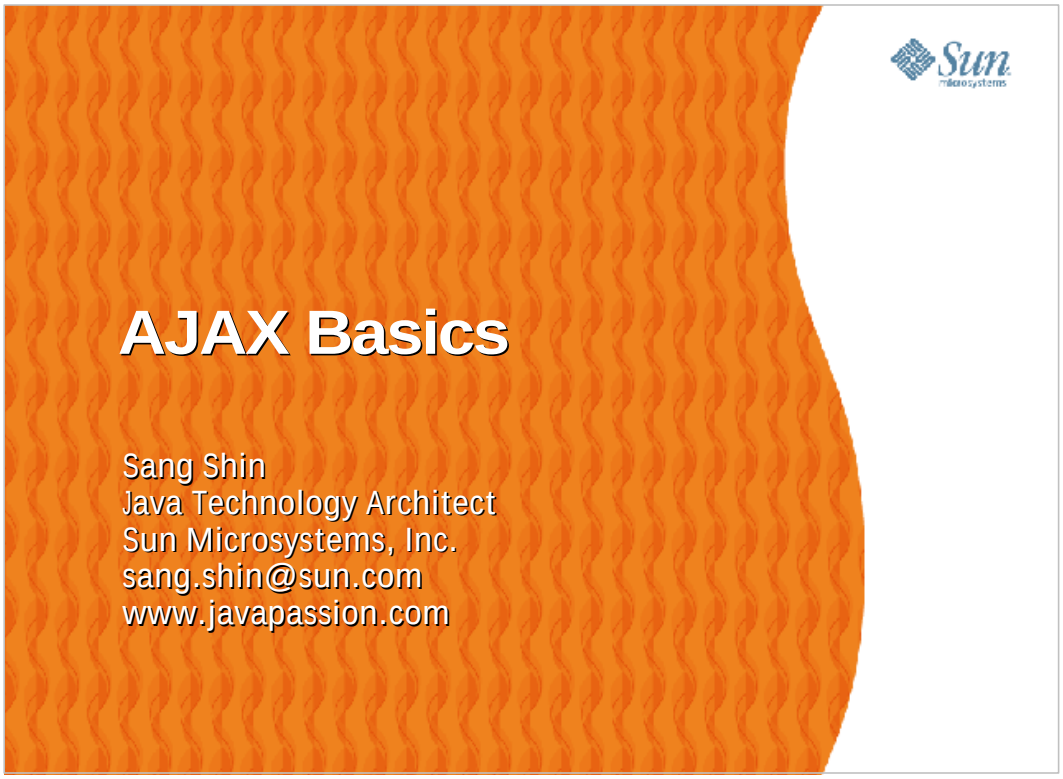
First of all, more and more AJAX-enabled JSF components will be developed and available so that you can build your Web applications with AJAX behavior but with minimum development effort.

The way that the XMLHttpRequest object gets created will be standardized. As I mentioned before, Microsoft is committed to use the standard way of creating it.

The future generation of browsers will be more and more AJAX friendly.

And further enhancement in framework support will continue. Of course, people will identity and share best practice and design patterns for developing AJAX applications.

Creating AJAX-enabled applications will become easier as new frameworks are created and existing frameworks evolve to support the interaction model.



Thanks for attending the presentation.