ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

IT4552 – Web programming

Chapter 11. Javascript & AJAX

1

Content

2

1.1. What is Javascript (JS)?

- Originally to be called LiveScript
  - Developed by Netscape
- Relationship to Java?
  - Not directly, but …
    - Shares syntax, keywords
    - Named to take advantage of Java mania
- Variants
  - Microsoft developed its own JScript (mostly the same)
  - A common subset of both is standardized as ECMAscript

3

1.1. What is Javascript (2)?

- More than form validation
- Client-Side Scripting Language
  - Dynamic
  - Weakly Typed
  - Object-Oriented (Prototype-Based)
- Interpreted

4

JavaScript vs. Java

- JavaScript
  - Cannot draw, multi-thread, network or do I/O
- Java
  - Cannot interact with browser or control content
- JavaScript is becoming what Java was originally intended to be
  - Java applets: lightweight downloadable programs run within the browser for cross-platform compatibility
  - JS: lightweight and accomplish most of what applets do with a fraction of the resources

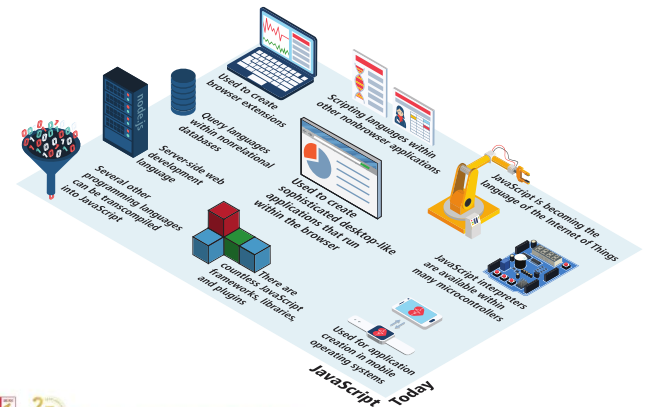
5

What is JS used for today?

- Handling User Interaction
  - Checking for accuracy and appropriateness of data entry from forms
  - Doing small calculations/manipulations of forms input data
  - Search a small database embedded in the downloaded page
  - Save data as cookie
- Generating Dynamic HTML documents
- Examples:
  - Bookmarklets, Google Maps, Google Suggest

6

What is JS used for today?



7

1.2. JS syntax basic

- Variable declaration
  - Explicit: var i = 12; // no 'var' in declaration
  - Implicit: i = 12;
- Variable scope
  - Global
    - Declared outside functions
    - Any variable *implicitly defined*
  - Local
    - *Explicit declarations inside functions*

8

1.2. JS syntax basic (2)

- JavaScript Hoisting
  - JavaScript Declarations are hoisted
  - Hoisting is JavaScript's default behavior of **moving all declarations to the top of the current scope** (to the top of the current script or the current function).
  - In JavaScript, a variable can be declared after it has been used. In other words; a variable can be used before it has been declared.

```
x = 5; // Assign 5 to x

elem =
document.getElementById("demo");
elem.innerHTML = x;

var x; // Declare x
```

```
var x; // Declare x
x = 5; // Assign 5 to x


elem =
document.getElementById("demo");
elem.innerHTML = x;
```


9

1.2. JS syntax basic (3)

- JavaScript Hoisting

```
var greeting = "hey hi";
var times = 4;

if (times > 3) {
   var greeting = "say Hello instead";
}

console.log(greeting); //"say Hello instead"
```


10

1.2. JS syntax basic (4)

- **let** and **const** keywords
  - Variables defined with let and const are hoisted to the top of the block, but not initialized.
- The let keyword was introduced in ES6 (2015).
  - Variables defined with let cannot be Redeclared.
  - Variables defined with let must be Declared before use.
  - Variables defined with let have Block Scope.

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has
already been declared
```

```
{
  let x = 2;
}
// x can NOT be used here
```


11

a. JS Variables and Literals

- Dynamic Typing - Variables can hold any valid type of value:
  - Number … var myInt = 7;
  - Boolean … var myBool = true;
  - Array … var myArr = new Array();
  - String … var myString = “abc”;
  - … and can hold values of different types at different times during execution

12

b. JS Operators

- Key Comparison Operators
  - >, <, <=, >=, !=, ==,
  - !, ||, &&
- Key Assignment Operators
  - +, -, *, /, %
  - =, +=, -=
  - ++, --

c. JS control statements

```
if(bork) {
  //...
} else {
  //...
}

switch(bork) {
  case 1:
  // if bork == 1...
  case 'whee':
  // if bork == 'whee'...
  case false:
  // if bork == false...
  default:
  // otherwise ...
}
```

```
while(bork){
  //...
}

for(var i = 0; i< 10; i++){
  //...
}

for(var element in array_of_elements){
  //...
}

do {
  //...
} while(bork);

try {
  //...
} catch(err){
  //...
}
```


d. JS functions

- Function declaration
  - Using the **`function`** reserved word
  - The return value and the types of the arguments are not declared
- Examples:

```
function square(x) { return (x * x); }
function factorial(n) {
  if (n <= 0) { return (1); }
  else {
  return (n * factorial(n - 1));
  }
}
```


e. JS functions (2)

- Calling a function
  - **`myFunc(arg1, arg2, …);`**
- Variable function
  - Functions can be passed and assigned to variables
  - Example

```
var fun = Math.sin;
alert("sin(pi/2)=" + fun(Math.PI/2));
```

e. JS functions (3)

❖ Anonymous Function Expressions

```
// defines a function using an anonymous
// function expression
var calculateSubtotal = function(price,quantity) {
      return price * quantity;
};
// invokes the function
var result = calculateSubtotal(10,2);
```


17

e. JS functions (4)

❖ Nested Functions

```
function calculateTotal(price,quantity) {
      var subtotal = price * quantity;
      return subtotal + calculateTax(subtotal);
      // this function is nested
      function calculateTax(subtotal) {
            var taxRate = 0.05;
            var tax = subtotal * taxRate;
            return tax;
      }
}
```


18

e. JS functions (5)

❖ Callback Functions

```
var calculateTotal = function (price, quantity, tax) {
    var subtotal = price * quantity;
    return subtotal + tax(subtotal);
};

var calcTax = function (subtotal) {
    var taxRate = 0.05;
    var tax = subtotal * taxRate;
    return tax;
};

var temp = calculateTotal(50,2,calcTax);
```

2 The local parameter variable tax is a reference to the calcTax() function

1 Passing the calcTax() function object as a parameter

We can say that calcTax variable here is a callback function

19

e. JS functions (6)

❖ Callback Functions

Passing an anonymous function definition as a callback function parameter

```
var temp = calculateTotal( 50, 2,
                function (subtotal) {
                    var taxRate = 0.05;
                    var tax = subtotal * taxRate;
                    return tax;
                }
);
```


20

f. JavaScript Output

- The document objects allows printing directly into the browser page (among other things)
- **window** object is implied
- Writing in text or HTML with script
  - No line-break

```
document.write("I am <B>BOLD</B>");
```

  - With line-break

```
document.writeln("I am <U>underlined</U>");
```


21

1.3. Methods of using JS

1. JS can reside in a separate page.
2. JS can be embedded in HTML documents in the <head> or in the <body>
   - Code in the <head> element is made available to be called later on in the document
   - Code in the <body> will be run when the document is parsed
3. JS object attributes can be placed in HTML element tags
   - E.g., **<body onLoad="alert('WELCOME')">**

22

Using Separate JS Files

- Linking can be advantageous if many pages use the same script.
- Use the source element to link to the script file.

```
<script src="myjavascript.js"
   language="JavaScript1.2"
   type="text/javascript">
</script>
```


23

Embedding JS in HTML

- When specifying a script only the tags <script> and </script> are essential, but complete specification is recommended:

```
<script language="javascript"
   type="text/javascript">
     <!--
     window.location="index.html"
        // End hiding script
     -->
</script>
```


24

Using Comment Tags

- HTML comment tags should bracket any script.
  - The tags hide scripts in HTML and prevent scripts from displaying in browsers that do not interpret JavaScript.
- JS comment
  - single-line comment
  - /* */: multiple-line comment

25

1.4. JS Objects

- JS: Not Object-Oriented, but Object-Based
- E.g.

```
function Person(myName, myAge){
  this.name = myName;
  this.age = myAge;
}
```

Java	JavaScript
Strongly-typed	Loosely-typed
Static	Dynamic
Classical	Prototypal
Classes	Functions
Constructors	Functions
Methods	Functions

```
var someGuy = new Person("Shawn", 28);
```


26

a. Accessing object properties

- Through '.'

```
var someGuy = new Person("Shawn", 28);
document.writeln('Name: ' +
someGuy.name);
```

- Objects and Associative Arrays are in fact two interfaces to the same data structure

→ Can access elements of someGuy like so: someGuy["age"] or someGuy["name"]

```
document.writeln('Name: ' +
someGuy["name"]);
```


27

b. Object functions

- Functions are just properties like any other property of an object (name, age, etc…)

```
function displayName() {
  document.writeln("I am " + this.name);
}
```

- To attach the function to Person, the constructor will become

```
function Person(myName, myAge) {
  this.name = myName;
  this.age = myAge;
  this.displayMe = displayName;
}
```


28

b. Object Functions (2)

- Then to call the function on the object

```
var someGuy = new Person("Shawn", 28);
someGuy.displayMe();
var someOtherGuy = new Person("Tim", 18);
someOtherGuy.displayMe();
```

- Declare the function inside the constructor:

```
function Person(myName, myAge) {
  this.name = myName;
  this.age = myAge;
  this.displayMe = function() {
     document.writeln("I am " + this.name);
  }
}
```


29

c. Object Literals

- Everything in JS is an Object
  - All literals are *object* literals.
- Those literals can be written:

```
<script type="text/javascript">
  var myNumber = new Number(123);
  var myString = new String('Bork!');
  var myBoolean = new Boolean(true);
  var myFunction = new Function('', "return 'hello'");
  var myRegExp = new RegExp('bork');
  var myArray = new Array();
  myArray[0] = 1; myArray[1] = 2; myArray[2] = 3;
  var myCarObject = new Object();
  myCarObject.color = 'red';
  myCarObject.tires = 4;
  myCarObject.windows = 6;
</script>
```


30

c. Object Literals (2)

- Object Literal Notation

```
var objName = {
  name1: value1,
  name2: value2,
  // ...
  nameN: valueN
};
```


31

Example 1

```
function myFunc(){  }
var myObject = new myFunc();
alert(typeof myObject);

function myFunc(){
 return 5;
}
var myObject = myFunc();
alert(typeof myObject);
```


32

Example 2

```
function myFunc(){
}

var myObject = new myFunc();
myObject.StringValue = "A String";
alert(myObject.StringValue);
var myObject2 = new myFunc();
alert(myObject2.StringValue);
```


33

Example 3

```
function myFunc(){
   this.StringValue = "This is a
  String";
}

var myObject = new myFunc();
myObject.StringValue = "myObject
  string";
var myObject2 = new myFunc();
alert(myObject.StringValue);
alert(myObject2.StringValue);
```


34

d. Inheritance in JS

- No built-in inheritance
  - Through Function
  - Through prototyping

35

Inheritance through functions

```
function superClass() {
   this.bye = superBye; this.hello = superHello;
}
function subClass() {
   this.abc = superClass;
   this.abc();
   this.bye = subBye;
}
function superHello() { return "Hello superClass";}
function superBye() { return "Bye superClass"; }
function subBye() { return "Bye subClass"; }

var newClass = new subClass();
alert(newClass.bye());
alert(newClass.hello());
```


36

Prototype object

```
function Person(firstName, lastName){
   this.firstName = firstName;
   this.lastName = lastName;
}



Person.prototype.showFullName = function() {
   console.log(this.firstName + ' ' + this.lastName);
}

var justin = new Person('Justin', 'Vo');
console.log(justin); // Person {firstName: "Justin", lastName: "Vo"}
justin.showFullName(); // Justin Vo
```


37

Inheritance through prototyping

- Prototype inheritance instead of class-based inheritance
- Object.prototype ~ super class
  - E.g. Complex object inherits properties from Complex.prototype and from Object.prototype
- Syntax
  - subClass.prototype = new superClass;

38

Prototyping Example

```
function superClass() {
   this.bye = superBye;
   this.hello = superHello;
}
function subClass() { this.bye = subBye; }
subClass.prototype = new superClass;
function superHello() {return "Hello superClass"; }
function superBye() { return "Bye superClass"; }
function subBye() { return "Bye from subClass"; }

var newClass = new subClass();
alert(newClass.bye());
alert(newClass.hello());
```


39

```
var person = {
  firstName: 'Hoang',
  lastName: 'Pham',
  showName: function() {
    console.log(this.firstName + ' ' + this.lastName);
  }
}; // object literal này có prototype là Object.prototype

function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.showName = function() {
     console.log(this.firstName + ' ' + this.lastName);
  };
}

var otherPerson = new Person('Hoang', 'Pham');
// object này có prototype là Person.prototype
// Person.prototype kế thừa Object.prototype
```


40

41

```
function Person() {
  this.firstName = 'Per';
  this.lastName = 'son';
  this.sayName = function() { return firstName + ' ' + lastName };
}

// Viết một Constructor Function khác
function SuperMan(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

// Ta muốn SuperMan sẽ kế thừa các thuộc tính của Person
// Sử dụng prototype để kế thừa
SuperMan.prototype = new Person();

// Tạo một object mới bằng Constructor Function
var sm = new SuperMan('Hoang', 'Pham');
sm.sayName(); // Hoang Pham. Hàm này kế thừa từ prototype của Person
```


42

43

e. Built-in Objects

- Number, Boolean, String
  - Primitive types are automatically coerced into Objects when assigned to variables.
    - `var str = "abc";`
    - str is a String object
  - Number and Boolean are boring!
  - String has some helpful properties/functions:
    - length
    - toUpperCase
    - substring
    - link
- Date
  - No properties, but contains a bunch of methods for getting, setting and manipulating dates.
- Math
  - Calculate PI, find the SIN or COS of an angle, etc..

44

1.5. JS Arrays

- Creating Arrays

```
var a = new Array(); // empty array
var b = new Array("dog", 3, 8.4);
var c = new Array(10); // array of size 10
var d = [2, 5, 'a', 'b'];
```

- Assigning values to Arrays

```
c[15] = "hello";
c.push("hello");
```

- Associative Arrays

```
var e = new Array();
e["key"] = "value";
e["numKey"] = 123;
```


45

1.5. Arrays (2)

- Properties and Methods
  - length
  - join()
  - reverse()
  - sort()
  - concat()
  - slice()
  - splice()
  - push() / pop()
  - shift() / unshift()
  - toString()
  - .....

46

Content

47

2.1. The `window` object

- We have already seen the 'document' object – how we print to screen
- 'window' object – JavaScript representation of a browser window
- Built-in properties
  - **`closed`**: A boolean value that indicates whether the window is closed.
  - **`defaultStatus`**: Default message that is loaded into the status bar when the window loads.

E.g. `window.defaultStatus = "A status bar";`

48

2.1. The `window` object (2)

- Built-in functions
  - `alert("message")`
  - `window.close()`
  - `confirm("message")`
  - `focus()`
  - `open("URLname","Windowname",["options"])`
    - `options: height, weight, alwaysRaised, location, menubar, etc..`
    - E.g. `open(http://google.com,"MyGoogle", "toolbar=no,alwaysRaised=yes");`

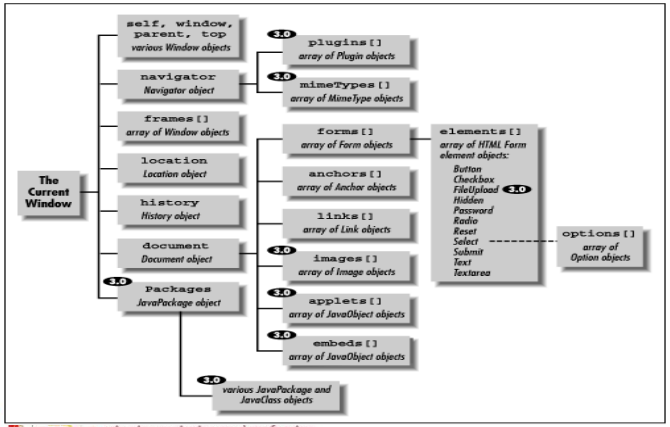
2.1. The `window` object (3)

- Built-in objects
  - `window.location`
    - **`href`** represents a complete URL.
    - **`hostname`** represents the concatenation *host:port*
    - `window.location.href="http://google.com";`
  - `window.history`
    - **`length`** reflects the number of entries in the history list
    - `history.go(-1)`
    - `history.back()`

2.2. The form objects

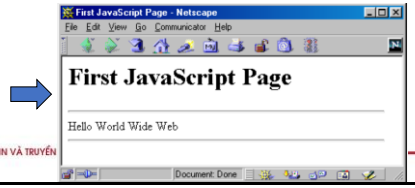
- Form objects can be accessed by:
  `window.document.formName` OR
  `window.document.forms[0]`
- Properties
  - `action, target, length, method, etc…`
- Functions
  - `window.document.formName.submit();`
  - `window.document.formName.reset();`
- Accessing Form Field Values
  - `window.document.formName.firstname.value`

(D)HTML Object Hierarchy



A Simple Script

```
<HTML>
   <HEAD>
      <TITLE>First JavaScript Page</TITLE>
   </HEAD>
   <BODY>
      <H1>First JavaScript Page</H1>
      <SCRIPT TYPE="text/javascript">
       <!--
          document.write("<HR>");
          document.write("Hello World Wide Web");
          document.write("<HR>");
       // -->
       </SCRIPT>
</BODY>
</HTML>
```



53

Extracting Document Info with JavaScript, Example

```
<HTML>
<HEAD>
   <TITLE>Extracting Doc Info with JavaScript</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Extracting Document Info with JavaScript</H1>
<HR>
<SCRIPT TYPE="text/javascript">
<!--
function referringPage() {
   if (document.referrer.length == 0)
      return("<I>none</I>");
   else return(document.referrer);
}
```


54

E.g. cont.

```
document.writeln
("Document Info:\n" + "<UL>\n" +
" <LI><B>URL:</B> " + document.location + "\n" +
" <LI><B>Modification Date:</B> " + "\n" +
document.lastModified + "\n" +
" <LI><B>Title:</B> " + document.title + "\n" +
" <LI><B>Referring page:</B> " + referringPage() + "\n"
+ "</UL>");
document.writeln("Browser Info:" + "\n" +
"<UL>" + "\n" +
" <LI><B>Name:</B> " + navigator.appName + "\n" +
" <LI><B>Version:</B> " + navigator.appVersion + "\n" +
"</UL>");
// -->
</SCRIPT>
<HR>
</BODY>
</HTML>
```


55

Extracting Document Info with JavaScript, Result 1

Extracting Document Info with JavaScript

Document Info:

- **URL:** http://localhost/extract_doc.html
- **Modification Date:** 11/02/2009 09:47:45
- **Title:** Extracting Doc Info with JavaScript
- **Referring page:** *none*

Browser Info:

- **Name:** Microsoft Internet Explorer
- **Version:** 4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; GTB6; SLCC2; .NET CLR 2.0.50727; .NET CLR 3

56

Extracting Document Info with JavaScript, Result 2

Referrer page for extract_doc example

Click here to go to extract_doc

Extracting Document Info with JavaScript

Document Info:

- **URL:** http://localhost/extract_doc.html
- **Modification Date:** 11/02/2009 09:47:45
- **Title:** Extracting Doc Info with JavaScript
- **Referring page:** http://localhost/referrer_example.html

Browser Info:

- **Name:** Netscape
- **Version:** 5.0 (Windows; en-US)

57

2.3. Event Handlers

- *Events are actions that occur usually as a* result of something the user does.
  - E.g. Clicking a button is an event, as is changing a text field or moving the mouse over a hyperlink.
- Eg: click, change, focus, load, mouseover, mouseout, reset, submit, select

58

2.3. Event Handlers (2)

- Use Various onXxx Attributes
  - onClick
  - onLoad
  - onMouseOver
  - onFocus
  - etc.

HTML document using the inline hooks

```
...
<script type="text/javascript" src="inline.js"></script>
...
<form name='mainForm' onsubmit="validate(this);">
    <input name="name" type="text"
        onchange="check(this);"
        onfocus="highlight(this, true);"
        onblur="highlight(this, false);">
    <input name="email" type="text"
        onchange="check(this);"
        onfocus="highlight(this, true);"
        onblur="highlight(this, false);">
    <input type="submit"
        onclick="function (e) {
                    ...
                }">
...
```

Notice that you can define an entire event handling function within the markup. This is NOT recommended!

inline.js

```
function validate(node) {
  ...
}
function check(node) {
  ...
}
function highlight(node) {
  ...
}
```


59

2.3. Event Handlers (3)

- You can use *event handlers, such as* **onChange and onClick, to make your** script react to events.

```
<input type="button" onClick="javascript:doButton()>
<select onChange="javascript:doChange()">
<a onClick="javascript:doSomething()"> </a>
<form onSubmit="javascript:validate()">
<body onLoad="javascript:init()">
```

- Event Listener Approach

```
var myButton = document.getElementById('example');
myButton.addEventListener('click', alert('some message'));
myButton.addEventListener('mouseout', funcName);
```


60

User Events, Example

```
<HTML>
<HEAD>
   <TITLE>Simple JavaScript Button</TITLE>
<SCRIPT TYPE="text/javascript">
<!--
function dontClick() {
   alert("I told you not to click!");
}
// -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="WHITE">
<H1>Simple JavaScript Button</H1>
<FORM>
<INPUT TYPE="BUTTON" VALUE="Don't Click Me"
                        onClick="dontClick()">
</FORM>
</BODY>
</HTML>
```


61

User Events, Result

62

Events

Mouse

- Click
- Dblclick
- Mousedown
- Mouseup
- Mouseover
- Mousemove
- Mouseout

Frame/Object

- Load
- Unload
- Abort
- Error
- Resize
- Scroll

Form

- Select
- Change
- Submit
- Reset
- Focus
- Blur

Keyboard

- Keypress
- Keydown
- Keyup

63

Content

64

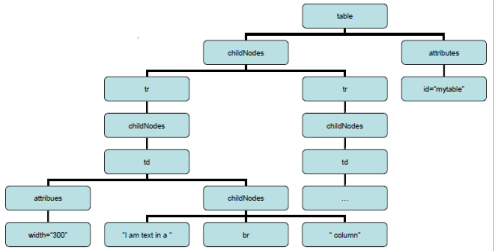
3.1. DOM (Document Object Model)

- W3C DOM, "The DOM"
  - Method of accessing / modifying XML information on the page
- Tree structure of all HTML elements, including attributes and text
- Contents can be modified or deleted
- New elements can be created and inserted into the DOM Tree

65

DOM Representation of HTML

```
<table id="mytable">
 <tr>
   <td width="300">
     I am text in a <BR> column
   </td>
 </tr>
 <tr>
   <td> … </td>
 </tr>
<table>
```

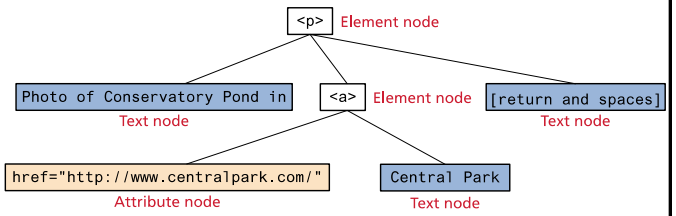


66

DOM Representation of HTML

- Nodes and NodeLists

```
<p>Photo of Conservatory Pond in
   <a href="http://www.centralpark.com/">Central Park</a>
</p>
```



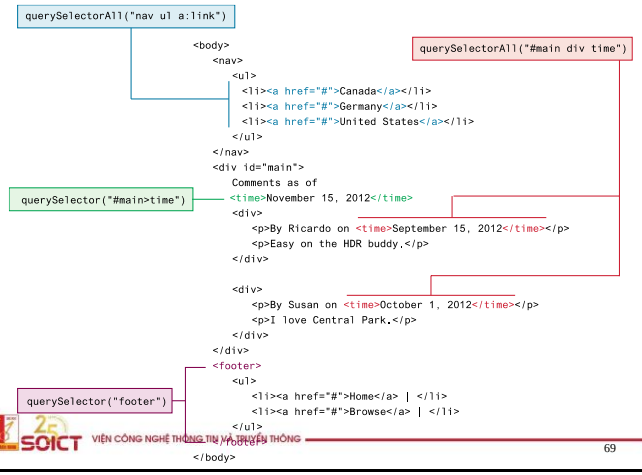
67

The `document` object

- The document object is the JavaScript interface to the DOM of any HTML page.
- Accessing elements:
  - By name
    `document.getElementsByTagName('td')[indexOfCol]`
  - By ID
    `document.getElementById('id')`
  - Walk the DOM Tree
    `document.childNodes[0].childNodes[1].childNodes[4]`
- Newer
  - querySelector() and
  - querySelectorAll()

68

The document object



69

DOM Element Attributes

DOM Attributes

- nodeName
- nodeValue
- nodeType
- parentNode
- childNodes
- firstChild
- lastChild
- previousSibling
- nextSibling
- attributes
- ownerDocument

Node Types

- 1 = an HTML element
- 2 = an element attribute
- 3 = text
- 8 = an HTML comment
- 9 = a document
- 10 = a document type definition

Here's a good article that uses these.

70

Manipulating the DOM

- Dynamically creating and adding elements
  - document.createElement
  - appendChild
- E.g.

```
function addDiv    (){
      var myElement = document.createElement('<div
    style="width:600; height:200;background-
   color:blue;">www.java2s.com</div>');
      document.body.appendChild(myElement);
}
```


71

Manipulating the DOM

- Dynamically creating and adding elements

1. Create a new text node

```
var text = document.createTextNode("this is dynamic");
```

"this is dynamic"

2. Create a new empty element

```
var p = document.createElement("p");
```

3. Add the text node to new element

```
p.appendChild(text);
```

 "this is dynamic"

4. Add the element to the

```
var first = document.getElementById("first");
first.appendChild(p);
```


72

```
<html>
<head>
  <title>Example Message Box Page</title>
  <script type="text/javascript">
    function doLoad()
    {
      document.getElementById('sweet-link').
      addEventListener('click', confirmClick, false);
    }//end doLoad

    function confirmClick()
    {
      return confirm('Are you sure to go to that link?');
    }//end confirmClick

    window.addEventListener('load', doLoad, false);
  </script>
</head>
<body>
  <a id="sweet-link" href="http://www.hut.edu.vn">HUT</a>
</body>
</html>
```


73

Manipulating the DOM

- Changing an Element's Style
- Changing an Element's Content

```
document.getElementById("here").innerHTML =
"foo<em>bar</em>";
```


74

3.2. Cookies

- Variables set by a webpage and stored on a user's computer
- Cookies expire (deleted from the user's computer) after a certain amount of time
- Mostly used to store user preferences, but can be used for other purposes as well
  - Can you think of one?
- Get the Cookie object in JavaScript.
  - **`window.document.cookie`** acts like a String with some unique properties

75

a. Writing cookies

- Cookies are created or modified by writing **`document.cookie = cookieString;`**
  - cookieString is a ';' delimited list of name=value pairs of all the properties of a cookie
  - Best way to set an expiry date is to use the JavaScript Date object to get a GMT date [toGMTString()].
    GMT Example: Thu, 31-Dec-1998 00:00:00 GMT
  - Alternatively you can set it (as above) using milliseconds from the current time.
  - E.g. **`document.cookie="numVisit=0;expires=1000"`**
- Append multiple cookies

```
document.cookie="numVisit=0;expires=10000"
document.cookie="name=Shawn;expires=10000"
```


76

b. Reading Cookies

- Browser Sends…
  Cookie: *name1=value1; name2=value2 …*
- The only part of the cookie that is visible when parsing/printing **`document.cookie`** is the name=value pair. All other attributes (expiry, etc…) are removed when sending a cookie.
- What happens when there are multiple cookies available?

- **`document.writeln(document.cookie)`** would give … **"`numVisit=0;name=Shawn`"**

77

Debugging Tools

- Mozila Firefox interactive JavaScript Console.
  - Shows all errors/warnings during run-time
- Mozilla Firefox DOM inspector
  - Shows a tree structure of the current document
- Mozilla Firefox Web Developer 0.9.3 (extension)
  - Display Form Details, View JavaScript Code, View Cookie Information
- Mozilla Firefox JavaScript Debugger 0.9.84 (extension)
  - Interactive walk-through of JavaScript code

79

Content

80

What is Ajax?

81

What is Ajax?

82

What is Ajax?

- Asynchronous
- JavaScript
- And
- XmlHttpRequest (XHR) or XML?
  - XHR based on DOM, CSS, XHTML, support across all browsers

83

What is Ajax?

- Asynchronous
  - Bits of data downloaded when needed
  - Download is initiated, but may not complete for a while
  - Meanwhile the user can continue working
- Javascript
  - The language used in Web Browsers to manipulate what appears
- With XML
  - The format of the data downloaded the JavaScript to modify what appears on the page

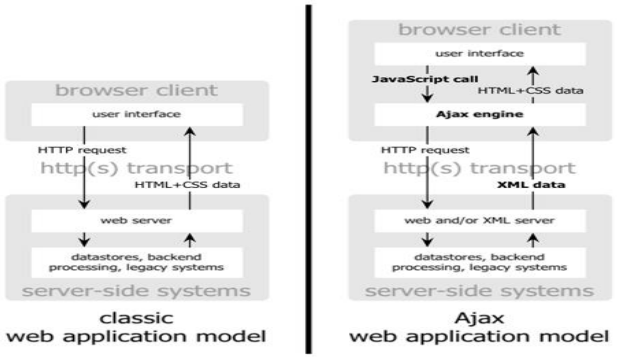
84

What is AJAX?

- Allows incremental update of Web pages.
- Built using standard web technologies
  - HTTP, (X)HTML, CSS, JavaScript, DOM, XML
- Examples:
  - Google Suggests (2005)
  - Google & Yahoo! Maps
  - Amazon A9 Search
  - Flickr, BaseCamp, Kayak
  - Yahoo! AJAX Library

85

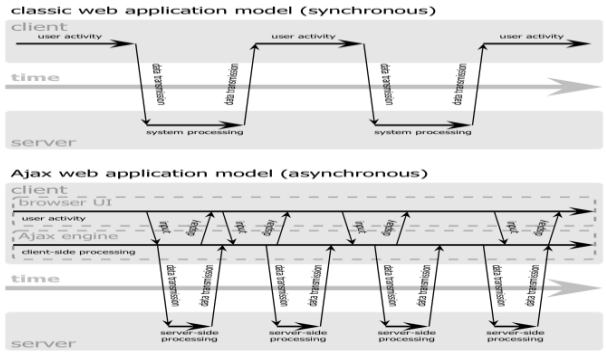
Why Ajax?



Source: Garrett(2005)

86

Why Ajax?



Source: Garrett(2005)

87

AJAX permits Rich Internet Applications (RIA)

- Applications that look and feel like desktop apps
  - In whole or in part
- A key part of "Web 2.0"

88

Suggested reference books for AJAX

- Visual Quickstart Guide: JavaScript and Ajax, 6th edition
  - By Tom Negrino and Dori Smith, 2007, Peachpit Press
  - Website: http://www.javascriptworld.com/
  - Covers css and everything you need to know
- Ajax Hacks: Tips and Tools for Creating Responsive Web Sites
  - Bruce W. Perry, 2006 O'Reilly
  - Also covers basics of Ruby on Rails

89

AJAX Alternatives

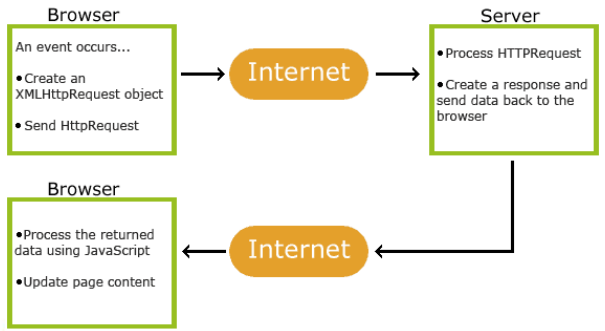
- Macromedia Flash
  - Requires a plug-in
    - So what? It comes already with almost every browser
- Java Web Start/Applets
- .NET – No Touch Deployment
  - Both need a runtime preinstalled
- Handheld device browsers generally do not support the full range of Ajax technologies.

90

Content

91

How AJAX works?



92

AJAX Implementation

- To implement AJAX we need to answer three questions:
  - What triggers the AJAX request?
    - Usually a JavaScript event (onBlur, onClick, etc.)
  - What is the server process that handles the AJAX request and issues the response?
    - Some kind of URL (use a Service Locator)
  - What processes the response from the server(what is the callback method)?
    - A JavaScript function that gets the response and manipulates the DOM, based on the text returned

93

XmlHttpRequest Object (XHR)

- The Heart of AJAX
- First implemented in IE in 1997 as part of the new DHTML standard
- Response comes in one of two properties:
  - responseXML – Returns a DOM document (can use functions such as, `getElementById()`)
  - responseText – A text string (can be HTML, or even JavaScript code)

94

Example: Step 1 - HTML code

```
<body>
 <form name="testForm">
     Input text: <input type="text"
          onkeyup="doWork();"
          name="inputText" id="inputText" />
     Output text: <input type="text"
          name="outputText" id="outputText" />
 </form>
</body>
```

- called in every case when a key is up (pressed)
- how we can send messages to the server script?

95

Example: Step 2 - Create XHR object

```
// Get the HTTP Object
function getHTTPObject(){
   if (window.ActiveXObject) {
       return new ActiveXObject("Microsoft.XMLHTTP");
   }
   else if (window.XMLHttpRequest) {
       return new XMLHttpRequest();
   }
   else {
       alert("Your browser does not support AJAX.");
       return null;
   }
}
```


96

Example: Step 3 - Implement business logic in client

```
function doWork(){
   httpObject = getHTTPObject();
   if (httpObject != null) {
      httpObject.open("GET",
      "upperCase.php?inputText="
            +document.getElementById(
                  'inputText').value, true);
      httpObject.send(null);
   httpObject.onreadystatechange=setOutput;
   }
}
```

How we can catch the response from the server?
- assign a function to a special property of the XHR object
- this function will be called if the state of the object was changed

97

Example: Step 4 - Change the value of the outputText field

readyState
0 = uninitialized
1 = loading
2 = loaded
3 = interactive
4 = complete

```
function setOutput(){
  if(httpObject.readyState == 4){
  document.getElementById('outputText').value
                = httpObject.responseText;
  }
}
```


98

Handling the Response

- Response can be one of the following:
  - Formatted data (XML, other custom format)
    - `XMLHttpRequest.responseXML`
    - Decouples the server from presentation issues
    - Could perform XSLT transformation on returned XML
  - HTML
    - `XMLHttpRequest.responseText`
    - Server generates HTML, script "injects" HTML via `innerHTML`
    - Server is now concerned with presentation
  - JavaScript
    - `XMLHttpRequest.responseText`
    - Use the `eval()` JavaScript command
    - Again, our server code is concerned with presentation

99

Example: Step 5 – Implement business logic in server (PHP)

```
<?php
  if (isset($_GET['inputText']))
  echo strtoupper($_GET['inputText']);
?>
```

- You can do anything you want here to do the required business logic
  - Do more complicated thing
  - Get something from database
  - …

100

AJAX Concerns

- Security
- Browser Compatibility
- Accessibility
- The Back Button
- What if JavaScript is Turned Off?

101

Example: AJAX Concerns

- User does not know updates will occur
- User does not notice an update
- User cannot find the updated information
- Unexpected changes in focus
- Loss of Back button functionality*
- URIs cannot be bookmarked*

102

AJAX packages

- Yahoo User Interface Library
  - http:developer.yahoo.com/yui/
  - sourceforge.net/projects/yui/
- Script.aculo.us
  - http://Script.aculo.us
- Google web toolkit
  - Write code in Java that is compiled to Javascript
  - http://code.google.com/webtoolkit/
- Dojo
  - http://www.dojotoolkit.org/
- jQuery
  - http://jquery.com/

103

AJAX packages (2)

- Moo.fx
  - http://moofx.mad4milk.net/
- Open Rico
  - http://openrico.org/
- Tibco General Interface
  - http://www.tibco.com/devnet/gi/default.jsp

104

Question?

105