Internet Technologies
5-Dynamic Web

F. Ricci
2010/2011
Content

- The "meanings" of dynamic
- Building dynamic content with Java EE (server side)
- HTML forms: how to send to the server the input
- PHP: a simpler language for building dynamic pages
- Servlets and JavaServer Pages
- Client side dynamic content with JavaScript
- JavaScript language
- DOM: Document Object Model
Dynamic Pages

- **Meaning 1: Server Side**
  - A page is dynamic because it is created differently every time you call it
  - Example: the page listing the students enrolled to Internet Technologies exam

- **Meaning 2: Client Side**
  - A page is dynamic because some code is executed in the browser in response to user input
  - Example: a page on an eCommerce web site that when you click on the "confirm" button it alerts that you are spending 379Euro on an Ipod Touch 32G.
Why Build Web Pages Dynamically?

- The page is **based** on **data submitted** by the **user**
  - E.g., results page from search engines and order-confirmation pages at on-line stores

- The page is **derived** from **data** that **changes frequently**
  - E.g., a weather report or news headlines page

- The page **uses** information from **databases** or other server-side sources (Content Management System)
  - E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale.
- Distributed multitiered application model for enterprise applications
- Application logic is divided into components according to function.
The client communicates with the business tier running on the Java EE server:
- either directly
- or, as in the case of a client running in a browser, by going through JSP pages or servlets running in the web tier.
Web Tier

- Java EE **web components** are either servlets or pages created using JSP technology (JSP pages)
  - **Servlets** are Java programming language classes that dynamically process requests and construct responses
  - **JSP pages** are text-based documents that execute as servlets but allow a more natural approach to creating static content
- The web tier might include a **JavaBeans** component to manage the user input and send that input to enterprise beans running in the business tier for processing
- **Static HTML pages** and **applets** are bundled with web components during application assembly but are not considered **web components** by the Java EE specification (are client components).
Business components and EIS Tiers

- **Business code**, is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in the business tier
  - *We will implement it with servlets (not with enterprise java beans)*

- The **enterprise information system** tier handles EIS software and includes:
  - enterprise infrastructure systems such as enterprise resource planning (ERP),
  - mainframe transaction processing,
  - database systems,
  - and other legacy information systems.
HTML Forms

- Form elements allow the user to **enter information** (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) and **send** this information to a server.

- Example:

```html
<form>
  First name: <input type="text" name="firstname"> <br>
  Last name: <input type="text" name="lastname">
</form>
```

- How it looks in the browser:

```
First name: [input]
Last name: [input]
```

http://www.w3schools.com/html/html_forms.asp
**HTML Forms**

```html
<form name="input" action="html_form_action.asp">
First name: <input name="firstname" type="text"> <br>
Last name: <input name="lastname" type="text">  <br>
<input value="Submit" type="submit">
</form>
```

- **action**: Specifies **URL of server component** that gets data
- `<input name="a-name"...>`: assign a name to the input data
- When user clicks “submit”, the parameters are sent to server
  - URL?name1=value1&name2=value2
Example

- We have a mini web server that simply echoes what is sent by the client
- If the action in the previous form is
  - `<FORM ACTION="http://localhost:8088/somevalue">
- Then the output is

```
EchoServer Results

Here is the request line and request headers sent by your browser:

GET /somevalue?FirstName=francesco&LastName=ricci HTTP/1.1
Host: localhost:8088
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.9) Gecko/20070125 Firefox/2.0.0.9
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```
Parameters with GET and POST

- **GET:** parameters are added at the end of the URL in encoded form.
  - If your URL is the following:
    - `http://localhost:8088/somevalue`
  - The parameter “FirstName” added to the URL:
    - `http://localhost:8088/somevalue?FirstName=Francesco`
  - Additional parameters can be added, separated by &:
    - `http://localhost:8088/somevalue?FirstName=Francesco&LastName=Ricci`

- **POST:** parameters are passed as the body of request with the same type of encoding.
- If you have lots of parameters or binary data, you may use the POST request.
Example using POST

```html
<form action="http://localhost:8088/somevalue" method="post">
First name: <input name="FirstName" value="" type="text"><br>
Last name: <input name="LastName" value="" type="text">
<p> <input type="submit"> </p>
</form>
```

---

**EchoServer Results**

Here is the request line and request headers sent by your browser:

```
POST /somevalue HTTP/1.1
Host: localhost:8088
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.9) Gecko/20070325 Firefox/2.0.0.9
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
```

First-Name=Lorenzo&Last-Name=Teleca
Get vs. Post

- **GET**
  - Attr/Val pairs attached after '?'
  - The url can be bookmarked and the action simply re-executed
  - If user refreshes, or clicks back button there is a Double Submit!
  - Use only for **idempotent** operations (e.g. \( f*f=f \))
    - E.g. *not use it if the server will charge you the cost of the items in your cart!*
  - You can type the data in by hand in the url requested (e.g. for testing purposes).
Get vs. Post

POST
- Attr/Val pairs attached as request body
- The url does not correspond to an operation and cannot be bookmarked
- If user refreshes, or clicks back button, browser may display warning
- Can use for non-idempotent operations
- Or idempotent ops with LONG URLs (browsers may limit the URL to few thousand characters)
- Useful for sending binary data
- Keep data private from someone looking over the user's shoulder (*but are still in the request body*).
Forms and Servlet example

- If I enter my name and submit I get the following result

```html
<html>
<body bgcolor="white">
<head>
<title>Request Parameters Example</title>
</head>

<h3>Request Parameters Example</h3>
Parameters in this request:
No Parameters, Please enter some

<form>
  First Name: 
  Last Name: 
  <input type="submit" value="Submit Query">
</form>

First Name: Francesco
Last Name: Ricci
```

form
Input Tag Types

- **Radio Buttons** are used when you want the user to select one of a limited number of choices

  ```html
  <form>
  <input type="radio" name="sex" value="male"> Male <br>
  <input type="radio" name="sex" value="female"> Female
  </form>
  ``

- **Checkboxes** are used when you want the user to select one or more options of a limited number of choices

  ```html
  <form>
  I have a bike: <input type="checkbox" name="vehicle" value="Bike" /> <br />
  I have a car: <input type="checkbox" name="vehicle" value="Car" /> <br />
  I have an airplane: <input type="checkbox" name="vehicle" value="Airplane" />
  </form>
  ```
Dynamic Web Documents

- Steps in processing the information from an HTML form
- Example shows a CGI but other applications could be executed using custom connectors of the web server

1. User fills in form
2. Form sent back
3. Handed to CGI
4. CGI queries DB
5. Record found
6. CGI builds page
7. Page returned
8. Page displayed
CGI – Common Gateway Interface

- Invented in 1993 by NCSA for HTTPd web server
  - Client requests program to be run on server-side
  - Web servers often have a cgi-bin/ directory to hold executable files called with CGI
  - Web server passes parameters to program through UNIX shell environment variables
  - Program spawned as separate process via fork
  - Program's output => Results
  - Server passes back results (usually in form of HTML)

- Good for interfacing external applications with information servers

- **Slow**: every time you call the cgi a new process must be started!

Who writes the headers in the response?
A sample HTML page with embedded PHP (Personal Home Page Tools)

```html
<html>
<body>
<h2>This is what I know about you</h2>
<?php echo $HTTP_USER_AGENT; ?>
</body>
</html>
```

PHP commands are tagged with `<?php ... ?>`
Dynamic Web Documents (3)

(a) Web page containing a form

(b) A PHP script for handling the output of the form (action.php)

(c) Output from the PHP script when the inputs are "Barbara" and 24 respectively

<static html>
<html>
<body>
<form action="action.php" method="post">
<p>Please enter your name: <input type="text" name="name"></p>
<p>Please enter your age: <input type="text" name="age"></p>
<input type="submit">
</form>
</body>
</html>

<static html>
<html>
<body>
<h1>Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>.
</body>
</html>

<static html>
<html>
<body>
<h1>Reply: </h1>
Hello Barbara.
Prediction: next year you will be 25
</body>
</html>

<dynamic html>
<html>
<body>
<h1>Reply: </h1>
Hello "Barbara".
Prediction: next year you will be 25
</body>
</html>

(a) Actually you should use $_POST["name"]

(b)

(c)
Servlet Roles

- Read the explicit data sent by the client
- Read the implicit HTTP request data sent by the browser
- Generate the results
- Send the explicit data (i.e., the document) to the client
- Send the implicit HTTP response data.
Servlet Architecture

HTTP Request

HTTP Response

Servlet Container

Java Virtual Machine (JVM)

Servlet 1

Servlet 2

Servlet n

Client Web

Web Server
What is a servlet

- Java Servlets/JSP are part of the Sun’s J2EE Enterprise Architecture
  - The web development part

- Java Servlet (http://www.oracle.com/technetwork/java/javaee/servlet/index.html)
  - is a simple, consistent mechanism for extending the functionality of a web server
  - Are precompiled Java programs that are executed on the server side
  - Require a Servlet container to run in

- Latest Servlet Specification is 3.0
Servlet/Container Engine

- Servlets/JSP require a **Container**
- Apache Tomcat is the **reference implementation** of the Servlet/JSP Specs
- It is open source, small, install quickly, and is FREE
- Latest Stable Version is 7.0.x implementing Servlet 3.0 and JSP 2.2 specifications.
- Web Site: [http://tomcat.apache.org](http://tomcat.apache.org)
- It include a simple HTTP 1.1 server, good enough for development and small intranets
- Tomcat is included in industrial application servers (e.g. **JBoss**) and in your IDE (**NetBeans**)
The Advantages of Servlets Over CGI

- **Efficient**: Threads instead of OS processes - one servlet object and each call in a separate thread
- **Convenient**: Lots of high-level utilities
- **Powerful**: Sharing data, pooling, persistence
- **Portable**: Run on virtually all operating systems and servers
- **Inexpensive**: There are plenty of free and low-cost servers
- **Secure**: No shell escapes, no buffer overflows
- **Mainstream**: See next page
Mainstream

- Popular:
  - The single most common use of Java technology
  - The leading technology for medium/large Web applications

- Supported by:
  - Apache, Oracle, IBM, Sybase, BEA, Macromedia, Caucho, Sun/iPlanet, New Atlanta, ATG, Fujitsu, Lutris, Silverstream, the World Wide Web Consortium (W3C), and many others
  - Plugins for IIS and Zeus

- Runs on:
  - Windows, Unix/Linux, MacOS, VMS, and IBM mainframe OSs

- Used for:
  - Airline companies, hotels, e-commerce sites, search engines, banks, financial sites, etc., etc., etc.
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Very simplistic servlet that generates plain text.
 * <P>
 * Taken from More Servlets and JavaServer Pages
 * from Prentice Hall and Sun Microsystems Press,
 * © 2002 Marty Hall; may be freely used or adapted.
 */

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
Here's the outline of a basic servlet that handles GET requests:

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SomeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                 throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser
    }
}
```
package moreservlets;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet2 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType = "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
        out.println(docType + "<HTML>
" + "<HEAD><TITLE>Hello</TITLE></HEAD>
" + "<BODY BGCOLOR="#FDF5E6">
" + "<H1>Hello</H1>
" + "</BODY></HTML>");
    }
}
JavaServer Pages (JSP)

- Use regular HTML for most of page
- Mark dynamic content with special tags
- Details in second half of course

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Welcome to Our Store</title></head>
<body>
<h1>Welcome to Our Store</h1>
<small>Welcome,
<!-- User name is "New User" for first-time visitors -->
<%= coreservlets.Utils.getUserNameFromCookie(request) %>
To access your account settings, click
<a href="Account-Settings.html">here.</a></small>
<p>
Regular HTML for rest of on-line store’s Web page
</p>
</body></html>
```
Client-Side Dynamic Web Page Generation

- (a) **Server-side** scripting with PHP or JSP
- (b) **Client-side** scripting with JavaScript
JavaScript was designed to add **interactivity** to HTML pages.

JavaScript is a **scripting** language.

A scripting language is a **lightweight programming** language.

A JavaScript consists of lines of executable computer code.

A JavaScript is usually **embedded** directly into HTML pages.

JavaScript is an **interpreted language** (means that scripts execute without preliminary compilation).

Java and JavaScript are two completely different languages in both concept and design.
What JavaScript can do

- JavaScript can put dynamic text into an HTML page - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page.

- JavaScript can react to events - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.

- JavaScript can be used to validate data - A JavaScript can be used to validate form data before it is submitted to a server.

- JavaScript can be used to detect the visitor's browser and - depending on the browser - load another page specifically designed for that browser.

- JavaScript can be used to create or modify cookies - to store and retrieve information on the visitor's computer.
Example

```html
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

- Produce a page with the string "Hello World!"
- `document.write` is a standard JavaScript method for writing output to a page
- Semicolons are **optional**!
- However, semicolons are required if you want to put more than one statement on a single line.
Scripts in the head or body

- JavaScripts in the **body** section (as in previous example) will be executed **while** the page **loads**
- JavaScripts in the **head** section will be executed when **called**

```html
<html>
<head>
    <script type="text/javascript">
        function message()
        {
            alert("This alert box was called with the onload event")
        }
    </script>
</head>
<body onload="message()">
</body>
</html>
```
You can create a variable with the `var` statement:
```javascript
var strname = some_value
```

You can also create a variable without the `var` statement:
```javascript
strname = some_value
```

You can assign a value to a variable like this:
```javascript
var strname = "Hege"
strname = "Hege"
```

When you declare a variable **within a function**, the variable can only be accessed within that function.

If you declare a variable **outside a function**, all the functions on your page can access it.
## Operators

**Arithmetic operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| +        | Addition                  | x=2
          | y=2
          | x+y     | 4      |
| -        | Subtraction               | x=5
          | y=2     | x-y     | 3      |
| *        | Multiplication            | x=5
          | y=4     | x*y     | 20     |
| /        | Division                  | 15/5
          | 5/2     |         | 3      |
| %        | Modulus (division remainder) | 5%2
          | 10%8    | 10%2   | 1      |
| ++       | Increment                 | x=5
          | x++     |         | x=6    |
| --       | Decrement                 | x=5
          | x--     |         | x=4    |

**Assignment operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Is The Same As</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>x=y</td>
<td>x=y</td>
</tr>
<tr>
<td>+=</td>
<td>x+=y</td>
<td>x=x+y</td>
</tr>
<tr>
<td>-=</td>
<td>x-=y</td>
<td>x=x-y</td>
</tr>
<tr>
<td>*=</td>
<td>x*=y</td>
<td>x=x*y</td>
</tr>
<tr>
<td>/=</td>
<td>x/=y</td>
<td>x=x/y</td>
</tr>
<tr>
<td>%=</td>
<td>x%=y</td>
<td>x=x%y</td>
</tr>
</tbody>
</table>
Comparison Operators

Assume that \( x = 5 \);

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>is equal to</td>
<td>( x == 8 ) is false</td>
</tr>
<tr>
<td>===</td>
<td>is exactly equal to (value and type)</td>
<td>( x == 5 ) is true ( x === &quot;5&quot; ) is false</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal</td>
<td>( x != 8 ) is true</td>
</tr>
<tr>
<td>&gt;</td>
<td>is greater than</td>
<td>( x &gt; 8 ) is false</td>
</tr>
<tr>
<td>&lt;</td>
<td>is less than</td>
<td>( x &lt; 8 ) is true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>is greater than or equal to</td>
<td>( x &gt;= 8 ) is false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>is less than or equal to</td>
<td>( x &lt;= 8 ) is true</td>
</tr>
</tbody>
</table>
If, else

<script type="text/javascript">
var d=new Date()
var time=d.getHours()
if (time<13)
    { document.write("<b>Good morning</b>") }
</script>

- **Jscript2** a "Good morning" greeting if the time is less than 13
- **General structure**
  if (condition1)
  { code to be executed if condition1 is true }
  else if (condition2)
  { code to be executed if condition2 is true }
  
- **Jscrip3**
Functions

- A function contains code that will be executed by an event or by a call to that function.
- If you want the function to return a value:

```javascript
function prod(a,b)
{ x=a*b
  return x }
```
JavaScript PopUp

- Alert Box
  - An alert box is often used if you want to make sure information comes through to the user (jscript4)

- Confirm Box
  - A confirm box is often used if you want the user to verify or accept something (jscript10)

- Prompt Box
  - A prompt box is often used if you want the user to input a value before entering a page (jscript11)
for, while, ... 

for (var=startvalue; var<=endvalue; var=var+increment) 
{ code to be executed } 

while (var<=endvalue) 
{ code to be executed } 

do 
{ code to be executed } 
while (var<=endvalue) 

- The **break** command will break the loop and continue executing the code that follows after the loop - ex
- The **continue** command will break the current loop and continue with the next value - ex
Events

- Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger JavaScript functions.
- Events are normally used in combination with functions - the function is executed after the event occurs.
- **onload and onUnload** - [jscrip5]
- **onFocus, onBlur and onChange** - [jscrip6]
- **onSubmit** - [jscript7]
- **onMouseOver and onMouseOut** - [ex, ex2]
Objects

- In JavaScript there are a number of built-in **objects**: special kind of data
- **Properties** are the values associated with an object
  
  Example: **length** is a property of the object **txt**
  
  ```javascript
  <script type="text/javascript">
  var txt="Hello World!" document.write(txt.length)
  </script>
  ```

- **Methods** are the actions that can be performed on objects
  
  Example: **write** and **toUpperCase** are methods
  
  ```javascript
  var str="Hello world!"
document.write(str.toUpperCase())
  ```
Objects

- **The String object** is used to manipulate a stored piece of text:
  ```javascript
  var txt="Hello world!"; // ref
  ```

- **The Date object** is used to work with dates and times:
  ```javascript
  var myDate=new Date(); // ref
  ```
  ```javascript
  var myDate=new Date();
  myDate.setDate(myDate.getDate()+5)
  ```

- **The Array object** is used to store a set of values in a single variable name:
  ```javascript
  var mycars=new Array();
  mycars[0]="Saab"
  mycars[1]="Volvo"
  mycars[2]="BMW"
  ```

- **The Boolean object** is an object wrapper for a Boolean value:
  ```javascript
  var myBoolean=new Boolean(false)
  var myBoolean=new Boolean(true)
  ```

- **The Math object** allows you to perform common mathematical tasks:
  ```javascript
  ```
More JavaScript Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window</strong></td>
<td>The top level object in the JavaScript hierarchy. The Window object represents a browser window. A Window object is created automatically with every instance of a <code>&lt;body&gt;</code> or <code>&lt;frameset&gt;</code> tag.</td>
</tr>
<tr>
<td><strong>Navigator</strong></td>
<td>Contains information about the client's browser</td>
</tr>
<tr>
<td><strong>Screen</strong></td>
<td>Contains information about the client's display screen</td>
</tr>
<tr>
<td><strong>History</strong></td>
<td>Contains the visited URLs in the browser window</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>Contains information about the current URL</td>
</tr>
</tbody>
</table>
The **HTML DOM** is a W3C standard and it is an abbreviation for the **Document Object Model** for HTML.

The HTML DOM defines a **standard set of objects for HTML**, and a standard way to access and manipulate HTML documents.

All HTML elements, along with their containing text and attributes, can be accessed through the **DOM**.

The HTML DOM is **platform and language independent** - can be used by any programming language (Java, JavaScript, and VBScript).
DOM structure

- According to the DOM, **everything** in an HTML document is a **node**
- Nodes have a **hierarchical relationship** to each other
Accessing nodes: by ID

- The `getElementById()` method returns the element with the specified ID

```html
<html>
<head>
<script type="text/javascript">
function getValue()
{
var x=document.getElementById("myHeader")
alert(x.innerHTML)
}
</script>
</head>
<body>
<h1 id="myHeader" onclick="getValue()">This is a header</h1>
<p>Click on the header to alert its value</p>
</body>
</html>
```
Accessing nodes: by tag name

- The getElementsByTagName() can be used on any HTML element, and also on the document

```html
<html>
<head>
<script language="JavaScript">
function function1() {
    var m = document.getElementsByTagName("P");
    alert(m.length);
}
</script>
</head>
<body>
<p>This is the first paragraph</p>
<p>This is the second paragraph</p>
<input type="button"
    value="Get the number of p elements in this page"
    onClick="function1();">
</body>
</html>
```
### Some DOM Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document</strong></td>
<td>Represents the entire HTML document and can be used to access all elements in a page</td>
</tr>
<tr>
<td><strong>Anchor</strong></td>
<td>Represents an <code>&lt;a&gt;</code> element</td>
</tr>
<tr>
<td><strong>Body</strong></td>
<td>Represents the <code>&lt;body&gt;</code> element</td>
</tr>
<tr>
<td><strong>Button</strong></td>
<td>Represents a <code>&lt;button&gt;</code> element</td>
</tr>
<tr>
<td><strong>Form</strong></td>
<td>Represents a <code>&lt;form&gt;</code> element</td>
</tr>
<tr>
<td><strong>Image</strong></td>
<td>Represents an <code>&lt;img&gt;</code> element</td>
</tr>
<tr>
<td><strong>Input button</strong></td>
<td>Represents a button in an HTML form</td>
</tr>
<tr>
<td><strong>Input text</strong></td>
<td>Represents a text-input field in an HTML form</td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td>Represents a <code>&lt;link&gt;</code> element</td>
</tr>
<tr>
<td><strong>Meta</strong></td>
<td>Represents a <code>&lt;meta&gt;</code> element</td>
</tr>
<tr>
<td><strong>Style</strong></td>
<td>Represents an individual style statement</td>
</tr>
<tr>
<td><strong>Table</strong></td>
<td>Represents a <code>&lt;table&gt;</code> element</td>
</tr>
<tr>
<td><strong>Textarea</strong></td>
<td>Represents a <code>&lt;textarea&gt;</code> element</td>
</tr>
</tbody>
</table>

[http://www.w3schools.com/js/js_obj_htmldom.asp](http://www.w3schools.com/js/js_obj_htmldom.asp)