Asynchronous JavaScript Technology and XML: Ajax

Corso di ***Applicazioni Telematiche***

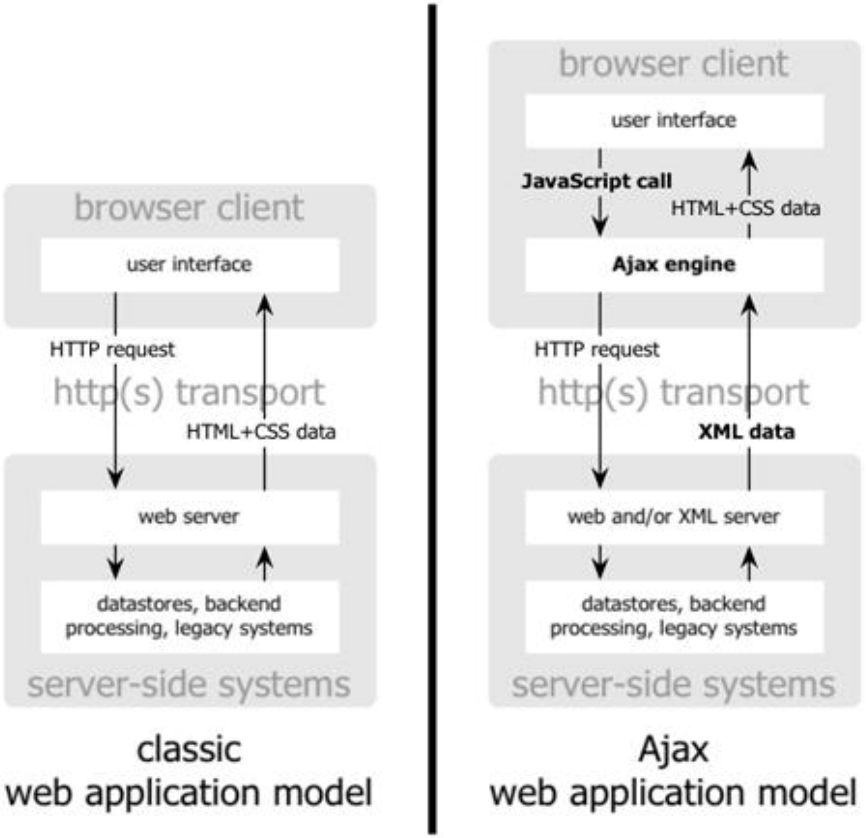
A.A. 2008-09 – Lezione n.23

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

AJaX : Asynchronous JavaScript and XML

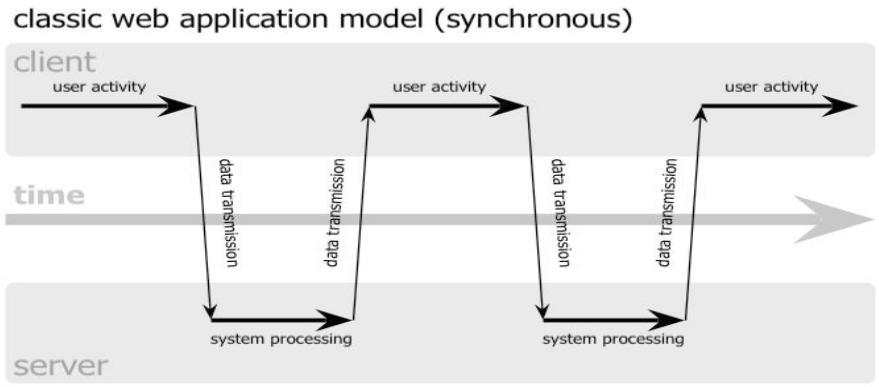


Le applicazioni AJaX sono delle applicazioni a pagina singola, in cui viene caricato l'*AJaX Engine* che permette di inviare richieste HTTP al server.

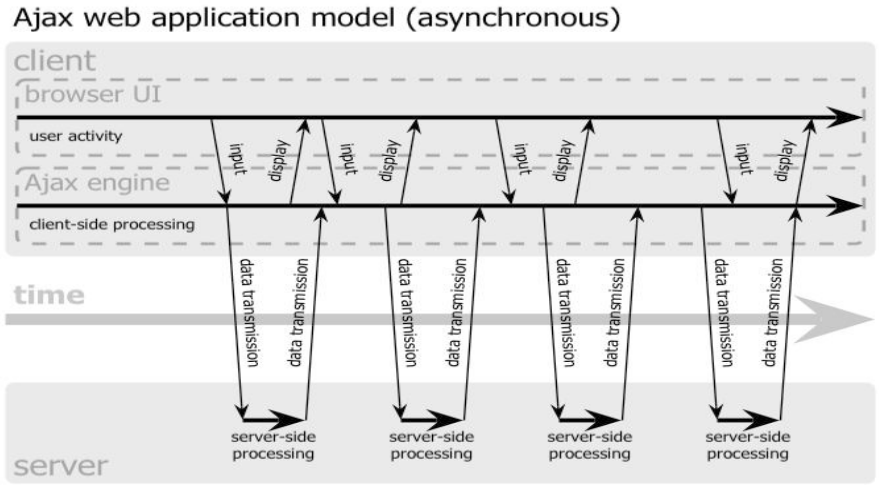
Il server non restituirà pagine HTML ma dati in formato XML con cui l'AJaX Engine aggiornerà la pagina.

L'AJaX Engine è implementato con un insieme di funzioni JavaScript che utilizzano l'oggetto *XMLHttpRequest* per inviare le richieste.

Modello di interazione AJaX



In un'applicazione classica l'utente invia i dati ed attende la risposta del server per poter continuare le proprie attività (Interazione Sincrona)

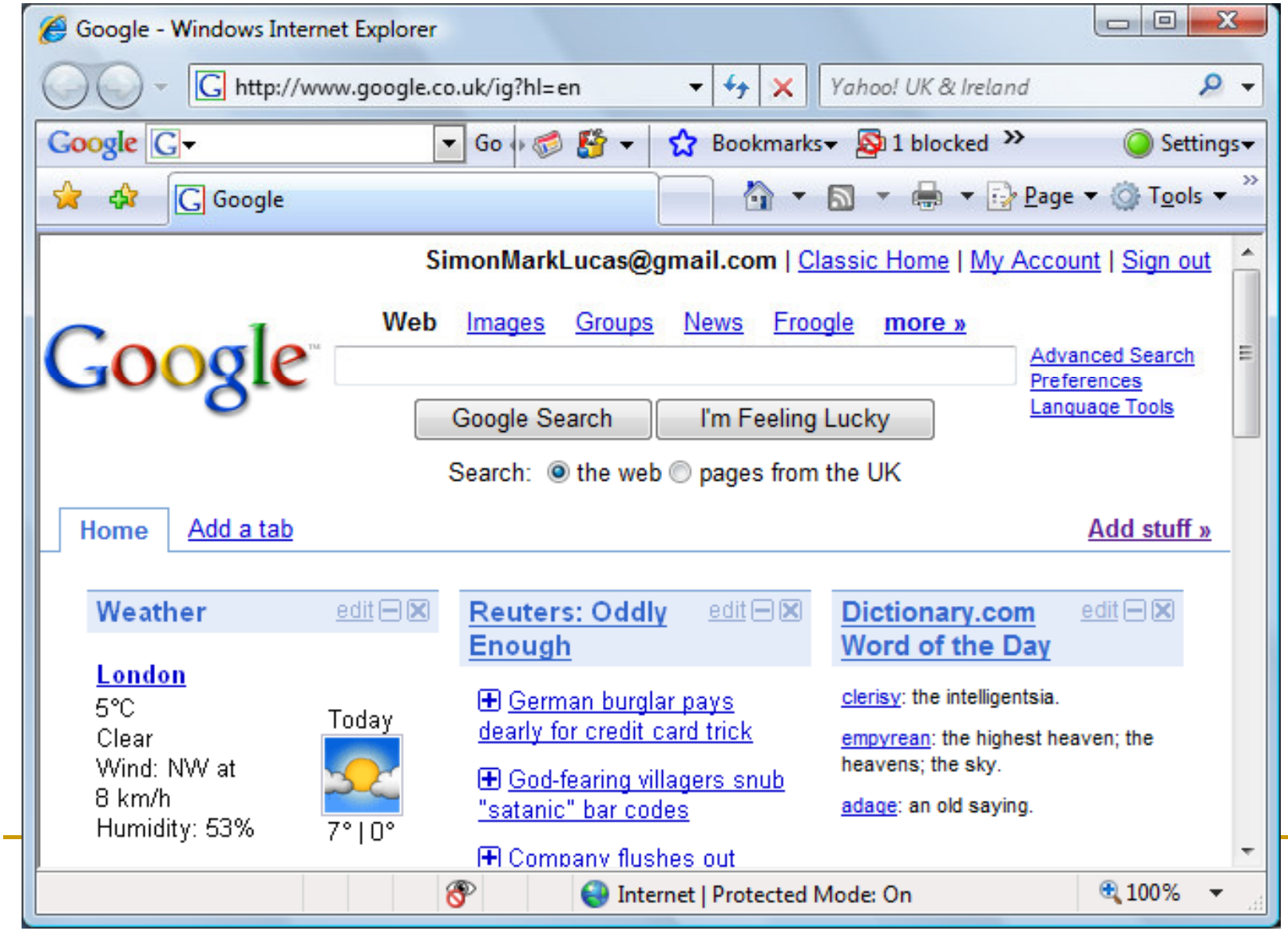


In un'applicazione AJaX i dati vengono inviati dall'AJaX engine in *background* rispetto all'interazione con l'utente, che può interagire con l'applicazione senza attendere la risposta del server (Interazione Asincrona)

Ajax: objectives

- Ajax is a technique for creating “better, faster, more responsive web applications”
- Web applications with Ajax are supposed to replace all our traditional desktop applications
- Ajax-enabled web is often referred to as “Web 2.0”

Example: Personalised Google



How Ajax works

1. You do something with an HTML form in your browser
2. JavaScript on the HTML page sends an HTTP request to the server
3. The server responds with a *small amount* of data, rather than a complete web page
4. JavaScript uses this data to modify the page

- This is faster because less data is transmitted and because the browser has less work to do
- NOTE: You have not browsed to a new URL; the URL "Forward" and "Back" button are not involved.

Ajax: applications

- **Real-time server-side form data validation** before the user submits a form
- **Autocompletion** of form data
- **Load on demand** based on a client event
- **Sophisticated user interface controls and effects** modifying the content of an HTML page, not requiring the user to reload the page
- **Partial submit:** An HTML page can submit form data as needed without requiring a full page refresh

Ajax: applications (cont.)

- **Refreshing data and server push:** HTML pages may poll data from a server for up-to-date data
  - Polling is not the most effecient means of ensuring that data on a page is the most current. Emerging techniques such as Comet are being developed to provide true server-side push over HTTP by keeping a persistent connection between the client and server.

Ajax: applications (cont.)

- **Mashups:** An HTML page can obtain data by mixing content or data from a third-party application such as Google Maps with your own application
- **Page as an application:** Ajax techniques can be made to create single-page applications that look and feel much like a desktop application

The XMLHttpRequest object

- JavaScript has to create an XMLHttpRequest object
- For historical reasons, there are three ways of doing this
  - For most browsers, just do
    ```
    var request = new XMLHttpRequest();
    ```
  - For some versions of Internet Explorer, do
    ```
    var request = new ActiveXObject("Microsoft.XMLHTTP");
    ```
  - For other versions of Internet Explorer, do
    ```
    var request = new ActiveXObject("Msxml2.XMLHTTP");
    ```
- The next slide shows a JavaScript function for choosing the right way to create an XMLHttpRequest object

Preparing the XMLHttpRequest object

- Once you have an XMLHttpRequest object, you have to prepare it with the open method
- ***request***.open(***method***, ***URL***, ***asynchronous***)
  - The ***method*** is usually 'GET' or 'POST'
  - The ***URL*** is where you are sending the data
    - If using a 'GET', data is appended to the URL
    - If using a 'POST', data is added in a later step
  - If ***asynchronous*** is true, the browser does not wait for a response (this is what you usually want)
- ***request***.open(***method***, ***URL***)
  - As above, with ***asynchronous*** defaulting to true

Sending the XMLHttpRequest object

- Once the XMLHttpRequest object has been prepared, you have to send it
- ***request***.send(null);
  - This is the version you use with a GET request
- ***request***.send(***content***);
  - This is the version you use with a POST request
  - The content has the same syntax as the suffix to a GET request
  - POST requests are used less frequently than GET requests
  - Example:

```
request.setRequestHeader('Content-Type',
                         'application/x-www-form-urlencoded');
request.send('var1=' + value1 + '&var2=' + value2);
```

On the server side

- The server gets a completely standard HTTP request
- In a servlet, this would go to a doGet or doPost method
- The response is a completely standard HTTP response
- Instead of returning a complete HTML page as a response, the server returns an arbitrary text string (possibly XML, possibly something else)

Getting the response

- Ajax uses asynchronous calls—you don't wait for the response
- Instead, you have to handle an event
  - request.onreadystatechange = someFunction;
    - This is a function assignment, *not* a function call
    - When the function is called, it will be called with no parameters
  - ```
    function someFunction() {
        if(request.readyState == 4){
            var response = request.responseText;
            // Do something with the response
        }
    }
    ```
- To be safe, set up the handler *before* you call the send function
    ```

The readyState property

- The readyState property defines the current state of the XMLHttpRequest object.
- Here are the possible values for the readyState property:
  - **readyState=0** after you have created the XMLHttpRequest object, but before you have called the open() method.
  - **readyState=1** after you have called the open() method, but before you have called send().
  - **readyState=2** after you have called send().
  - **readyState=3** after the browser has established a communication with the server, but before the server has completed the response.
  - **readyState=4** after the request has been completed, and the response data have been completely received from the server.
- Not all browsers use all states
- Usually you are only interested in state 4

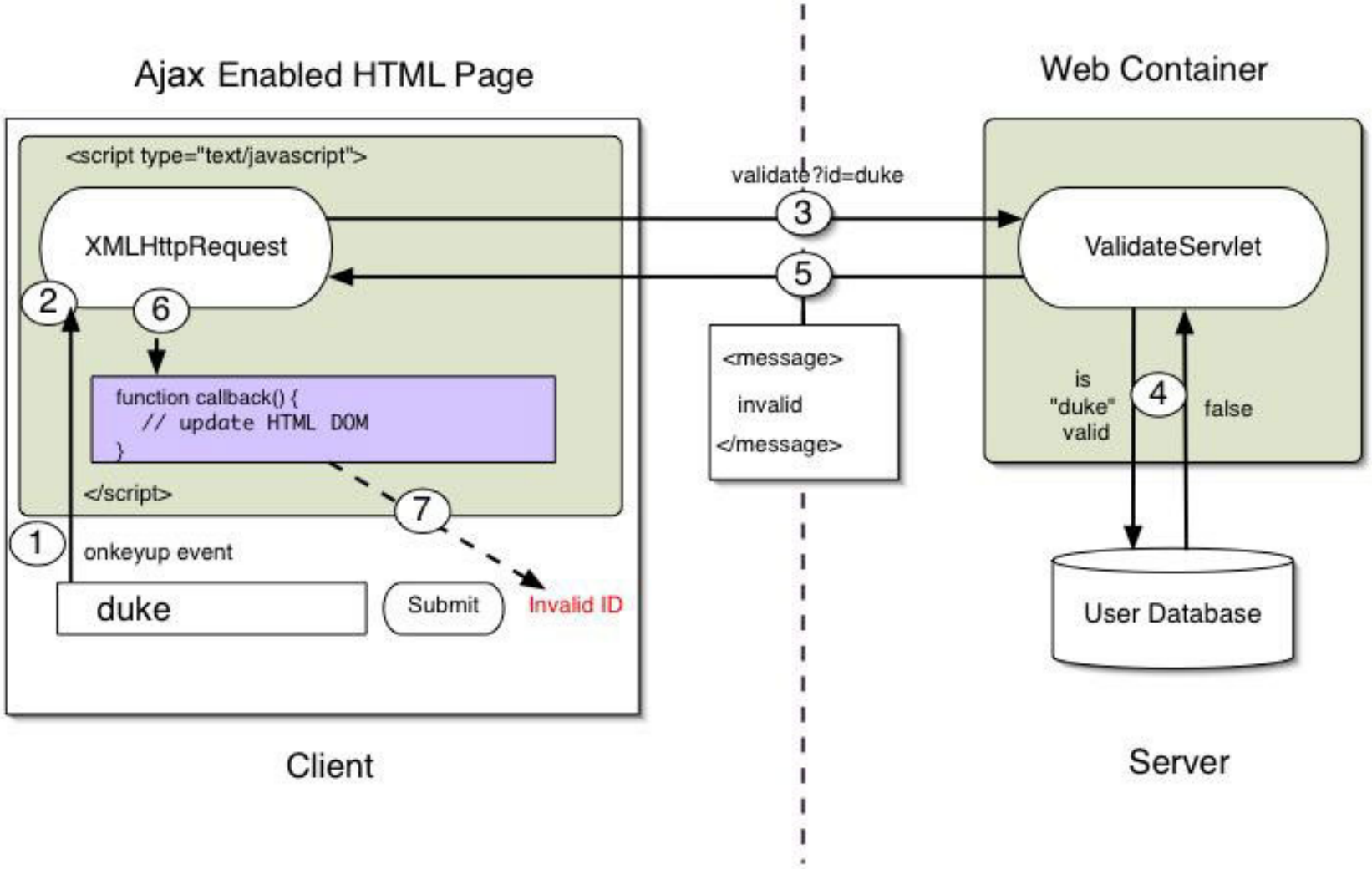
Ajax example: interaction with a validating servlet

- A web application contains a static HTML page with a form that requires server-side logic to validate form data without refreshing the page
- A server-side web component (servlet) named ValidateServlet provides the validation logic
- From:
  http://java.sun.com/developer/technicalArticles/J2EE/AJAX/

Ajax example

- Steps:
  - A client event occurs.
  - An XMLHttpRequest object is created and configured.
  - The XMLHttpRequest object makes a call.
  - The request is processed by the ValidateServlet.
  - The ValidateServlet returns an XML document containing the result.
  - The XMLHttpRequest object calls the callback() function and processes the result.
  - The HTML DOM is updated.

Ajax example: interactions



Ajax example: Javascript

- HTML form
  - `<input type="text" size="20" id="userid" name="id" onkeyup="validate();">`
- Javascript code

```
var req;
function validate() {
var idField = document.getElementById("userid");
var url="validate?id="+encodeURIComponent(idField.value);
if (typeof XMLHttpRequest != "undefined") {
   req = new XMLHttpRequest(); }
else if (window.ActiveXObject) {
   req = new ActiveXObject("Microsoft.XMLHTTP"); }
req.open("GET", url, true);
req.onreadystatechange = callback;
req.send(null);
}
```

Ajax example: validating servlet

```
public class ValidateServlet extends HttpServlet {
  private ServletContext context;
  private HashMap users = new HashMap();
  public void init(ServletConfig config)
  throws ServletException
  {
    super.init(config);
    this.context = config.getServletContext();
    users.put("greg","account data");
    users.put("duke","account data");
  }

...
```

Ajax example: validating servlet

```
...
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws IOException, ServletException
  {
    String targetId = request.getParameter("id");
    if ((targetId!=null)&&!users.containsKey(targetId.trim()))
    {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("<message>valid</message>");
    } else
    {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("<message>invalid</message>");
    }
  }
}
```

Ajax example: Javascript - callback

```
function callback() {
  if (req.readyState == 4) {
    if (req.status == 200) {
      // update the HTML DOM based on
      // whether or not message is valid
      parseMessage();
    }
  }
}

function parseMessage() {
  var message =
  req.responseXML.getElementsByTagName("message")[0];
  setMessage(message.childNodes[0].nodeValue);
}
```

Ajax sample: HTML with JavaScript

```
<html><head>
<script type="text/javascript">
var req;
function validate() { … }
function callback() { … }
function setMessage(message) {
    var mdiv = document.getElementById("userIdMessage");
    if (message == "invalid") {
mdiv.innerHTML="<div style=\"color:red\">Invalid User Id</div>";
    } else {
mdiv.innerHTML="<div style=\"color:green\">Valid User Id</div>";
    }
}
</script>
</head>
```

Ajax sample: HTML file (cont.)

```
…
<body>
<input type="text"
            size="20"
              id="userid"
            name="id"
         onkeyup="validate();">
<div id="userIdMessage"></div>
</body>
</html>
```

Ajax sample in action

Ajax frameworks: JaxCent

- From:
  http://trijug.org/downloads/JaxcentPresentationTrijug.pdf
- HelloWorld.html:

```
<HTML>
<HEAD>
<TITLE>Hello World Sample</TITLE>
<SCRIPT SRC="/jaxcent20.js"></SCRIPT>
</HEAD>
<BODY>
The following paragraph has been marked with an ID for
ease of finding it from Java.
<P ID="text">Some old text</P>
</BODY>
</HTML>
```

JaxCent: Java class

- HelloWorld.java:

```
package jaxlet;
import jaxcent.*;
public class HelloWorld extends jaxcent.JaxcentPage {
   HtmlPara para = new HtmlPara( this, “text” );
   public HelloWorld()
   {
     try {
       para.setInnerHTML( “Hello, World!” );
     } catch (Exception ex) {
       ex.printStackTrace();
     }
   }
}
```

GWT: Google Web Toolkit

- Ajax programming is hard without the right support
- So many things to worry about:
  - HTML
  - JavaScript
  - Java (or other server-side language)
  - JSP?
  - CSS
- GWT: develop entire web applications in pure Java
  - With a simple HTML container page to indicate basic layout