AJAX

Andrea Ferracani
Client side programming
client - server communication
andreaferracani@gmail.com

DOM - Document Object Model

The **Document Object Model** (**DOM**) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.

Objects in the DOM tree may be addressed and manipulated by using methods on the objects.

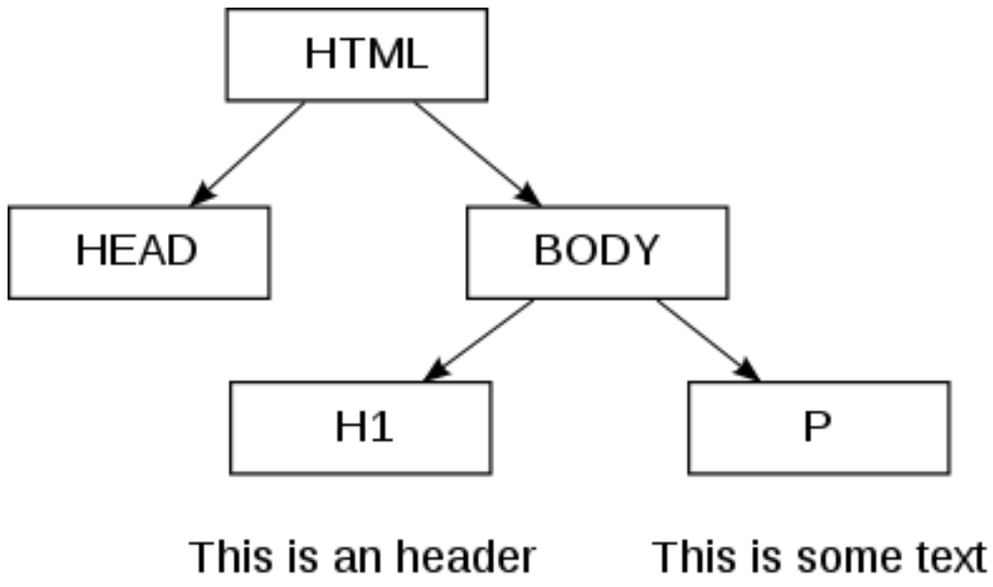
The public interface of a DOM is specified in its application programming interface (API).

Web browsers usually use an internal model similar to the DOM in order to render a document (such as an HTML page). **The DOM APIs are also used to inspect or modify a Web page from JavaScript code**. In other words, the Document Object Model is the way JavaScript sees the browser state and the HTML page it contains.

When an HTML page is rendered in a browser, the browser parses the markup (e.g. HTML), downloaded from the web-server into an **in-memory DOM**. The DOM is used to construct additional internal structures used to display the page in the browser window.

The nodes of every document are organized in a **tree structure, called the DOM tree**. The topmost node in the DOM tree is the **Document** object. Each node has zero or more children.

DOM - Document Object Model




```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
        |-> Anchor
          |-> text node
      |-> Form
          |-> Text-box
          |-> Text Area
          |-> Radio Button
          |-> Check Box
          |-> Select
          |-> Button
```

What’s AJAX

Ajax is a technology used to **improve the user experience** of a web application.

The acronym stands for **Asynchronous JavaScript and XML**.

Ajax does not represent a new technology. It 'a **set of technologies**:

- **Javascript**: used to create client-side functionality, to manipulate objects in the DOM (document object model)
- **XMLHttpRequest**: it is the object that allows asynchronous communication with the server allowing you to send parameters (name, value pairs) to a server in a GET or POST
- **A server side language**: in our case PHP, it will not answer an HTML page but in a data interchange format: **XML** or **JSON** (JavaScript Object Notation)

Javascript intro: http://www.w3schools.com/js/js_intro.asp, http://www.w3schools.com/jsref/default.asp, http://www.w3schools.com/htmldom/dom_methods.asp
XMLHttpRequest: http://www.w3schools.com/xml/xml_http.asp
XML and JSON samples: http://www.w3schools.com/xml/xml_view.asp, http://www.w3schools.com/json/default.asp

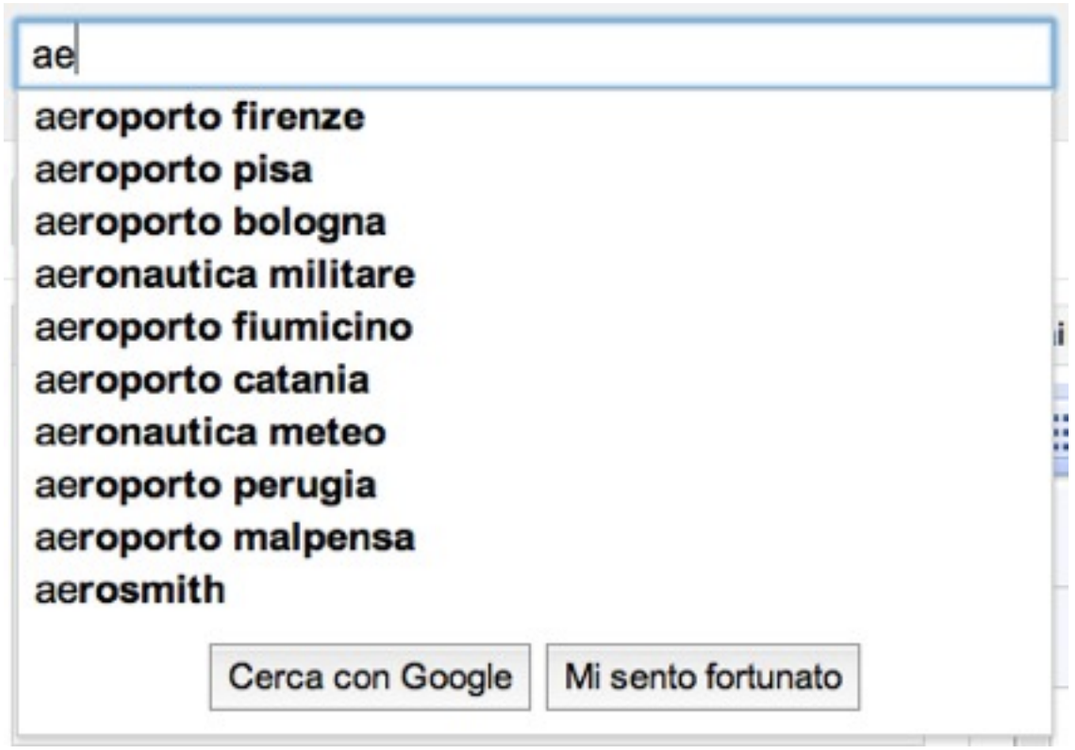
Why AJAX

Delete the page reload: can make asynchronous calls to a server.

Simple scenario

If we were to implement **a system of suggestion of an input form** we would have 3 possible scenarios:

- The user fills and submits the form (html) to the server (php for example.) The server checks input data and returns the page. In this case between request and response there is a **time delay**.
- The request is performed **using JavaScript** as the user types. In this case there is no delay but we have only **a subset of functionality** made possible by a client side language.
- **Through Ajax**, it benefits from the possibilities of JavaScript, but also it is able to communicate with a server side language as the user types in the fields of the form (think of a combo of suggestions populated based on the characters entered by the user)



AJAX was made popular in 2005 by Google, with **Google Suggest**.

Google Suggest is using AJAX to create a very dynamic web interface: when you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

Others client side technologies

On your computer you can also use:

- **Java applets**: runs within the browser through the Java virtual machine
- **Macromedia Flash**: runs within the browser via the Flash player plugin
- **Microsoft Silverlight**: runs within the browser through a light version of the. NETFramework

The above technologies are rather **heavy for simple tasks**, such as the validation of a form.

AJAX history

- **1995**: asynchronous loading introduced in Java applets in the first version of the Java language.
- **1996**: Internet Explorer introduced the iframe element to HTML, which also enabled asynchronous loading
- **1999**: Microsoft utilized its iframe technology to dynamically update the news stories and stock quotes on the default page for Internet Explorer (http://home.microsoft.com).
- **1999**, Microsoft created the XMLHTTP ActiveX control in Internet Explorer 5, which was later adopted by Mozilla, Safari, Opera and other browsers as the XMLHttpRequest JavaScript object.
- Microsoft has adopted the native XMLHttpRequest model as of Internet Explorer 7, though the ActiveX version is stil

Best of AJAX

There are many Ajax applications, and we use it every day!

- Bing Maps
- Google Maps
- Yahoo Maps
- Flickr
- Picasa Web Albums
- Google
- Yahoo
- Gmail
- Digg

Benefits vs. disavantages

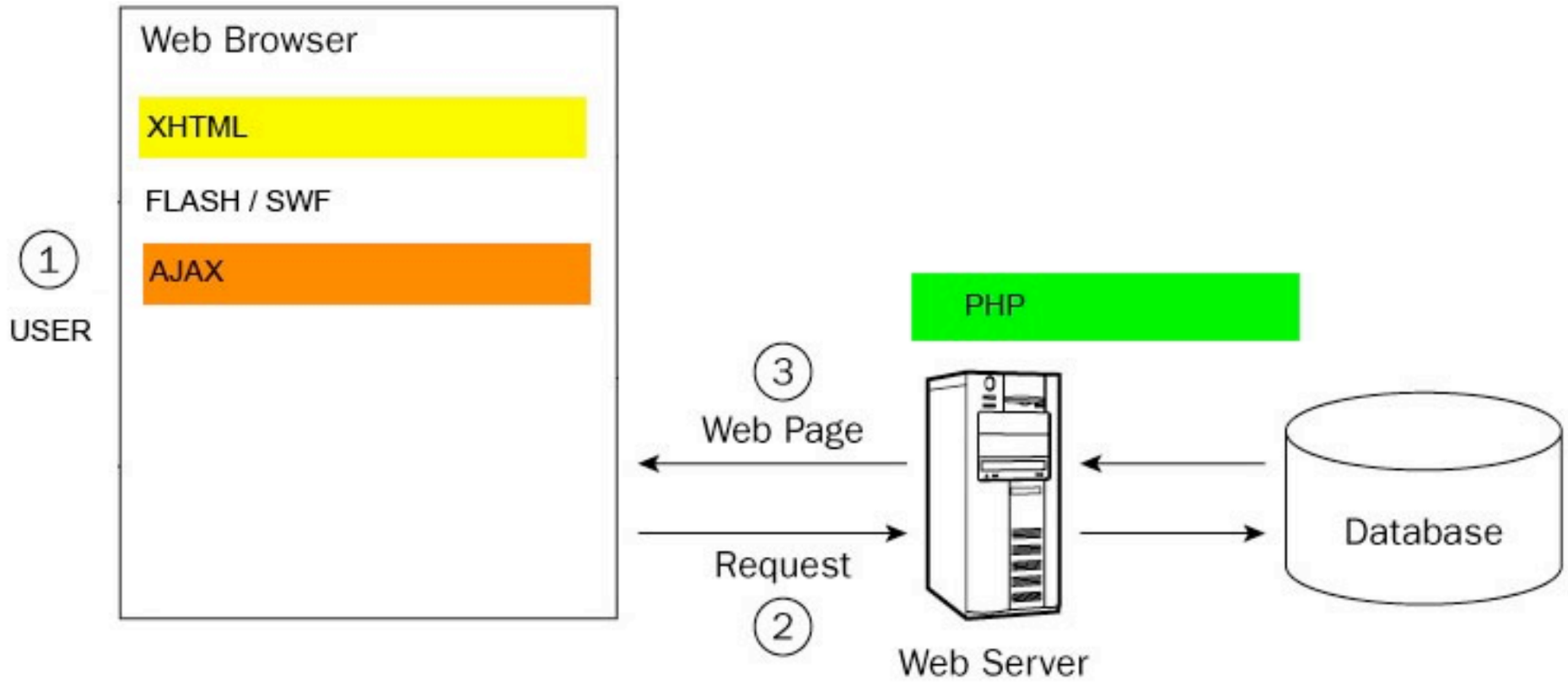
Benefits:

- you can create applications **responsive and intuitive**
- uses existing technologies so there is **nothing to install**

Disavantages:

- people often forget that **not all browsers support JavaScript** or it can be disabled on the client side: the basic functionality should exist anyway. **Ajax is a plus**.
- Search engines do not have the ability to run JavaScript. So the **content loaded with Ajax are not indexed**.
- To **bookmark** an AJAX page is problematic because the address bar of the browser will not change when loading content. It's necessary to save application states, for example through anchors. www.my-example.com/my-Ajax-app/#state1
- Same problem as the previous point is with the **back and forward buttons**.
- Same origin policy prevents some Ajax techniques from being used across domains, although the W3C has a draft of the XMLHttpRequest object that would enable this functionality. Methods exist to sidestep this security feature by using a special Cross Domain Communications channel embedded as an iframe within a page, or by the use of JSONP.

  check http://code.google.com/p/reallysimplehistory/

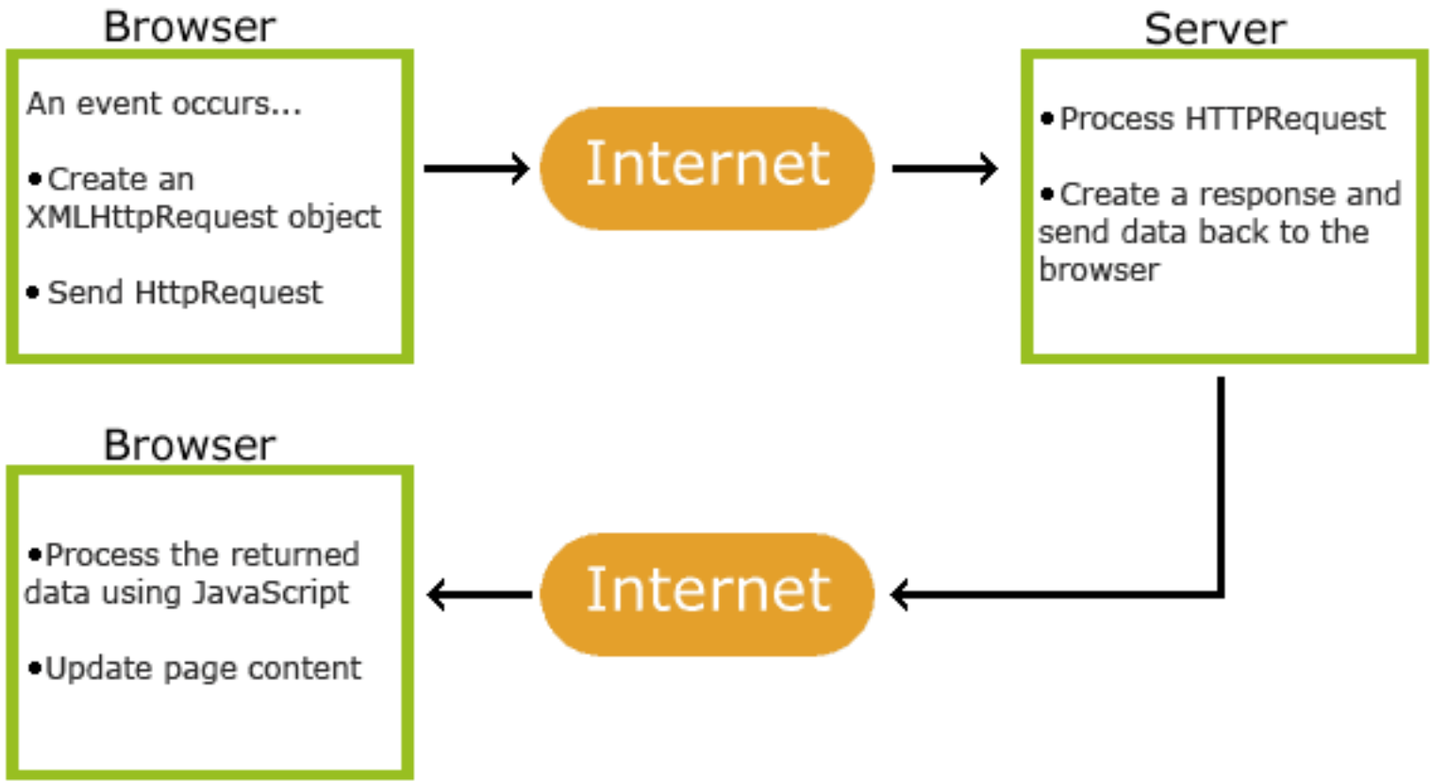


- XHTML: it defines the markup of the web page
- FLASH: it adds interactivity to the XHTML page and communicates with PHP
- AJAX: it adds interactivity to the XHTML page and communicates with PHP

- PHP: it communicates with FLASH /AJAX widgets and with the database

AJAX in short

- It sends server requests by the **XMLHttpRequest object**
- It manages responses in **XML** o **JSON**
- It is able to get **references to tags** in the HTML page
- **It generates HTML** parsing and understanding server responses



XMLHttpRequest - First sample

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>

<link rel="stylesheet" href="../../css/style.css" type="text/css" />

        <!-- CHIAMATA A JAVASCRIPT -->
        <script type="text/javascript" src="json2.js"></script>
        <script type="text/javascript" src="mysql_3.js"></script>

</head>
```

XMLHttpRequest - sample

http://localhost:8888/ajax_course_2011/examples/5_mysql/index.html

Loads a list of books asynchronously

XHTML code

```
//function called when the page loads
<body>

    <div class="content">

        <h2>PHP e MYSQL</h2>
        <p>Questa lista di libri è caricata dinamicamente dal <strong>database</strong> con
        <strong>AJAX</strong>!</p>

        //function called when user press the button
        <button onclick='process()'>Carica lista</button>

         <!-- wrapper of server response -->
         <div id="myDivElement"></div>

    </div>

  </body>
</html>
```

XMLHttpRequest - sample

AJAX code

```
// it contains the reference to the XMLHttpRequest object, instantiation
var xmlHttp = createXmlHttpRequestObject();

// cross browsers function to create the instance
function createXmlHttpRequestObject() {

// variable which contains the reference to the XMLHttpRequest object
        var xmlHttp;

// se il browser è Internet Explorer 6 o più vecchio
        if(window.ActiveXObject) {
                try {
                        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {
                        xmlHttp = false;
                }
        }
```

XMLHttpRequest - sample

Codice AJAX

```
// check if Mozilla or others
else {
        try {
                xmlHttp = new XMLHttpRequest();
        } catch (e) {
                xmlHttp = false;
        }
}

        // return the object or create error alert
        if (!xmlHttp) {
                alert("Error creating the XMLHttpRequest object.");
        } else {
                return xmlHttp;
        }
}
```

XMLHttpRequest - sample

Codice AJAX

```
// function which opens the connection with PHP by XMLHttpRequest object and make the object waiting for the response

function process() {
        // if XMLHttpRequest object exists
        if (xmlHttp) {

        // try to connect to server
        try {
            // open the connection
            xmlHttp.open("GET", "index.php", true);

            // set the event handler
            xmlHttp.onreadystatechange = handleRequestStateChange;

            // send the request
            xmlHttp.send(null);
        }
        // error message
        catch (e) {
            alert("Can't connect to server:\n" + e.toString());
        }
}
```

XMLHttpRequest - sample

JAX code

```
// function which checks the request state on server

function handleRequestStateChange() {

        // if readyState is 4, completed
        if (xmlHttp.readyState == 4) {

                // check if HTTP status is OK
                if (xmlHttp.status == 200) {
                        try {
                                // if OK, make something
                                handleServerResponse();
                        } catch(e) {
                                // show error message
                                alert("Error reading the response: " + e.toString());
                        }
                } else {
                    // show status message
                    alert("There was a problem retrieving the data:\n" + xmlHttp.statusText);
                }
        }
}
```

XMLHttpRequest - sample

AJAX code

```
// functio which handle server response

function handleServerResponse() {

    // read the response
    responseJSON = JSON.parse(xmlHttp.responseText);

    // generate HTML output
    var html = "";

    // iterate the array e generate HTML
    for (var i=0; i<responseJSON.books.length; i++) {
        html += responseJSON.books[i].title +  "<br/>";
    }

    // obtain a reference to the html wrapper
    myDiv = document.getElementById("myDivElement");

    // generate HTML
    myDiv.innerHTML = "<p>Server says: </p>" + html;
}
```

XMLHttpRequest - sample

PHP code

//**index.php** - this file makes the query on the database, format the data and return them to AJAX

```
<?php
  // add headers that tell to the browser the format of the response
  header('Content-Type: text/json');

   // include necessary files for database connection and error handling
  require_once('config.php');
  require_once('error_handler.php');

// connect to the database
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);

// formulate the query
$query = 'SELECT book_id, book_title FROM books';

// execute query
$result = $mysqli->query($query);

// create an array for books
$books = array();
```

XMLHttpRequest - sample

PHP code

```
// fetch the result
while ($row = $result->fetch_array(MYSQLI_ASSOC)) {

    // extract book id and title
    $book_id = $row['book_id'];
    $book_title = $row['book_title'];

    // create an associative array per for each book and put it in the books array
    $book = array('title' => $book_title,'id' =>$book_id);
    array_push($books, $book);
}

// create a new associative array to label the books array
$response = array('books' => $books);

// encode the array in JSON
echo json_encode($response);

// close input stream
$result->close();

// close database connection
$mysqli->close();
?>
```

XMLHttpRequest - sample

PHP code

```
// File config.php: define constants to access database

<?
    // defines database connection data
    define('DB_HOST', 'localhost');

    define('DB_USER', 'ajax_course_2013');

    define('DB_PASSWORD', 'ajax_course_2013');

    define('DB_DATABASE', 'ajax_course_2013');
?>
```

XMLHttpRequest - sample

Codice PHP

```
// File error_handler.php: custom function override for error handling

<?php
   // assign default action for error handling
   set_error_handler('error_handler', E_ALL);

   // define error handling function
   function error_handler($errNo, $errStr, $errFile, $errLine) {

      // clean output already generated
      if(ob_get_length()) ob_clean();

      // error output
      $error_message = 'ERRNO: ' . $errNo.chr(10) . 'TEXT: ' . $errStr.chr(10) . 'LOCATION: ' . $errFile .
                       ', line ' . $errLine;

      echo $error_message;
       exit;
  }
?>
```

XMLHttpRequest - Others examples

- http://localhost:8888/ajax_course_2013/examples.html
- http://www.w3schools.com/ajax/ajax_aspphp.asp
- http://www.w3schools.com/ajax/ajax_database.asp
- http://www.w3schools.com/ajax/ajax_xmlfile.asp

RDBMS

We used a Relational **Database Management System (RDBMS)** to store data. (MAMP / XAMP / WAMP / LAMP / AppServ)

So you have to:

- Create a **database**
- Write **SQL queries** to find and manipulate data
- Connecting to MySQL through **PHP code**
- **Send SQL queries** to the database

In creating the tables within a database the meaning of these terms are important:

- **Primary keys**: the primary key is a special column that ensures that each record is unique (usually the column of ID)
- **Data types**: numeric types, characters and strings, dates
- **NULL** and **NOT NULL** columns: if the field of a column it can be undefined
- **Default values** of columns
- **Auto_increment** columns: is usually used for the primary key
- **Indexes** are objects of the database that improve the search columns, unfortunately slow down other operations!

RDBMS

SELECT sample

SELECT <column list>
FROM <table name(s)>
[WHERE <restrictive condition(s)>]

INSERT sample

INSERT INTO <table name> [(column list)]
VALUES (column values)

UPDATE sample

UPDATE <table name>
SET <column name> = <new value> [, <column name> = <new value> ...]
[WHERE <restrictive condition>]

DELETE sample

DELETE FROM <table name>
[WHERE <restrictive condition>]

How to add records in our database:

INSERT INTO books (book_title) VALUES ('Ajax first book');
INSERT INTO books (book_title) VALUES ('Ajax second book');
INSERT INTO books (book_title) VALUES ('Ajax third book');

XHR2 and HTML5

Strictly speaking **XHR2 isn't HTML5**. However, it's part of the incremental improvements browser vendors are making to the core platform.

XMLHttpRequest Level 2 introduces a slew of new capabilities which put an end to crazy hacks in our web apps; things like:

- cross-origin requests,
- uploading progress events, and
- support for uploading/downloading binary data.

These allow AJAX to work in concert with many of the bleeding edge HTML5 APIs such as **File System API**, Web Audio API, and WebGL.

XHR2 support: http://caniuse.com/xhr2

XHR2 and HTML5

Now XHR can specify a response format:

xhr.**responseType**

Before sending a request, set the xhr.responseType to "text", "arraybuffer", "blob", or "document", depending on your data needs. Note, setting xhr.responseType = '' (or omitting) will default the response to "text".

xhr.**response**

After a successful request, the xhr's response property will contain the requested data as aDOMString, ArrayBuffer, Blob, or Document (depending on what was set forresponseType.)

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {
  if (this.status == 200) {
    // Note: .response instead of .responseText
    var blob = new Blob([this.response], {type: 'image/png'});
    ...
  }
};

xhr.send();
```

XHR2 data download

A Blob can be used in a number of places, including saving it to **indexedDB**, writing it to the **HTML5 File System**, or creating an **Blob URL**, as seen in this example.

```
window.URL = window.URL || window.webkitURL;  // Take care of vendor prefixes.

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {
  if (this.status == 200) {
    var blob = this.response;

    var img = document.createElement('img');
    img.onload = function(e) {
      window.URL.revokeObjectURL(img.src); // Clean up after yourself.
    };
    img.src = window.URL.createObjectURL(blob);
    document.body.appendChild(img);
    ...
  }
};

xhr.send();
```

XHR2 data upload

Being able to download data in different formats is great, but it doesn't get us anywhere if we can't send these rich formats back to the server.

XMLHttpRequest has limited us to sending DOMString or Document (XML) data for some time. **Not anymore**.

A revamped **send() method** has been overridden to accept any of the **following types**:

- DOMString
- Document
- FormData
- Blob
- File
- ArrayBuffer.

XHR2 data upload

Sending string data: xhr.send(DOMString)

```
function sendText(txt) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) {
    if (this.status == 200) {
      console.log(this.responseText);
    }
  };
  xhr.send(txt);
}

sendText('test string');
```

```
function sendTextNew(txt) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.responseType = 'text';
  xhr.onload = function(e) {
    if (this.status == 200) {
      console.log(this.response);
    }
  };
  xhr.send(txt);
}

sendTextNew('test string');
```

XHR2 data upload

Many people are probably accustomed to using **jQuery plugins** or other libraries to handle AJAX form submissions. Instead, we can use **FormData**, another new data type conceived for XHR2. FormData is convenient for creating an HTML <form> on-the-fly, in JavaScript.

```
function sendForm() {
  var formData = new FormData();
  formData.append('username', 'johndoe');
  formData.append('id', 123456);

  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);
}
```

XHR2 data upload

FormData objects can be initialized from and existing **HTMLFormElement** on the page.

```
<form id="myform" name="myform" action="/server">
  <input type="text" name="username" value="johndoe">
  <input type="number" name="id" value="123456">
  <input type="submit" onclick="return sendForm(this.form);">
</form>
```

```
function sendForm(form) {
  var formData = new FormData(form);

  formData.append('secret_token', '1234567890'); // Append extra data before send.

  var xhr = new XMLHttpRequest();
  xhr.open('POST', form.action, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);

  return false; // Prevent page from submitting.
}
```

XHR2 data upload - file

An HTML form **can include file uploads** (e.g. <input type="file">) and FormData can handle that too. Simply append the file(s) and the browser will construct a **multipart/form-data request** when send() is called:

```
function uploadFiles(url, files) {
  var formData = new FormData();

  for (var i = 0, file; file = files[i]; ++i) {
    formData.append(file.name, file);
  }

  var xhr = new XMLHttpRequest();
  xhr.open('POST', url, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);  // multipart/form-data
}

document.querySelector('input[type="file"]').addEventListener('change',
function(e) {
  uploadFiles('/server', this.files);
}, false)
```

XHR2 data upload

We can also **send File or Blob data** using XHR. Keep in mind all Files are Blobs, so either works.

```
<progress min="0" max="100" value="0">0% complete</progress>
```

```
function upload(blobOrFile) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  // Listen to the upload progress.
  var progressBar = document.querySelector('progress');
  xhr.upload.onprogress = function(e) {
    if (e.lengthComputable) {
      progressBar.value = (e.loaded / e.total) * 100;
      progressBar.textContent = progressBar.value; // Fallback for unsupported browsers.
    }
  };

  xhr.send(blobOrFile);
}
```

XHR2 data upload

Last but not least, we can send **chunks of data** with **ArrayBuffers**

```
function sendArrayBuffer() {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  var uInt8Array = new Uint8Array([1, 2, 3]);

  xhr.send(uInt8Array.buffer);
}
```

XHR2 - Cross Origin Resource Sharing (CORS)

CORS allows web applications on one domain to make cross domain AJAX requests to another domain. It's dead simple to enable, only requiring a single response header to be sent by the server.

Let's say your application lives on **example.com** and you want to pull data from **www.example2.com**. Normally if you tried to make this type of AJAX call, the request would fail and the browser would throw an origin mismatch error. With CORS, www.example2.com can choose to allow requests from example.com by simply adding a header:

```
Access-Control-Allow-Origin: http://example.com
```

```
Access-Control-Allow-Origin: *
```

Check here how to enable cors: http://enable-cors.org/

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.example2.com/hello.json');
xhr.onload = function(e) {
  var data = JSON.parse(this.response);
  ...
}
xhr.send();
```

XHR2 - Practice 1

Let's say you have an **image gallery** and want to fetch a bunch of images then **save them locally** using the **HTML5 File System**. One way to accomplish this would be to request images as **Blobs** and write them out using **FileWriter**.

```
window.requestFileSystem  = window.requestFileSystem || window.webkitRequestFileSystem;

function onError(e) {
  console.log('Error', e);
}

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {

  window.requestFileSystem(TEMPORARY, 1024 * 1024, function(fs) {
    fs.root.getFile('image.png', {create: true}, function(fileEntry) {
      fileEntry.createWriter(function(writer) {

        writer.onwrite = function(e) { ... };
        writer.onerror = function(e) { ... };

        var blob = new Blob([xhr.response], {type: 'image/png'});

        writer.write(blob);

      }, onError);
    }, onError);
  }, onError);
};

xhr.send();
```

XHR2 - Practice 2

Using the File APIs, we can minimize the work **to upload a large file**. The technique is to slice the upload into **multiple chunks**, spawn an XHR for each portion, and put the file together on the server. This is similar to how **GMail uploads** large attachments so quickly. (http://localhost:8888/html5-2013/html5/44-file-upload.html)

```
function upload(blobOrFile) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };
  xhr.send(blobOrFile);
}

document.querySelector('input[type="file"]').addEventListener('change', function(e) {
  var blob = this.files[0];

  const BYTES_PER_CHUNK = 1024 * 1024;  // 1MB chunk sizes.
  const SIZE = blob.size;

  var start = 0;
  var end = BYTES_PER_CHUNK;

  while(start < SIZE) {
    upload(blob.slice(start, end));

    start = end;
    end = start + BYTES_PER_CHUNK;
  }
}, false);
```

jQuery

jQuery is a **JavaScript Library**. [http://jquery.com/].

jQuery greatly **simplifies** JavaScript programming and it is **easy to learn**.

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Resources:
http://www.w3schools.com/jquery/jquery_examples.asp [w3c samples]
http://www.w3schools.com/jquery/jquery_ref_selectors.asp [selectors]

Demos:
http://localhost:8888/jqia.source/chapter2/lab.selectors.html [selectors]
http://localhost:8888/jqia.source/chapter3/lab.move.and.copy.html [move DOM]
http://localhost:8888/jqia.source/chapter5/custom.effects.html [effects]
http://localhost:8888/jqia.source/chapter5/lab.effects.html [lab effects]
http://localhost:8888/jqia.source/chapter7/photomatic/photomatic.html [photogallery]
http://localhost:8888/jqia.source/chapter9/dimensions/lab.scroll.html [scroll]
http://localhost:8888/jqia.source/chapter9/ui/lab.draggables.html [draggables]
http://localhost:8888/jqia.source/chapter9/ui/lab.droppables.html [droppables]
http://localhost:8888/jquery_course_2013/