CORSO DI LAUREA IN INFORMATICA

PROGRAMMAZIONE WEB

JQUERY E AJAX

La funzione principale

- La funzione di base per inviare richieste AJAX è **$.ajax()** che accetta un argomento che specifica le proprietà della chiamata. Le impostazioni più comuni sono:

– **data**: dati da inviare in formato query string (**k1=v1&k2=v2**) o JSON (**{"k1":"v1", "k2":"v2"}**);

– **dataFilter(data, type)**: callback per filtrare preventivamente i dati in arrivo;

– **dataType**: tipo di dato della risposta (xml, html, script, json, jsonp, text);

– **error(XMLHttpRequest, textStatus, errorThrown)**: callback in caso di errore;

– **success(data, textStatus, XMLHttpRequest)**: callback in caso di successo;

– **type**: il tipo della richiesta (GET, POST, ...);

– **url**: l'URL a cui inviare la richiesta

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_get

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"
></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $.get("demo_test.asp", function(data, status){
            alert("Data: " + data + "\nStatus: " + status);
        });
    });
});
</script>
</head>
<body>

<button>Send an HTTP GET request to a page and get the result
back</button>

</body>
</html>
```

- https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_post

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"
></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $.post("demo_test_post.asp",
        {
          name: "Donald Duck",
          city: "Duckburg"
        },
        function(data,status){
            alert("Data: " + data + "\nStatus: " + status);
        });
    });
});
</script>
</head>
<body>

<button>Send an HTTP POST request to a page and get the result
back</button>

</body>
</html>
```

Specificare opzioni di default

- $.ajaxSetup() consente di specificare le opzioni comuni a tutte le chiamate:

```
$.ajaxSetup({ "type": "POST",
        "url": "ajax.php",
        "success": function(data) {
        $("#bar") .css("background","yellow") .html(data); }
});
```

- E poi effeJuare la chiamata:

```
$.ajax({
        "data": { "var1":"abc",
        "var2":"123" }
});
```

Load()

- The jQuery load() method is a simple, but powerful AJAX method.
- The load() method loads data from a server and puts the returned data into the selected element.
- **Syntax:**

  $(*selector*).load(*URL,data,callback*);

- The required URL parameter specifies the URL you wish to load.

Load.html (jquery-ajax-1)

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.
js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").load("demo_test.txt");
    });
});
</script>
</head>
<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>

</body>
</html>
```

Load() 2

- The optional callback parameter specifies a callback function to run when the load() method is completed. The callback function can have different parameters:
- responseTxt - contains the resulting content if the call succeeds
- statusTxt - contains the status of the call
- xhr - contains the XMLHttpRequest object
- The following example displays an alert box after the load() method completes. If the load() method has succeeded, it displays "External content loaded successfully!", and if it fails it displays an error message:

```
$("button").click(function(){
    $("#div1").load("demo_test.txt", function(responseTxt, statusTxt, xhr){
        if(statusTxt == "success")
            alert("External content loaded successfully!");
        if(statusTxt == "error")
            alert("Error: " + xhr.status + ": " + xhr.statusText);
    });
});
```

Example (AjaxServletIntegration)

Creating a complete **"toy" web application**. It is a "toy" application in the sense that it does only two things and we are using no extra features to make the environment beautiful. The purpose of the application will be simple:

- Add a band name with a list of albums (separated by commas) and press the **"Submit"** button to add them to the database.
- Press the **"Show bands!"** button to get a list of the bands, or the **"Show bands and albums!"** button to get a list of bands with their albums.

index.jsp

```
17
18      <body>
19          <h1>Ajax - Servlets Integration Example</h1>
20          <p>This is an example of how to use Ajax with a servlet
    backend.</p></br>
21
22          <h3>Select a button to get the relevant information.</h3>
23
24          <!-- Buttons that will call the servlet to retrieve the
    information. -->
25          <button id="bands" type="button">Show bands!</button>
26          <button id="bands-albums" type="button">Show bands and
    albums!</button>
27
28          <!-- We need to have some empty divs in order to add the
    retrieved information to them. -->
29          <div id="band-results"></div></br></br>
30          <div id="bands-albums-results"></div></br></br>
31
32
33          <h3>Add the band information and press submit!</h3>
34          <h4>Band name: </h4><input type="text" id="band-name-input"
    value=""><br>
35          <h4>Albums: </h4><input type="text" id="album-input"
    value="">(Separated by commas)<br>
36          <input type="submit" id="submit-band-info" value="Submit">
37      </body>
38  </html>
```

Metodi AJAX “scorciatoia”

- Per diversi usi comuni nelle applicazioni web jQuery ci mette a disposizione delle funzioni “scorciatoia” che effettuano alcune impostazioni automaticamente e poi effettuano la chiamata:
- – **$.get()** e **$.post();**
- – **$.getJSON();**
  – **$.getScript();**
  – **.load();**

$.get() e $.post()

- • Chiamata GET senza parametri:
- **$.get("ajax.php", function(data){**

  $("#bar")

  .css("background", "yellow")

  .html(data); });
- Chiamata POST con parametro:
- **$.post("ajax.php", {"var1":"abc"},**

  function(data){

  $("#bar") .css("background", "yellow") .html(data);

});

$.getJSON()

- •Supponiamo di avere una pagina json.php come segue:
- **<?php echo '{"var1":"abc", "var2":"123"}'; ?>** • Possiamo utilizzare il seguente codice per recuperare i dati in formato JSON:

```
$.getJSON("json.php", function(data){ $("#bar")
                        .css("background", "yellow")
                        .html(data.var1+", "+data.var2);
 });
```

Making a GET request using Ajax, from the frontend

- So, the first thing we need to do is find a way to ask the server for the data that we need, in this case Band names or Bands and Albums. We have already added two ids to the respective buttons ("bands" and "bands and albums") so we need to **bind an event to these in order to make a call to the server every time a button is pressed**. We will use some Javascript for that, contained in the *buttonEventsInit.js* file.

buttonEventsInit.js

```
01 // When the page is fully loaded...
02 $(document).ready(function() {
03
04     // Add an event that triggers when ANY button
05     // on the page is clicked...
06     $("button").click(function(event) {
07
08         // Get the button id, as we will pass it to the servlet
09         // using a GET request and it will be used to get different
10         // results (bands OR bands and albums).
11         var buttonID = event.target.id;
12
13         // Basic JQuery Ajax GET request. We need to pass 3
   arguments:
14         //      1. The servlet url that we will make the request
   to.
15         //      2. The GET data (in our case just the button ID).
16         //      3. A function that will be triggered as soon as the
   request is successful.
17         // Optionally, you can also chain a method that will handle
   the possibility
18         // of a failed request.
19         $.get('DBRetrievalServlet', {"button-id": buttonID},
20             function(resp) { // on sucess
21                 // We need 2 methods here due to the different ways
   of
22                 // handling a JSON object.
23                 if (buttonID === "bands")
24                     printBands(resp);
25                 else if (buttonID === "bands-albums")
26                     printBandsAndAlbums(resp);
27             })
28             .fail(function() { // on failure
29                 alert("Request failed.");
30             });
31     });
32 });
```

buttonEventsInit.js

- **As soon as the page is loaded** (we do this to be sure that all the elements are in place), we bind a click event to every button element in the page. From now on, **every time a button is clicked,** a GET request will be sent to the server with the information of which button was pressed. The server will send back the right response (in the form of a **JSON object,** we will explain later about that) and we will do different things to this object depending on the button that was pressed (because each button will receive a **differently structured** JSON object).

Processing the request and sending the data back to the client

- The app that will contain two kinds of catalogues: Bands, and Bands with albums. So we are using:
- **MusicDatabase.java:** A class that provides a **persistent object** that will contain the information that needs to be sent back to the client.
- **DBRetrievalServlet.java:** A servlet that will be used to process the **GET** request and using other classes, **provide a response** with the queried information.
- **BandWithAlbums.java:** A class that will be used to create new "data-holding objects", in our cases containing a band name, and a list of albums.

DBRetrievalServlet.java

```
@WebServlet("/DBRetrievalServlet")
public class DBRetrievalServlet extends HttpServlet {

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

        // We set a specific return type and encoding
        // in order to take advantage of the browser capabilities.
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
```

DBRetrievalServlet.java

```
        // Depending on the GET parameters, passed from the Ajax
call,
        // we are able to differentiate the requests and call the
appropriate
        // method. We can always use more classes for more use-
cases.

        // The response object returns the information (as a JSON
object in String form)
        // to the browser.
        String buttonID = request.getParameter("button-id");
        switch (buttonID) {
            case "bands":
                response.getWriter().write(MusicDatabase
                        .getInstance()
                        .getBands());
                break;

            case "bands-albums":
                response.getWriter().write(MusicDatabase
                        .getInstance()
                        .getBandsAndAlbums());
                break;
        }
    }
```

MusicDatabase.java

```
    public void setBandAndAlbums(String bandName, ArrayList
bandAlbums) {
        bandNames.add(bandName);
        bandsAndAlbums.add(new BandWithAlbums(bandName,
bandAlbums));
    }

    public String getBands() {
        return new Gson().toJson(bandNames);
    }

    public String getBandsAndAlbums() {
        return new Gson().toJson(bandsAndAlbums);
    }
}
```

MusicDatabase.java (2)

- **Gson** is a Java library from Google, which enables us to create JSON objects easily from a Java object. That object can be anything, from a simple data structure, to an objects that contains information an other data structures etc. In our case, we have 2 such datastructures:
- A List<String> bandNames, which contains only the band names in the form of Strings.
- A List<BandWithAlbums>, which contains objects, which in turn contain the band name and a list of their albums.

The **BandWithAlbums class**

```
package jsonObjects;

import java.util.ArrayList;

public class BandWithAlbums {

    String bandName;
    ArrayList bandAlbums;

    public BandWithAlbums(String bandName, ArrayList bandAlbums) {
        this.bandName = bandName;
        this.bandAlbums = bandAlbums;
    }
}
```

The **BandWithAlbums class** enables us to hold more information about a band. It is a data-holding class, which contains the band name as a String and a list of their bands as a List<String>. By returning this object, you return all the associated information as well.

```
10 function printBands(json) {
11
12     // First empty the <div> completely and add a title.
13     $("#band-results").empty()
14         .append("<h3>Band Names</h3>");
15
16     // Then add every band name contained in the list.
17     $.each(json, function(i, name) {
18         $("#band-results").append(i + 1, ". " + name + " </br>");
19     });
20 };
```

resultPrinter.js

```
function printBandsAndAlbums(json) {

    // First empty the <div> completely and add a title.
    $("#bands-albums-results").empty()
        .append("<h3>Band Names and Albums</h3>");

    // Get each band object...
    $.each(json, function(i, bandObject) {

        // Add to the <div> every band name...
        $("#bands-albums-results").append(i + 1, ". " +
bandObject.bandName + " </br>");
        // And then for every band add a list of their albums.
        $.each(bandObject.bandAlbums, function(i, album) {
            $("#bands-albums-results").append("--" + album + "
</br>");
        });
    });
};
```

Making a POST request using Ajax (insertBandInfo.js)

```
$(document).ready(function() {

    // Add an event that triggers when the submit
    // button is pressed.
    $("#submit-band-info").click(function() {

        // Get the text from the two inputs.
        var bandName = $("#band-name-input").val();
        var albumName = $("#album-input").val();

        // Fail if one of the two inputs is empty, as we need
        // both a band name and albums to make an insertion.
        if (bandName === "" || albumName === "") {
            alert("Not enough information for an insertion!");
            return;
        }

        // Ajax POST request, similar to the GET request.
        $.post('DBInsertionServlet',{"bandName": bandName,
"albumName": albumName},
            function() { // on success
                alert("Insertion successful!");
            })
            .fail(function() { //on failure
                alert("Insertion failed.");
            });
    });
});
```

Saving the user input. DBInsertionServlet.java

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

        Map<String, String[]> bandInfo = request.getParameterMap();

        // In this case here we are not using the data sent to just
do different things.
        // Instead we are using them as information to make changes
to the server,
        // in this case, adding more bands and albums.
        String bandName =
Arrays.asList(bandInfo.get("bandName")).get(0);
        String albums =
Arrays.asList(bandInfo.get("albumName")).get(0);

        MusicDatabase.getInstance()
            .setBandAndAlbums(bandName,
getAlbumNamesFromString(albums));

        // return success
        response.setStatus(200);
    }

    // Split the album String in order to get a list of albums.
    private ArrayList getAlbumNamesFromString(String albums) {
        return new ArrayList(Arrays.asList(albums.split(",")));
    }
}
```

- When the servlet gets the request, it extracts the bandName from the request map, and a String containing the album names. We create a list of Albums by **splitting the** String **into parts, when we find a comma**. In the end we call the MusicDatabase instance where we add the band name and the album list, and if you check out the class definition from before, you can see that:
- We add the band name to the bandNames list.
- We create a new Band object (using the name and the list of albums) and we add it to the bandsWithalbums list.
- After that the servlet is done and it sends a **SUCCESS status response** back to the client. We have added everything to our lists and it is ready to be sent in a JSON format when we ask for it.