

JQuery

Objectives

jQuery & AJAX

AJAX

Traditionally webpages required reloading to update their content. For web-based email this meant that users had to manually reload their inbox to check and see if they had new mail.

This had huge drawbacks: it was slow and it required user input. When the user reloaded their inbox, the server had to reconstruct the entire web page and resend all of the HTML, CSS, JavaScript, as well as the user's email.

This was hugely inefficient. Ideally, the server should only have to send the user's new messages, not the entire page.

By 2003, all the major browsers solved this issue by adopting the XMLHttpRequest (XHR) object, allowing browsers to communicate with the server without requiring a page reload.

AJAX

The XMLHttpRequest object is part of a technology called Ajax (Asynchronous JavaScript and XML).

Using Ajax, data could then be passed between the browser and the server, using the XMLHttpRequest API, without having to reload the web page.

With the widespread adoption of the XMLHttpRequest object it quickly became possible to build web applications like Google Maps, and Gmail that used XMLHttpRequest to get new map tiles, or new email without having to reload the entire page.

AJAX

Ajax requests are triggered by JavaScript code; your code sends a request to a URL, and when it receives a response, a callback function can be triggered to handle the response.

Because the request is asynchronous, the rest of your code continues to execute while the request is being processed, so it's imperative that a callback be used to handle the response.

AJAX

Unfortunately, different browsers implement the Ajax API differently. Typically this meant that developers would have to account for all the different browsers to ensure that Ajax would work universally.

Fortunately, jQuery provides Ajax support that abstracts away painful browser differences.

It offers both a full-featured `$.ajax()` method, and simple convenience methods such as `$.get()`, `$.getScript()`, `$.getJSON()`, `$.post()`, and `$.load()`.

AJAX

Most jQuery applications don't in fact use XML, despite the name "Ajax"; instead, they transport data as plain HTML or JSON (JavaScript Object Notation).

In general, Ajax does not work across domains. For instance, a webpage loaded from example1.com is unable to make an Ajax request to example2.com as it would violate the same origin policy.

As a work around, browsers have implemented a technology called Cross-Origin Resource Sharing (CORS), that allows Ajax requests to different domains.

jQuery's Ajax-Related Methods

jQuery's core `$.ajax()` method is a powerful and straightforward way of creating Ajax requests.

The `$.ajax()` method is particularly valuable because it offers the ability to specify both success and failure callbacks.

jQuery's Ajax-Related Methods

```
$.ajax({  
  // The URL for the request  
  url: "post.php",  
  // The data to send (will be converted to a query string)  
  data: {  
    id: 123  
  },  
  // Whether this is a POST or GET request  
  type: "GET",  
  // The type of data we expect back  
  dataType : "json",  
})  
// Code to run if the request succeeds (is done);  
// The response is passed to the function  
.done(function( json ) {  
  $( "<h1>" ).text( json.title ).appendTo( "body" );  
  $( "<div class=\"content\">" ).html( json.html ).appendTo( "body" );  
})
```

Contd..

jQuery's Ajax-Related Methods

Contd..

```
// Code to run if the request fails; the raw request and  
// status codes are passed to the function  
.fail(function( xhr, status, errorThrown ) {  
    alert( "Sorry, there was a problem!" );  
    console.log( "Error: " + errorThrown );  
    console.log( "Status: " + status );  
    console.dir( xhr );  
})  
// Code to run regardless of success or failure;  
.always(function( xhr, status ) {  
    alert( "The request is complete!" );  
});
```

jQuery's load() Method

The .load() method is unique among jQuery's Ajax methods in that it is called on a selection. The .load() method fetches HTML from a URL, and uses the returned HTML to populate the selected element(s).

In addition to providing a URL to the method, you can optionally provide a selector; jQuery will fetch only the matching content from the returned HTML.

```
// Using .load() to populate an element  
$( "#newContent" ).load( "/foo.html" );
```

```
// Using .load() to populate an element based on a selector  
$( "#newContent" ).load( "/foo.html #myDiv h1:first", function( html ) {  
    alert( "Content updated!" );  
});
```

jQuery's load() Method Example

```
<body>
<div id="divTestArea1"></div>

<script type="text/javascript">
    $(function()
    {
        $("#divTestArea1").load("content.html");
    });
</script>
<br>

<div id="divTestArea2"></div>

<script type="text/javascript">
    $(function()
    {
        $("#divTestArea2").load("content.html #divContent");
    });
</script>
</body>
```

jQuery's get() and post() Methods Examples

If you don't need the extensive configurability of \$.ajax(), and you don't care about handling errors, the Ajax convenience functions provided by jQuery can be useful, terse ways to accomplish Ajax requests.

These methods are just "wrappers" around the core \$.ajax() method

```
// Get plain text or HTML
$.get( "/users.php", {
  userId: 1234
}, function( resp ) {
  console.log( resp ); // server response
});
// Add a script to the page, then run a function defined in it
$.getScript( "/static/js/myScript.js", function() {
  functionFromMyScript();
});
// Get JSON-formatted data from the server
$.getJSON( "/details.php", function( resp ) {
  // Log each key in the response data
  $.each( resp, function( key, value ) {
    console.log( key + " : " + value );
  });
});
```

AJAX and Forms

jQuery's ajax capabilities can be especially useful when dealing with forms.

There are several advantages, which can range from serialization, to simple client-side validation (e.g. "Sorry, that username is taken"), to [prefilters](#)

Serialization

Serialization

Serializing form inputs in jQuery is extremely easy. Two methods come supported natively: `.serialize()` and `.serializeArray()`.

The `.serialize()` method serializes a form's data into a query string. For the element's value to be serialized, it **must** have a name attribute.

Please note that values from inputs with a type of checkbox or radio are included only if they are checked.

Turning form data into a query string

```
$( "#myForm" ).serialize();
```

// Creates a query string like this:

`field_1=something&field2=somethingElse`

Serialization

serializeArray() method is similar to the `.serialize()` method, except it produces an array of objects, instead of a string.

/Creating an array of objects containing form data

```
$( "#myForm" ).serializeArray();
```

// Creates a structure like this:

```
[  
  {  
    name : "field_1",  
    value : "something"  
  },  
  {  
    name : "field_2",  
    value : "somethingElse"  
  }  
]
```


jQuery AJAX Form Submit Example

For jQuery Ajax Form Submit, we can use **jQuery.ajax()** or **jQuery.post()** method.

To serialize form data, we can use **jQuery.serialize()** or **jQuery.serializeArray()**.

```
<form name="ajaxform" id="ajaxform" action="ajax-form-submit.jsp"
method="POST">
    First Name: <input type="text" name="fname" value =""/> <br/>
    Last Name: <input type="text" name="lname" value ="" /> <br/>
    Email : <input type="text" name="email" value=""/> <br/>
</form>
```

jQuery AJAX Form Submit Example

```
//callback handler for form submit
$("#ajaxform").submit(function(e){
    var postData = $(this).serializeArray();
    var formURL = $(this).attr("action");
    $.ajax(
    {
        url : formURL,
        type: "POST",
        data : postData,
        success:function(data, textStatus, jqXHR) {
            //data: return data from server
        },
        error: function(jqXHR, textStatus, errorThrown) {
            //if fails
        }
    });
    e.preventDefault(); //STOP default action
    e.unbind(); //unbind. to stop multiple form submit.
});
$("#ajaxform").submit(); //Submit the FORM
```

Form data submission example

jquery_serialize.html

```
<!DOCTYPE html>
<html><head>
<title>New document</title>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
<script>
    $(document).ready(function(){
        $("button").click(function(){
            $("div").text($("#form").serialize());
        });
    });
</script>
</head>
<body>
<form action="">
    First name: <input type="text" name="FirstName" value="Mickey"><br>
    Last name: <input type="text" name="LastName" value="Mouse"><br>
</form>

<button>Serialize form values</button>

<div></div>

</body></html>
```

The jQuery get() example

```
<!DOCTYPE html>
<html><head>
<title>New document</title>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
</head>
<body>
<script type="text/javascript">
    $(function()
    {
        $.get("content.html", function(data, textStatus)
        {
            alert("Done, with the following status: " + textStatus + ". Here is the response: " + data);
        });
    });
</script>
</body>
</html>
```

serializeArray() example

jquery_serializeArray.html

```
<!DOCTYPE html>
<html><head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        var x = $("form").serializeArray();
        $.each(x, function(i, field){
            $("#results").append(field.name + ":" + field.value + " ");
        });
    });
});
</script>
</head>
<body>
<form action="">
    First name: <input type="text" name="FirstName" value="Mickey"><br>
    Last name: <input type="text" name="LastName" value="Mouse"><br>
</form>
<button>Serialize form values</button>
<div id="results"></div>
</body>
</html>
```

About JSON

What is JSON?

- [JSON](#) stands for JavaScript Object Notation. In simple terms JSON is a way of formatting data for, e.g., transmitting it over a network.
- Why would we choose JSON over say XML? The key advantage of using JSON is efficiency. JSON is less verbose and cluttered, resulting in fewer bytes and a faster parse process.
- This allows us to process more messages sent as JSON than as XML.

Getting JSON data

The `$.getJSON()` method is a handy helper for working with JSON directly if you don't require much extra configuration.

`$.getJSON()` syntax

```
$.getJSON(url, data, success);
```

Same as

```
$.ajax({  
  url: url,  
  dataType: 'json', //json data type  
  data: data,  
  success: callback,  
  error: callback  
});
```

Besides the required URL parameter we can pass in two optional parameters. One represents the data to send to the server, the other one a callback to trigger in case of a successful response.

So the three parameters correspond to:

- 1.The url parameter is a string containing the URL to which the request is sent.
- 2.The optional data parameter is either an object or a string that is sent to the server with the request.
- 3.The optional SUCCESS(data, textStatus, jqXHR) parameter is a callback function executed only if the request succeeds.

Getting JSON data Example

```
<!DOCTYPE html>
<html><head>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
<script type = "text/javascript">
```

```
$(document).ready(function() {
    $("#driver").click(function(event){

        $.getJSON("result.json", function(jd) {
            $('#stage').html('<p> Name: ' + jd.name + '</p>');
            $('#stage').append('<p>Age : ' + jd.age+ '</p>');
            $('#stage').append('<p> Sex: ' + jd.sex+ '</p>');
        });
    });
});
```

result.json

```
{
  "name": "Srinivas Reddy",
  "age" : "49",
  "sex": "male"
}
```

```
</script>
</head>
<body>
```

<<p>Click on the button to load result.json file –</p>
<div id = "stage" style = "background-color:#eee;">
STAGE
</div>

<input type = "button" id = "driver" value = "Load Data">
</body></html>

Another JSON data Example

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
</head>
<body>
<a name="#ajax-getjson-example"></a>
<div id="div1">
<h2>Car sale report</h2><br>
<div id="div2"></div><br>
<button id="btn1" type="button">Click to load car sale report (json type)</button>
<br><br>
<div>
<a href="car-sale.json" target="_blank">Original car sale report</a><br>
</div>

</body>
</html>
```

car-sale.json

car-sale.json

























```
[  
  { "Manufacturer": "Toyota", "Sold": 1200, "Month": "2012-11" },  
  { "Manufacturer": "Ford", "Sold": 1100, "Month": "2012-11" },  
  { "Manufacturer": "BMW", "Sold": 900, "Month": "2012-11" },  
  { "Manufacturer": "Benz", "Sold": 600, "Month": "2012-11" },  
  { "Manufacturer": "GMC", "Sold": 500, "Month": "2012-11" },  
  { "Manufacturer": "HUMMER", "Sold": 120, "Month": "2012-11" }  
]
```

Another JSON data Example Contd..

```
<script type="text/javascript">
$(document).ready(function (){
    $("#btn1").click(function(){

        $.getJSON("car-sale.json", function(data){
            var html = [];
            /* loop through array */
            $.each(data, function(index, d){
                html.push("Manufacturer : ", d.Manufacturer, ", ",
                    "Sold : ", d.Sold, ", ",
                    "Month : ", d.Month, "<br>");
            });
            $("#div2").html(html.join("")).css("background-color", "orange");
        }).error(function(jqXHR, textStatus, errorThrown){ /* assign handler */
            alert("error occurred!");
        });
    });
});
</script>
```

Handling JSON Data : Web Application

- ▼  HandlingJsonJQuery
 - >  Deployment Descriptor: HandlingJsonJQue
 - >  JAX-WS Web Services
 - ▼  Java Resources
 - ▼  src
 - ▼  com.digitech.business tier
 - >  MovieManager.java
 - >  MovieTO.java
 - ▼  com.digitech.persistence tier
 - >  IMovie.java
 - >  MovieService.java
 - ▼  com.digitech.utility
 - >  MySQLConnection.java
 - ▼  com.digitech.web tier
 - >  GetMovies.java
 - >  InsertMovie.java
 - >  Libraries
 - >  JavaScript Resources
 - >  build
 - ▼  WebContent
 - >  META-INF
 - >  WEB-INF
 -  insert_movie.xhtml
 -  movies_list.html

```
create table movie(  
  movie_id int not null primary key AUTO_INCREMENT ,  
  movie_name longtext  
);
```

Handling JSON Data : Web Application

```
public class MySqlConnection {  
    static{  
        try {  
            Class.forName("com.mysql.jdbc.Driver");  
        } catch (ClassNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
  
    public static Connection getConnection() throws SQLException{  
        return  
        DriverManager.getConnection("jdbc:mysql://localhost:3306/test",  
            "root", "root");  
    }  
}
```

Handling JSON Data : Web Application

```
public interface IMovie {  
    public String insertMovie(MovieTO movieTO);  
    //public MovieTO getMovieById(Integer id);  
    public List<MovieTO> getAllMovies();  
}
```

Handling JSON Data : Web Application contd..

```
public class MovieService implements IMovie{
    @Override
    public String insertMovie(MovieTO movieTO) {
        String sql="insert into movie(movie_name) values(?)";
        try(Connection connection=MySQLConnection.getConnection();
            PreparedStatement
            preparedStatement=connection.prepareStatement(sql)
        ){
            preparedStatement.setString(1, movieTO.getMovie_name());
            int n=preparedStatement.executeUpdate();
            if(n>0){
                return "SUCCESS";
            }else{
                return "FAILED";
            }
        }
        catch(SQLException e){
            e.printStackTrace();
        }
        catch(Exception e){e.printStackTrace();}
        return null;
    }
}
```

Handling JSON Data : Web Application contd..

@Override

```
public List<MovieTO> getAllMovies() {  
String sql="select * from movie";  
try(Connection connection=MySQLConnection.getConnection());  
Statement statement=connection.createStatement();  
{  
ResultSet resultSet=statement.executeQuery(sql);  
List<MovieTO> movieList=new ArrayList<>();  
while(resultSet.next()){  
MovieTO movieTO=new MovieTO();  
populateMovieObject(resultSet,movieTO);  
movieList.add(movieTO);  
}  
return movieList;  
}  
catch(SQLException e){  
e.printStackTrace();  
catch(Exception e){  
e.printStackTrace();  
}  
return null;
```

```
private void populateMovieObject(ResultSet resultSet,  
MovieTO movieTO) throws SQLException {  
movieTO.setMovie_id(resultSet.getInt("movie_id"));  
movieTO.setMovie_name(resultSet.getString("movie_n  
ame"));  
}  
}
```


Handling JSON Data : Web Application contd..

```
public class MovieTO {  
    private Integer movie_id;  
    private String movie_name;  
  
    public Integer getMovie_id() {  
        return movie_id;  
    }  
    public void setMovie_id(Integer movie_id) {  
        this.movie_id = movie_id;  
    }  
    public String getMovie_name() {  
        return movie_name;  
    }  
    public void setMovie_name(String movie_name) {  
        this.movie_name = movie_name;  
    }  
}
```

Handling JSON Data : Web Application contd..

```
public class MovieManager {  
  
    public List<MovieTO> getAllMovies() {  
        MovieService service=new MovieService();  
        return service.getAllMovies();  
    }  
  
    public String insertMovie(MovieTO movieTO) {  
        MovieService service=new MovieService();  
        return service.insertMovie(movieTO);  
    }  
}
```

Handling JSON Data : Web Application contd..

```
@WebServlet("/InsertMovie")
```

```
public class InsertMovie extends HttpServlet {
```

```
private static final long serialVersionUID = 1L;
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```

```
    PrintWriter out=response.getWriter();
```

```
    String message=request.getParameter("message");
```

```
if(message.length()>0){
```

```
        MovieTO movieTO=new MovieTO();
```

```
        movieTO.setMovie_name(message);
```

```
        MovieManager manager=new MovieManager();
```

```
        String result=manager.insertMovie(movieTO);
```

```
        out.println("<html><body>"+result+"</body></html>");
```

```
        System.out.println(result);
```

```
}else{
```

```
        out.println("<html><body>"+ "Enter Movie Name"+"</body></html>");
```

```
}
```

```
}
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {doGet(request, response);}}
```

Handling JSON Data : Web Application contd..

```
@WebServlet("/GetMovies")
```

```
public class GetMovies extends HttpServlet {
```

```
private static final long serialVersionUID = 1L;
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {
```

```
    PrintWriter out=response.getWriter();
```

```
    MovieManager manager=new MovieManager();
```

```
    List<MovieTO> movieList=manager.getAllMovies();
```

```
    if(movieList.size() != 0){
```

```
        Gson gson = new Gson();
```

```
        String movies= gson.toJson(movieList);
```

```
        out.println("{\"MoviesList\":\""+movies+"\"}");
```

```
    }else{
```

```
        out.println("<html><body<h1>"+ "No Records Found" + "</h1></body></html>");
```

```
    }
```

```
}
```

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

```
    doGet(request, response);
```

```
}}
```

Note:

GSON is Google's JSON parser and generator for Java. Google developed GSON for internal use but open sourced it later

Handling JSON Data : Web Application contd..

```
$(document).ready(function() {  
    $('#InsertButton').click(function() {  
        var MSG = $("#Message").val();  
        var dataString = 'message='+ MSG;  
        $.ajax({  
            type: "POST",  
            url: "InsertMovie",  
            data: dataString,  
            cache: false,  
            success: function(data) {  
                $("#Message").val("");  
                $("#content").append(data)  
                $("#Message").focus();  
            }  
        });  
        return false;  
    });  
}); </script>
```

insert_movie.xhtml

```
<body>  
<textarea id='Message'></textarea><br/>  
<input type='button' value='Insert' id='InsertButton'/>  
<div id='content'></div>  
</body>
```

Handling JSON Data : Web Application contd..

movies_list.html

```
$(document).ready(function() {  
    $.ajax ({  
        type: "GET",  
        url: "GetMovies",  
        dataType:"json",  
        success: function(data) {  
            if(data.MoviesList.length) {  
                $.each(data.MoviesList, function(index,data) {  
                    var msg_data=  
                        "<div id='msg'+data.movie_id+'>"+data.movie_name+"</div>";  
                    $(msg_data).appendTo("#content");  
                });  
            }  
            else {  
                <  
                $("#content").html("No Results");  
            }  
        }  
    });  
    return false;  
});
```

```
<body>  
    <div id='content'></div>  
</body>
```



Thank You!