Javascript tutorial

Rui Couto José C. Campos

Sistemas Interactivos
Mestrado Integrado em Engenharia Informática
DI/UMinho
March 17, 2020

Contents

1 Introduction

The previous tutorials provided an overview about creating the content, structure and style of a Web page. Content, however, was added statically. This tutorial presents the required steps to fill the table content dynamically.

This tutorial addresses the behavioural aspect of a Web page. It illustrates how to use Javascript to interact with a Web server and manipulate the DOM[1]. The usage of jQuery is also addressed.

2 Base template

The starting point is the Web page resulting from the previous (Bootstrap) tutorial[2]. Most of the code will be placed in the `script.js` and `index.html` files.

3 Javascript basics

Contrary to HTML/CSS, Javascript is an imperative programming language. Javascript scripts correspond to code and functions, which are invoked in response to events in the page (e.g. when the page is loaded or when a user interacts with some control).

Javascript's syntax is inspired by languages such as Java and C. Blocks of code are defined by "`{ }`", and "`;`" is used to end statements. Javascript features the usual control structure: `if`, `while`, `do/while`, `for` and `switch`. The `for/in` statement loops through the properties of an object. Javascript is dynamically typed. What this means is that the types of variables are generally not known at compile time. See Section 4 for more on variables.

In order to check that everything is working as expected, some minimal examples can be put to work. The first proposed test is to create a popup message, using the `alert()` function (c.f. Figure 1), and the second is to print to the console, using `console.log()` (c.f. Figure 2).

You can do the above by placing the Javascript code inside a `<script>` tag on the HTML file. A better alternative is to place the code in external "`.js`" files (e.g. `script.js`) and load those into the page. To load `script.js`, the declaration `<script src="scrip.js"/>` can be used.

The code placed in the `script.js` file will load once the browser loads the page referencing it. Code inside a function is only executed when the function is called. Code outside function definitions is automatically run. Thus, if you call the `alert()` method at the beginning of the `script.js` file, a popup window will

[1] DOM – Document Object Model. See: `https://www.w3schools.com/whatis/whatis_htmldom.asp`

[2] A basic working version is provided in case you have not completed the previous tutorial.

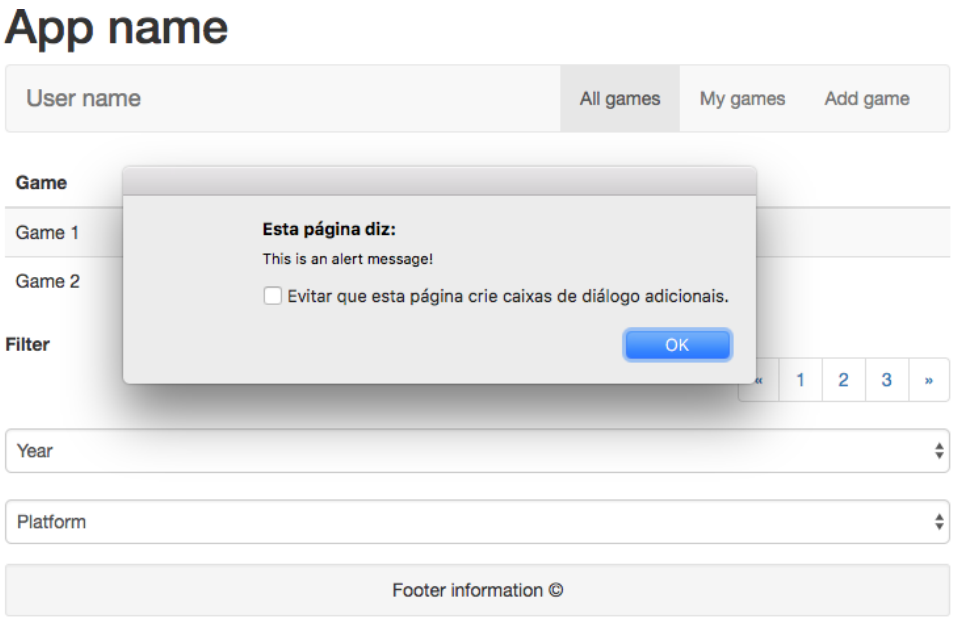

Figure 1: Output via an alert message.

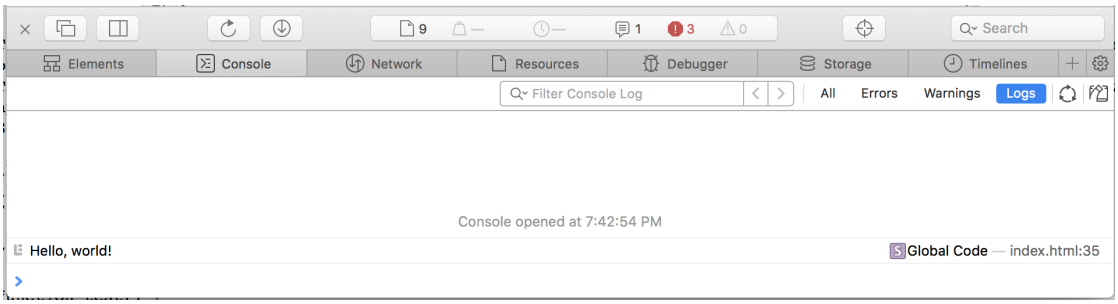

Figure 2: Output via the console.

automatically appear when the file is loaded. See the following example:

```
//script.js

//code here will run automatically

function test() {
  //this code will run when the function is invoked
}
```

Modern browsers have developer tools to help in the development process. For instance in Google Chrome, Mozilla Firefox or Safari simply right-click in a page, and select inspect element. Such will show a new window or frame, where

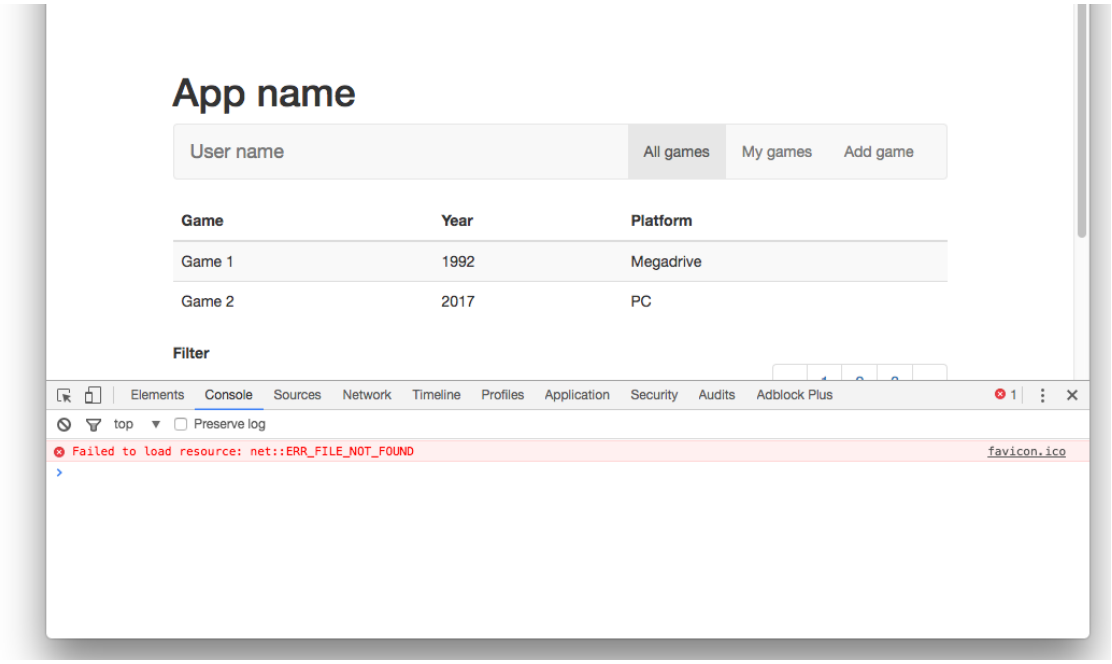

Figure 3: Javascript console with an error message.

the console is accessible. This console will show errors, output messages and allow you to interact with the browser (using Javascript). Figure 2 shows the Safari console displaying log messages. Figure 3 shows the Google Chrome console, with an error message indicating a missing file.

Tasks

1. Create a popup message with "Hello, world!" as content. This popup should appear once the page is loaded.
2. Create a function `sayHello()` which prints to the terminal "Hello, world!", and invoke it through the console.
3. Confirm that everything is running as expected, by checking the produced outputs.

4 Javascript objects and classes

Javascript has the concept of object. Objects can be defined dynamically. For example, the following code defines an (empty) object and adds two fields (name and year) to it:

```
var game = {};
game.name = "Oxenfree";
game.year = 2017;
```

Javascript also supports the notion of classes and, as of ECMAScript 2015, a class can be defined as follows[3]:

```
class Game {
     //constructor
    constructor() {
        this._name = "Oxenfree";
        this._year = 2017;
    }

    //getter for _nome
    get name() {
        return this._name;
    }

    //setter for _nome
    set name(v) {
        this._name = v;
    }

    //example method
    toString() {
         return "Games(" + this._name + ", " + this._year + ")";
    }
}
```

Note that object properties (instance attributes in "Java *speak*") are not explicitly declared as part of the class definition. They are dynamically created when assigned some value in the constructor or methods (you should follow the convention to assign – i.e. *create* – all properties in the constructor). The keyword `this` is used to reference the object. As all other Javascript variables, instance attributes are dynamically typed (i.e., they have the type of the value they hold at each moment).

Notice the convention used to indicate private variables (the use of underscore), and the definition of getters and setters to access them (this is **just a convention** – `user.name` uses the getter method, but writing "`user._name`" in the example below would not trigger any error)[4].

[3] Browser support will vary – IE and Opera Mini do not support this at the moment.

[4] Note: the use of the # prefix to declare private fields has been proposed for ECMAScript 2019.

Using a class in Javascript is similar to Java. See the following example:

```
var user = new Game();
console.log(user.name);
console.log(user.toString());
```

Note the `var` keyword to declare the variable. This make the variable have the scope of the current code block. Variables are implicitly *defined* the first type they are assigned. The use of `var` is optional (but **highly recommended**). Without the `var` keyword the scope automatically becomes global (i.e. the variables become global variables).

Tasks

4. Complete the class `Game` with the platform property, and add a method to check if a game is more recent than another (i.e. `game1.moreRecent(game2)`). Write your code in `scripts/game.js`.

5 Manipulating the DOM

5.1 Plain Javascript

Javascript has the capability of manipulating the DOM of the Web pages. This makes it possible to access the page elements, and access or modify their contents, appearance and even behaviour. When writing code for a Web page, we are in an event-based programming model. The browser controls execution and we provide event handlers to be executed when relevant events occur. We can think of event event handlers in terms of two general types:

- even handlers used to react to browser events (e.g., a page has finished loading);
- even handlers used to react to user actions on the page (e.g., a button has been clicked).

Manipulating the DOM consists of three steps:

First define which event[5] will trigger the behaviour. In order to achieve this, it is possible to specify, in the HTML elements, which Javascript functions are associated with which events. E.g., to invoke a function when a mouse hovers some `<div>` we can write:

```
<div onmouseover="myFunction()">over here</div>
```

[5] List of possible events `https://www.w3schools.com/jsref/dom_obj_event.asp`

Second in the function, it is necessary to retrive from the DOM the element(s) of the page that will be accessed. This requires using DOM functions such as `document.getElementById()`[6]:

```
var elem = document.getElementById("myDiv");
```

Third the properties and functions of the retrieved element can then be used to achieve the desired effect[7]. For instance, modifying style properties can be done through the `style` property[8]:

```
elem.style.color = "blue";
```

Through this series of steps, it is possible to manipulate the page.

Tasks

An example is proposed, in order to understand the principles of DOM manipulation. The example is a counter of the number of clicks in a button.

5. In the HTML file, create the button to be clicked and a `<div>` where the number of clicks will be displayed. Give the `<div>` some `id` (e.g. `myId`).
6. In the `script.js` file:
   (a) Define a variable to act as counter and initialise it to zero[9].
   (b) Create a function which sets the contents of the just created `<div>` to the value of the counter (you can achieve this by setting the `innerHTML` property of the `<div>`) and increments it.
   (c) Update the contents of the `<div>` to 0 (by invoking the function).
7. Back in the HTML, associate the `onClick` event of the button with your function.
   (a) do it in the HTML file, as explained above;
   (b) explore how to do it in the Javascript file instead.
8. Load the page in a browser and click the button multiple times, observing the effect on the `<div>`.

[6] See the page `https://www.w3schools.com/jsref/dom_obj_document.asp` for more information on the attributes and API of the `document` object.

[7] See the page `https://www.w3schools.com/jsref/dom_obj_all.asp` for more information on DOM nodes' properties and API.

[8] See the page `https://www.w3schools.com/js/js_htmldom_html.asp` for more information on how to modify HTML.

[9] It is possible to avoid using a variable by working directly with the contents of the `<div>` (you can try it), however, this would be less MVC-like.

5.2 jQuery

Javascript can became very verbose and hard to read/maintain. jQuery is a Javascript library which eases programming. jQuery greatly reduces the number of lines of code written and standardises how code is written. A typical jQuery expression selects some elements in the DOM and applies one or more functions on them[10]:

```
1 $(selector).action();
```

jQuery selectors follow the CSS syntax. To act on the elements, jQuery provides a set of unified methods[11] (e.g., `html()` for HTML, `css()` for CSS, `text()` for the text contents, etc.). When used without parameters they act as getters, and when used with parameters, act as setters. Hence, to find an element given its id, and set its colour to blue, we write[12]:

```
1 $("#myDiv").css("color","blue");
```

Note that in the following example the selector will return a collection of elements (all `<h1>` elements in the page). In that case the method is applied to every element of the collection. Note also how, in the example below, methods are chained (this gives JQuery a functional flavour).

```
1 $("h1").css("color","blue").prepend("Capitulo - ");
```

Tasks

9. Modify the click counter to use jQuery instead of pure Javascript.

6 Asynchronous requests

When clicking on an anchor (a link) a synchronous request if made by the browser to the web server to obtain the content of the document pointed at by the anchor. This inevitably implies a page refresh to display the new contents.

Asynchronous requests (Ajax) allow the pages to dynamically load content from the web server without the need to reload the whole page. jQuery greatly eases the process of loading retrieved remote information. Specifically, the `$.get()` method supports performing requests and handle responses. For this tutorial, the webservice located at `http://ivy.di.uminho.pt:8080/GamesLibraryProvider/` will be considered.

[10] A parallel can be made between jQuery and Java 8's internal iterators.

[11] See the API at `https://api.jquery.com`.

[12] Not only is this more concise than the plain Javascript solution presented earlier, but if a class is used instead of an id, then the colour is changed for all elements of that class!

A sample list of games is provided at the URL: `http://ivy.di.uminho.pt:8080/GamesLibraryProvider/GamesService?action=list`. The following example shows how to retrieve the information and display it locally:

```
$.get("http://ivy.di.uminho.pt:8080/GamesLibraryProvider/GamesService?action=list")
 .done(function(data) {
         $("#myDiv").html(data);
      });
```

This code will load the content available at the provided URL, and once the content is loaded, it will execute the code inside `done()`. In this case, it will look for a tag with ID `myDiv` and set the retrieved information as its content.

Listing all the games is one of the two services provided. If you visit `http://ivy.di.uminho.pt:8080/GamesLibraryProvider/` you will see it shows how to access the available services: list all games, and access the details of a particular game. Hence, to access the details of the "GTA 20V" game, the URL `http://ivy.di.uminho.pt:8080/GamesLibraryProvider/GamesService?action=get&value=GTA%20V` should be used.

Tasks

10. Add a div at the bottom of the page you created for the last tutorial. Give it the name `"myRawData"`

11. Create a `loadGames()` function to load the list of games, and display it in the `"myRawData"` `<div>`. Ignore its format for now. You should define the function in a external `.js` and load it at the end of the HTML file with a `<script>` tag.

12. Setup the function to be run once the page is loaded. There are two alternatives: associate it with the `onLoad` event of `<body>`; associate it with jQuery's ready event. In the first case, the function will execute once the page is fully loaded (including images, etc.). In the second case, it will run as soon as the HTML has been loaded.

    The recommended approach to define a callback to JQuery's `ready` event is (add the code to your `.js` file)[13]:

    ```
    $(function() {
        loadGames();
    });
    ```

    Note the use of an anonymous function. This makes it easier to add further behaviour to the ready event.

[13] A now deprecated alternative (but one which made the event more visible) was: `$().ready(function() { loadGames();});`

6.1 JSON

JSON (JavaScript Object Notation) is a popular format to exchange data in REST formats, and can be easily interpreted and manipulated by Javascript. JSON is a textual format which corresponds to key-value pairs. An example is as follows:

```
{
    "user": "username",
    "age": 10,
    "ids": ["1", "2", "3"],
    "contact": {
        "address": "street ...",
        "number": "n"
    }
}
```

The JSON[14] is derived from Javascript so JSON objects are defined inside `{ }`. Data is written as pairs of name/value using the syntax `"name":"value"`. Names must be strings). Value can be strings, integers, arrays (c.f. `ids` above) or even another object (e.g. `contact` above).

In order to validate the syntax of a JSON object some tools can be used. The website `http://jsonlint.com/` is one example.

Converting a JSON string to a Javascript object is straightforward. The JSON class (part of Javascript) provides the method `parse`[15]. This method parses the text and creates an object with the corresponding structure. Next follows an example on how to parse and use JSON.

```
var json = "{\"name\":\"a user\",\"age\":\"20\"}";
var u = JSON.parse(json);
console.log(u.name);
```

Note that the object `u` is dynamically created, and its attributes automatically set. The inverse process can be achieved with the method `JSON.stringify()`.

6.2 Putting the pieces together

The list of games (previously retrieved from the webservice) is provided in JSON. At this stage, all requirements are gathered to process such information. It is now possible to convert the JSON to the corresponding object, and through DOM manipulation fill the table content[16].

[14] See the following URL for more details `http://www.json.org/`

[15] See the following URL for more details `https://www.w3schools.com/js/js_json_parse.asp`

[16] See the following URL for more information on how to dynamically create a table `https://www.w3schools.com/jsref/met_table_insertrow.asp`

Tasks

Go back to the `loadGames()` function and:

13. Parse the information retrieved from the web service to an object.
14. Iterate the object[17], and dynamically fill the table with the game data (the static information is no longer required). Explore JQuery's API for methods to append rows to the table.

[17] See `https://api.jquery.com/jQuery.each/`. Look also at `https://www.w3schools.com/js/js_loop_for.asp` and `https://www.w3schools.com/jsref/jsref_forEach.asp` for different ways to iterate a collection in Javascript.