Introduction to Web Programming

Chap. 6 / jQuery

Anne Jeannin-Girardon, PhD | anne.jeannin@unistra.fr
Associate Professor, University of Strasbourg

What is jQuery ?

“jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.”

https://jquery.com/

- jQuery is javascript library (a script) you include in your web page

Loading jQuery from a CDN

- CDN: Content Delivery Network
- Resources located on web servers, for instance at Google's
  - "The Google Hosted Libraries is a stable, reliable, high-speed, globally available content distribution network for the most popular, open-source JavaScript libraries." https://developers.google.com/speed/libraries/
- Just embed the following snippet into your page :

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"> </script>
```

Why use a CDN ?

- Maybe the script is already in the cache of the user (loaded previously from the CDN during an other browsing session)
- No need to download updates : just change the include snippet
- CDN servers are very likely to be highly performant (google, microsoft, ...)

Syntax

Define actions on selectors (CSS selectors) - the two following instructions are strictly identical :

```
jQuery(selector).action();
```

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

In natural language : “ask jQuery to select the element(s) selector and perform the action action on this(ese) element(s)”

First example

```
$("a").click(
    function() {
      alert("You clicked on " + this.href);
    }
);
```

- `$("a")` : select *all* a elements in the document
- When the event **`click`** is detected, execute the function given as parameter (between **`()`**)
- Here, it opens an alert window displaying the href value of <u>the element that was clicked</u> (**`this`**)
- Note : here, the function is anonymous (it has no name)

Where should you write your jQuery code ?

- In your regular javascript script
- Just remember to wrap it in a dedicated section, as shown here
- (It's just to make sure your document is loaded before you start doing stuff with it)

jQuery

```
/* this is your “regular” script */

var v = "Hello world!";

$(document).ready(function(){
  // Write jQuery code here
});
```

jQuery selectors

Basic selectors

`*` all elements

`element` elements with that name

`#id` element with the id `id` specified

`.class` elements with the class `class` specified

There are more !
https://www.w3schools.com/jquery/jquery_selectors.asp

Hierarchical selectors

`ancestor descendant` all descendant that are children of ancestor

`parent > child` direct child of elt parent

`previous + next` elements immediately followed by previous elt

`previous ~ siblings` all elements that are siblings of previous

jQuery events

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

```
$(".myClass").dblclick(function(){
    $(this).hide();
});
```

this, again

```
$(".myClass").dblclick(function(){
    $(this).hide();
});
```

- `$(".myClass")` : select *all* elements having the class myClass
- You want a given behavior to possibly apply to a range of elements…
- … but not all at once ! only the one concerned by the event (here, the element you double-clicked)

Apply effects with jQuery

- Hide / display elements
  - Without effects : `hide()`, `show()`, `toggle()`
  - Fade : `fadeOut()`, `fadeIn()`, `fadeToggle()`
  - Slide : `slideUp()`, `slideDown()`, `slideToggle()`
  - General syntax : `$("selector").effect(speed, callback);`
- Animate elements
  - `animate({parameters}, speed, callback)`
  - Parameter example : `left: 200px`

What's a callback ?

- A callback is a function that is called after an effect is completely finished

An example is way more explanatory :

Sans callback

```
$("#element").hide(1000);
alert("The element is hidden");
```

Avec callback

```
$("#element").hide(1000, function() {
    alert("The element is hidden");
});
```

Animation cascade

```
$('#myElement').slideUp(1000).slideDown(1000);
```

Iterate over selected elements

```
$('.box').each(function(){
  $(this).action(param);
});
```

Using jQuery to play with the DOM

- Elements' content and attributes
- Element's CSS
- Inserting, removing elements

Get content from elements

- `$("#elem").text()`
  returns the textual content of the element `#elem`
- `$("#elem").html()`
  returns the HTML content of the element `#elem`
- `$("#field").val()`
  returns the value set in the form field `#field`

Get content from an element we clicked

```
$('.pickMe').click(function(){
  let content = $(this).text();
  console.log(content);
});
```

Set elements content

- `$("#elem").text("some text")`
  sets the textual content of the element `#elem`
- `$("#elem").html("some <em>text</em>")`
  sets the HTML content of the element `#elem`
- `$("#field").val("some text")`
  sets the value set in the form field `#field`
- The methods are the same (same name) but have a different signature (they have arguments) => this is called method overload

What about element attributes ?

```
$('#buttonAttr').click(function(){
  $('#image-1').attr('src', 'img/noidea.jpg');
  $('#image-1').attr('alt', 'Studying sloth');
  /*
  $('#image-1').attr({'src': 'img/noidea.jpg' ,
                      'alt':'Studying sloth'});
  */
});
```

And CSS properties ?

```
$('#element').css('color', 'white');

// Or

$('#element').css({'color' : 'white', 'font-size' : '0.9em'});
```

More about CSS

```
$('#buttonAddCSS').click(function(){
  $('#content-7').addClass('class2');
});

$('#buttonDelCSS').click(function(){
  $('#content-7').removeClass('class1');
});

$('#buttonTogCSS').click(function(){
  $('#content-7').toggleClass('class1');
});
```

Setting content using callbacks

The callback functions has 2 arguments : the index of the element and the original content (both are set automatically)

```
$('#buttonSetWithCallback').click(function(){
  $('.quote').text(function(elem_index, prev_content){
    return "Element " + elem_index
          + " / Prev. content : "
          + prev_content;
  });
});
```

Adding elements

```
var elt = $('<p id="content-2"></p>').text('So easy !')
$('#content-1').append(elt); // append elt after #content-1

// Available methods : append(), prepend(), after(), before()
```

Removing elements

```
$('#element').remove();

// We can also filter what we remove,
// e.g. remove all p having the class myClass
$('p').remove('.myClass');
```

Traversing the DOM : parents

```
// Direct parent
var parent  = $('#element').parent();

// All the parents
var parents = $('#element').parents();

// All parents until #anotherElt
var grandparents = $('#element').parentsUntil($('#anotherElt'));
```

Traversing the DOM : descendants

```
// Direct children (possibly filtered using a selector as argument)
var directChildren = $('#element').children();

// All children (at any level of the DOM tree below #element)
// matching the selector given as argument
var allChildren    = $('#element').find($('em'));
```

Traversing the DOM : siblings

```
// All siblings at the same level (possibly filtered
// using a selector as argument)
var siblings = $('#element').siblings();

// Direct next sibling
var directSibling = $('#element').next();

// All next siblings
var nextSiblings = $('#element').nextAll();

// All next sibling until #anotherElt
var nextUntilSiblings = $('#element').nextUntil('#anotherElt');

// Go backward in the DOM with prev(), prevAll() and prevUntil()
```

Traversing the DOM : filters

```
// First p element contained in #element
var first = $('#element p').first();

// Last p element contained in #element
var last = $('#element p').last();

// Returns the p element indexed 1 (counting from 0)
var par = $('p').eq(1);

// Returns all p elements having class myClass
var par2 = $('p').filter('.myClass');

// Returns all p elements _not_ having class myClass
var par3 = $('p').not('.myClass');
```

Local storage & JSON

- What is the local storage ?
- Formatting data with JSON

“Cookies can crumble”

- Cookies are small bits of information, sent by a server and stored in the browser
- They are used to remember states
  - Shopping carts
  - Browsing activity (clicked that particular button)
  - Logins
  - …
- Cookies are used by the server (it can read them)
- A cookie cannot exceed 4KB

Local & Session storage

- Storage capacity : 5 MB
- Only available on the client side
- 2 types of storage
  - Local storage : persistent (data are still here when the browser restarts)
  - Session storage : emptied when the browser is closed
- Data model : associative arrays (key/value)

Using the local storage

- Start by checking whether or not it's supported by your browser :

Local storage

```
if (typeof(Storage) !== "undefined") {
  // you can use the local storage
} else {
  // local storage is not supported
}
```

Writing data

- The local storage can only store character strings
- There are two ways of storing data -- remember that LS stores associative data

Writing data

```
/* Creates the key 'name' and
   associate the value 'Doe' */

// 1st way
localStorage.setItem('name', 'Doe');

// 2nd way
localStorage.name = 'Doe';
```

Keys are unique !

If you write

```
localStorage.setItem('name', 'Doe');
```

followed by

```
localStorage.setItem('name', 'Smith');
```

you will overwrite the previous value associated to the key 'name'

Reading data

- You can access values using their key.
- If you try to access the value of a key that does not exist, the returned value is `null`

Reading data

```
/* Fetch the value associated to the key given as parameter */

// 1st way
var item = localStorage.getItem('name');

// 2nd way
var item = localStorage.name;
```

Development tools and local storage

CHROME

MDN web docs

Technologies

& Guides

Sign in

Search

localStorage.getItem()

Languages

The `getItem()` method of the `Storage` interface, when passed a key name, will return that key's value, or `null` if the key does not exist, in the given `Storage` object.

Elements | Console | Application

Application
- Manifest
- Service Workers
- Clear storage

Storage
- Local Storage
  - https://developer.m
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache
- Cache Storage
- Application Cache

Frames

Filter

Key	Value
taskTracker	1550248275287

1 1550248275287

Line 1, Column 1

Console | Network conditions | What's New

Development tools and local storage

FIREFOX

MDN web docs

Tech ies

& Guides

Get Firefox help

Get web development help

Join the MDN community

Report a content problem

Report a bug

Sign in

Search

localStorage.getItem()

Languages

Inspector | Console | Storage

Cache Storage

Cookies

Indexed DB

Local Storage

https://developer.mozilla.org

Session Storage

Filter items

Key | Value

No data present for selected host

Data serialization with JSON

- Serialization = converting data into a character string (for storage, sharing, ...)
- JSON = JavaScript Object Notation

The local storage stores only character strings, but we work on complex objects

=> serialize these objects with JSON before storing them in the local storage

Serialization

```
function Person(lastname, firstname, age) {
  this.lastname  = lastname;
  this.firstname = firstname;
  this.age = age;
  this.sayHi = function(){
    return "Hi, I'm " + this.firstname;
  }
};

var myObj = new Person('Doe', 'John', 25);

var myObjJSON = JSON.stringify(myObj);
localStorage.setItem('personObj',myObjJSON);
```

Key	Value
personObj	{"lastname":"Doe","firstname":"John","age":25}
personObjRaw	[object Object]

But… where’s my method sayHi() ?

- JSON is meant to hold data only…
- In the object paradigm, it does not make sense to “store” functions in a serialized object
- What interest us in an object is its properties (the behavior is common across all instances !)
- So, how to we get back the object AND its behaviors ? Let’s focus on the properties for now…

Data deserialization with JSON

- To deserialize a serialized object (i.e. build an object from the attributes stored in the stringified object), use `JSON.parse(serializedObj)`

Deserialization

```
var myObj2 = JSON.parse(myObjJSON);

// or, retrieving the serialized object from the local storage
var myObj_serialized = localStorage.getItem('personObj');
var myObj_parsed     = JSON.parse(myObj_serialized);
```

Wait, what about sayHi() ?

Copy parsed object properties

```
var personObj = new Person();
// Copy the attributes of myObj_parsed into personObj
Object.assign(personObj, myObj_parsed); // ES6 native
console.log(personObj.sayHi()); // ok !
```

Chapter recap

- CDN & jQuery syntax
- Animate elements
- DOM : accessing elements content
- DOM : adding / removing elements
- Traversing the DOM
- LocalStorage
- Data serialization with JSON