jQuery 2013 Webinar - Documentation

Table of Contents

1. Introduction

In the last webinar we were introduced to Knockout.js and the ASP.NET Web API as well as REST principles. We continuously used a JavaScript library called "jQuery". During the recent webinars nobody asked for an explanation of the jQuery code. This was no surprise, since Query is the most popular JavaScript library on the market. Chances are very high that you have already seen jQuery in the wild, too. In most of my projects I have seen jQuery as a standard for DOM manipulation and AJAX handling. These use cases are inbuilt since the very early beginning of jQuery. Due to the fact that everything works smoothly and that API stayed compatible for most of the cases, a lot of people do not really care about the progress that the framework has made since the early days. Let's start with an introduction for jQuery newcomers and continue with some parts that have changed during the versions.

2. jQuery core

2.1. Basics

This plain JavaScript snippet manipulates the Document Object Model (DOM) a bit by adding a background color and some padding to all elements:

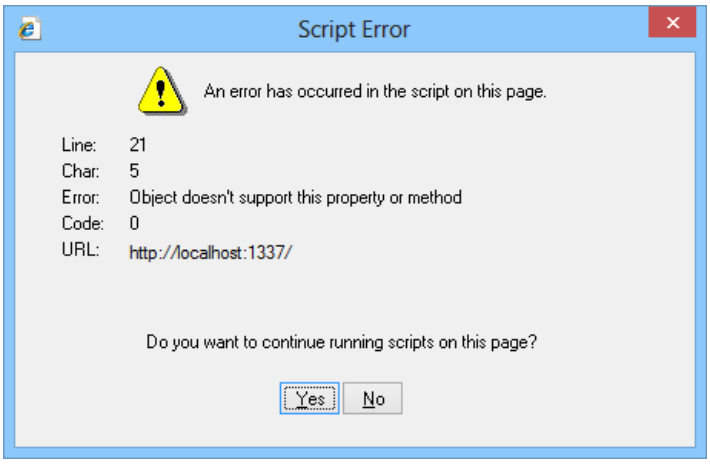
```
<div class="postit">
    <h2>Hello World</h2>
</div>

<script>
    var divs = document.getElementsByClassName('postit');

    for (i = 0; i < divs.length; i++) {
        divs[i].style.backgroundColor = '#F0F0A6';
        divs[i].style.padding = '10px';
    }
</script>
```

During development you want to concentrate on solving tasks instead of writing boilerplate code. But hey, at least it works, doesn't it?

It doesn't.. Soon we would find out that there are numerous compatibility issues across different browsers. This snippet for example doesn't works in until IE9!



(reference)

The shown method is specified since the very early days, but for a full decade the Internet Explorer hasn't implemented it. So we are forced to use the following extra bits of code to get the expected result in IE8 (older version will still struggle):

```
if(!document.getElementsByClassName) {
    document.getElementsByClassName = function(className) {
        return this.querySelectorAll("." + className);
    };
    Element.prototype.getElementsByClassName = document.getElementsByClassName;
}
```

Or we use a small framework that hardens and simplifies DOM manipulation, event handling, animation and AJAX. The same result can be expressed like this with jQuery.

```
<div class="postit">
    <h2>Hello World</h2>
</div>

<script>
$(".postit").css({
    backgroundColor: '#F0F0A6',
    padding: '10px'
});
</script>
```

It's much more easier to write jQuery than pure JavaScript. For most cases there is always the same procedure:

1. Select some HTML
2. Do something with it

2.1.1 Selectors

JQuery uses the Sizzle Selector Engine to select one or more HTML elements via CSS3 selectors. For this, the $ function must be called with a string containing a such a selector. The matched HTML elements will be presented as an array-like collection of jQuery objects. The term **jQuery "wrapped set"** is often used.

Here are some examples:

```
$("#myId")       // selects (one) element with the id "myId"
$(".myClass")    // selects all elements with the class name "myClass"
$("div")         // selects all elements of type "div"
$("div:first")   // selects the first element of type "div"
$("div:odd")     // selects odd "div" elements, zero-indexed
$("input:not(:checked)") // equals to
$("input").not(":checked")
```

There are much more selectors to discover.

Performance Tips:

1. **Use IDs where possible**

```
$("#myId")
```

The native document.getElementById method will be used.

2. **Avoid selecting by class only**

```
$(".myClass")    // slow
$("div.myClass") // fast
```

The second option wouldn't be slow in older browsers and might help other developers to understand your code, too.

3. **KISS**
Keep it Simple. Complex selectors are just an result of complex HTML. Complex selectors are slow.

4. **Increase Specificity from Left to Right**
If no optimization can be applied, the selector engine (as well as the browsers CSS selectors) are matched from right to left. This can be be very slow if there are a lot of paragraphs in the document:

```
$("div.content p"); // might be slow
```

First it will find all p elements in the document and then checks them against the "div.content" condition.
Depending on the document, it might be a good idea to help the selector engine. This example reduces the amount of work, since the left side is matched first and the next search can iterate over a reduced amount of elements:.

```
$("div.content").find("p"); // might be faster
```

Please note: those tricks should be only done when necessary. You will know when it's time to optimize the selectors. To find hot spots and to verify the optimization, you should take a closer look on a performance comparison tools. (eg. Firebug)

5. **Avoid touching the DOM**
This is the most important rule: All browsers there have two engines; the Rendering Engine and the JavaScript engine. As soon as the JS engine is going to talk to the DOM (provided by the rendering engine) speed is going drastically down. (by the way: Chrome/Blink plans to change that)

```
// bad:
$("p").css("color", "blue");
$("p").addClass("myClass");
.
// chaining
$("p").css("color", "blue")
  .addClass("myClass");
.
// save to local variable
var $p = $("p");
$p.css("color", "blue")
$p.addClass("myClass");
```

2.1.2 Actions on jQuery objects

The result of such a selection is a jQuery wrapped set.
The elements of a set can be called **jQuery "wrapped objects"**. A huge amount of functions can be applied on them. This includes:

1. Manipulation of elements
`append(), appendTo(), remove(), before(), after()`

2. Change of element attributes
`css(), attr(), html(), val(), addClass()`

3. Traversing
`find(), is(), prevAll(), next(), hasClass()`

4. Events
`click(), on(), off(), trigger() --- Deprecated: bind(), live()...`

5. Effects
`show(), fadeOut(), toggle(), animate()`

6. AJAX
   `ajax(), get(), post()`

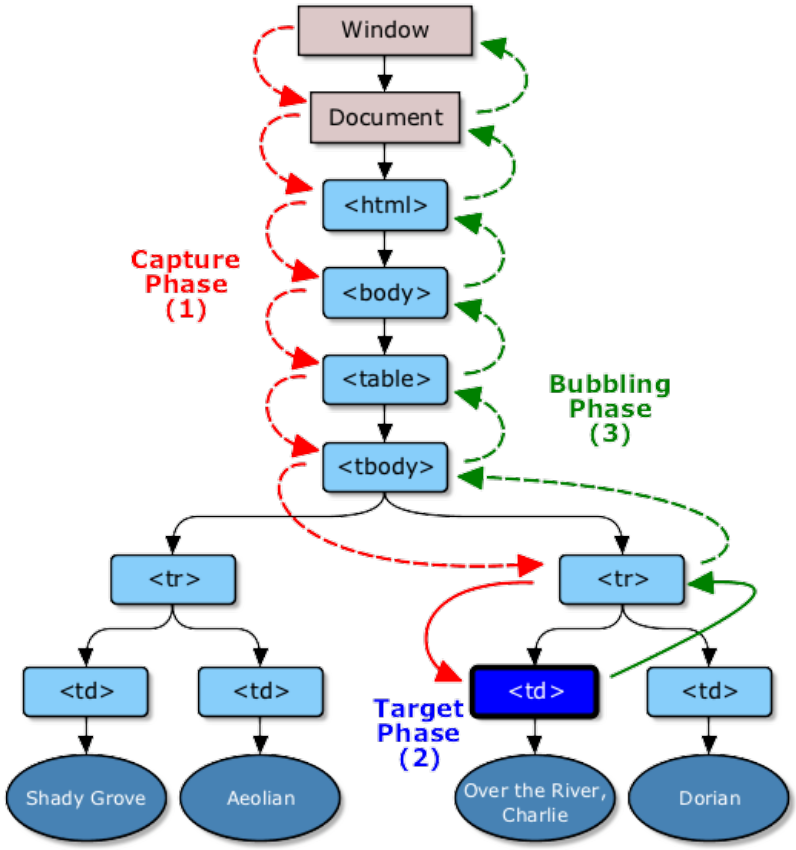
One example which combines a lot of functions:

```
$('<h1>')
    .prependTo("body")
    .text(":-(")
    .css({
        position: "absolute",
        top: "50%",
        left: "50%",
        zIndex: "999"
    })
    .click(function () {
        $(this).fadeOut(function() {
            $(this).load("/examples/smilie.html").fadeIn();
        });
    });
```

2.2.Event Models

DOM Level 3 Event Model

All user interaction on a website is driven by events. The Document Object Model (DOM) Level 3 Events Specification is are standardized by the W3C. As soon as event (eg. mouse or keyboard activity) occurs and event object is created. As the next step, the event object must complete one or more event phases. The specification defines three event phases: capture phase, target phase and bubble phase. Events "flow" through the DOM until they have either finished all phases or were stopped by a script.



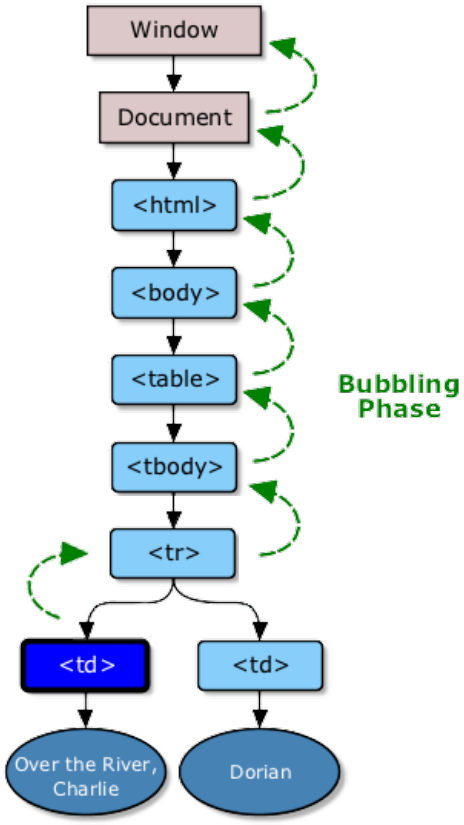
This example shows an event binding with plain DOM methods:

```
document.getElementsByTagName('a')[0]
    .addEventListener("click", function() {
        alert(this.text);
});
```

Needless to say that all browsers should implement the specification. But as you might have guessed, they don't.

jQuery Event Model

JQuery does a great job by hiding the browsers differences. It just works!™ The capture phase was skipped due to to its lack of support in IE. It also simplifies the API. In the end, jQuery strips everything except bubbling (technical term: event propagation)



The rest is very easy: Select one or more DOM elements. Apply a handler to it:

```
$('a').on('click', function() {
    alert("Hello World");
});

// shortcut
$('a').click(function() {
    alert("Hello World");
});
```

For your reference, here is a list off all jQuery Event methods (including shortcut methods). There are more events that can be caught, but here you should take care of browser support.

Advanced jQuery Events

The bad news: the event API was changed several times and a lot of functions are now deprecated or even removed.
The good news: the latest API is really easy to use and avoids drawbacks of previous version.

Lets assume we have a found a nice jQuery plugin to display data in a nicely formatted table. After the document is ready, we are loading data via AJAX. Additionally we would like to show a message when somebody clicks on a row. (see demo project!)

```
<script>
require(['jquery', 'jquery.dataTables'], function() {

    $(function() {

        // loads table
        $('#example').dataTable({
            sAjaxSource: "/api/Note/"
        });

        // registers click handler (does not work!)
        $('td').click(function() {
            alert($(this).text());
        });
```

```
    });
});
</script>

<table class="display" id="example">
    <thead>
        <tr>
            <th>Id</th>
            <th>Title</th>
            <th>Message</th>
            <th>Added</th>
            <th>Categories</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td colspan="5" class="dataTables_empty">Loading data from server</td>
        </tr>
    </tbody>
</table>
```

The table looks great, but the click handler does not work!
The problem behind this is very easy to understand. `$(function(){})` is a shortcut for `$(document).ready(function() {});`. It fires only once, after the DOM content was loaded. But the table is constructed after that event. So the selector for the handler will have no match at all, hence no binding will be applied.

JQuery 1.3 was introduced with `.live()` which magically worked every time, event if an targeted element was added later on. As of jQuery 1.4.3, it was superseded by `.delegate()` due to performance issues. Since version 1.7 the jQuery team finally recommends `.on()`.

```
$(selector).live(events, data, handler);                // jQuery 1.3+
$(document).delegate(selector, events, data, handler);  // jQuery 1.4.3+
$(document).on(events, selector, data, handler);        // jQuery 1.7+
```

> Good to know: As we have seen, browser events bubble from the innermost element (the event target) where they occur all the way up to the document element. The live() method waited on the document element for events, which was very slow! Now delegated events are recommended.

This reworked example uses event bubbling and waits an a element that is already existing on startup. With the help of the selector we can change the scope again to the <td> element.

```
// registers a delegated event handler
$(function() {
    $('table').on('click', 'td', function() {
        alert($(this).text());
    });
});
```

2.3. Asynchronous Communication

Since the very first day jQuery is providing a stable and easy to use AJAX API. Again it is working around browser issues and is even able to communicate between different domains. (it avoids the Same-Origin-Policy by using JSONP)

This is the old way of expressing an AJAX call, where multiple callbacks must be part of the method signature:

```
$.ajax({
    url: "/examples/webinar.json",
    success: function(result) {
        $.each(result, function(index, value) {
          console.log(value.Title);
        });
    }
});
```

However, this way had limitations. The refactored AJAX methods provide a deferred object (jqxhr). This leads to a cleaner API. The biggest advantage is the fact, that the deferred object is a chainable utility object. It can register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

```
$.ajax("/examples/webinar.json")
.done(function(result) {
    $.each(result, function(i, value) {
        console.log(value.Title);
    });
});
```

Deferred objects implement the Javascript Promise Pattern and are inspired by the CommonJS Promises proposal A which is still under discussion . JQuery promises can be composed with multiple `done` and `fail` handlers. After an asynchronous operation ended, jQuery will run the **.done()** handlers when all the passed promises succeed, its **.fail()** handlers if any of them fail and finally its **.always()** handlers.

The API holds a lot of nugets. In this example we are waiting for two calls to finish:

```
$.when(
    $.ajax("/examples/webinar.json"),
    $.ajax("/examples/webinar.json"))
.done(function(result1, result2) {
    var bothResults =  result1[0].concat(result2[0]);
    $.each(bothResults, function(i, value) {
        console.log(value.Title);
    });
});
```

3. ASP.NET MVC & Bundling

Bundling and minification are a new feature of ASP.NET MVC 4.5. The demo code bundles all scripts to improve download sizes and performance. Bundling and minification is enabled or disabled by setting the value of the debug attribute in the compilation Element in the Web.config file. Change the debug switch to false to enable it.

```
<system.web>
    <compilation debug="false" />
</system.web>
```

4. More

There are several more things to discover.
You should start by downloading the sources of the demo app.

Will be published after the presentation.

A great book is "jQuery in Action". It goes into details where other books still cover boring topics!