

JQUERY



Agenda

2

- What is the jQuery?
- Wrapped Set
- Useful API
- DOM selection
- DOM traversal
- DOM creation
- DOM events

iQuery

3

- ❑ “Write less, do more”
- ❑ Open source JavaScript library
- ❑ Released at 2006
- ❑ Provides CSS 3 based syntax for DOM traversing
- ❑ Eliminates cross-browser differences
- ❑ Extensible
- ❑ Making DOM manipulation easier
- ❑ <http://jquery.com/>

Why jQuery ?

4

- ❑ You have no other choice 😊
- ❑ Open source
- ❑ Free
- ❑ Adopted by many leading companies
- ❑ Easy to integrate into existing application
- ❑ IE6+ (Wow ...)
- ❑ Very active community
- ❑ Lightweight

Alternatives

5

Topics

[Subscribe](#)**jquery**

Search term

prototype

Search term

dojo

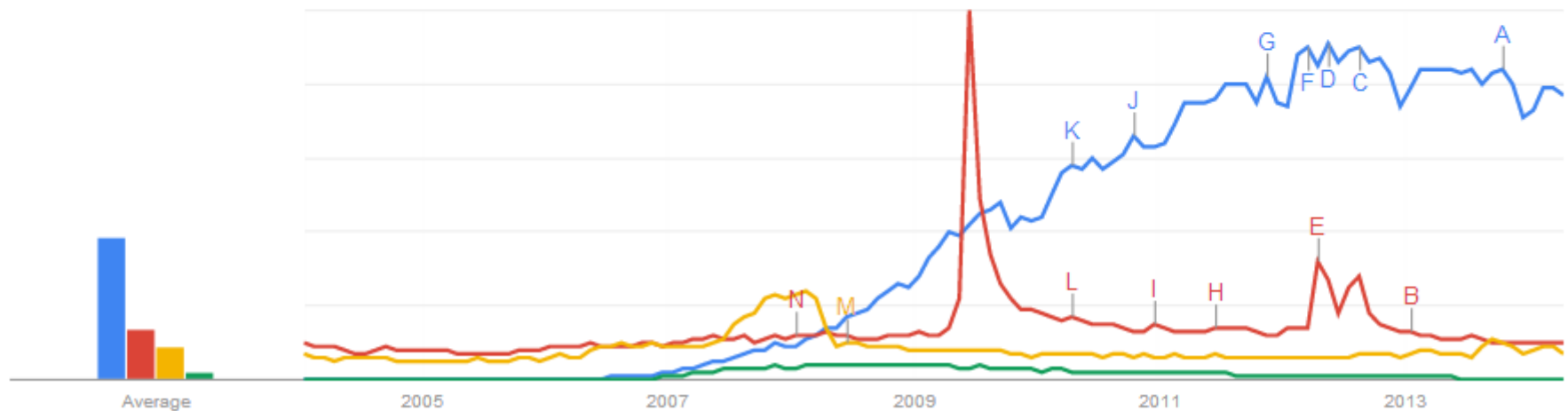

Search term

mootools

Search term

[+ Add term](#)

Interest over time

☒ News headlines☐ Forecast 

Getting Started

6

- Download jQuery script from <http://jquery.com/>
 - Compressed
 - Uncompressed
- Can use CDN instead of local script
 - Google
 - Microsoft
 - jQuery
- Include it in your HTML
- Start using the **jQuery** global object

jQuery Global Object

7

- It is actually a function
- Has an alias named `$`
- There are different ways to use it
 - ▣ `$("div")` – Search
 - ▣ `$("<div />")` – Create
 - ▣ `$(function(){...})` – DOM Ready
 - ▣ `$(element)` – Wrap native DOM element
 - ▣ `$.ajax` – Global API
- See next slides

\$ Conflicts

8

- \$ alias is not reserved for jQuery
- Other 3rd party libraries might use it too
- How can we ensure no conflicts ?
 - ▣ Use closure
 - ▣ Never use the global \$

```
(function ($) {  
    // $ here is for sure jQuery  
  
    // Ask jQuery to restore original $ value  
    // Can omit this line  
    $.noConflict();  
})(jQuery);
```


Initialization

9

- Most of the time we are using jQuery for DOM manipulation
 - ▣ For example, animation
- To manipulate the DOM we must first wait for it to be completely loaded
- There is a standard DOM event named **DOMContentLoaded**
 - ▣ However, is not supported under old browsers IE8-

Waiting for the DOM

10

- jQuery offers cross browser DOM ready event
 - ▣ On new browsers it uses DOMContentLoaded
 - ▣ Old browsers require some nasty tricks
- Two ways to do the same thing

```
<script>
    $(document).ready(function () {
        alert("DOM is ready");
    });

    $(function () {
        alert("DOM is ready");
    });
</script>
```

DOM Selection

11

- Construct a CSS selector
- Send it to \$ as a string
- jQuery looks for all matching elements
- Returns an array of results
 - ▣ A.K.A jQuery wrapped set
 - ▣ Offers a rich API

```
// Change all links color to red
```

```
var res = $("a");  
res.css("color", "red");
```

DOM Selection

12

- Select according to class

```
$(".items")
```

- Select according to html tag

```
$("div")
```

- Select according to id

```
$("#button1")
```

- Combination of above

```
$("#main input.simpleButton")
```

Wrapped Set – Be Aware

13

- ❑ jQuery wrapped set is an array of DOM elements
- ❑ Not array of jQuery objects !!!

```
$(function () {  
    var set = $(".items .item");  
    for (var i = 0; i < set.length; i++) {  
        var item = set[i];  
  
        alert(item.innerHTML);  
    }  
});
```

Method Chaining

14

- Most jQuery functions return the original result set
- This allow us to chain method calls

```
//  
// Change all links color to red and text to XXX  
//  
$("a").css("color", "red").text("XXX");
```

- This style of writing is very popular amongst jQuery developer
- However, code is not clear

Attribute selectors

15

- Attribute starts with

```
$(".item[type^=button]")
```

- Attribute contains

```
$(".item[type*=button]")
```

- Attribute equals

```
$(".item[type=button]")
```

- Has attribute

```
$(".item[type]")
```

Pseudo selectors

16

□ Select even element

```
$("tr:even")
```

□ Select N child

```
$("tr:nth-child(3n)")
```

□ Do not match

```
$("input:not([type=button])")
```


Form Selectors

17

- Input tags of type button and buttons tags

```
$(".:button")
```

- All input tags, select, and textarea

```
$(".:input")
```

- Only checked checkboxes and radio buttons

```
$(".:checked")
```

- Selected option inside select tag

```
$(".:selected")
```

Position Selectors

18

- Select by index

```
$("#div:eq(2)")
```

- Select elements with index greater than

```
$("#div:gt(1)")
```

- Select elements with index less than

```
$("#div:lt(4)")
```

- Select first and last element

```
$("#div:first") $("#div:last")
```

Working with a jQuery Object

19

- Assuming you hold a reference to jQuery object
- You can invoke any of jQuery DOM element APIs
 - ▣ Too many to cover
- Important ones
 - ▣ html & text
 - ▣ css & addClass
 - ▣ attr & prop
 - ▣ bind & delegate & on

html vs. text

20

□ Get/set the content of an element

```
$("#div").html()
```

```
$("#div").html("New Content")
```

□ Get/set the text of an element

▣ All tags are removed

```
$("#div").text()
```

```
$("#div").text("New Text")
```

Styling - css

21

- Get/set the inline styles of an element

```
$("#div").css("background-color")
```

```
$("#div").css("background-color", "red")
```

- Multiple styles

```
$("#button").css({  
  color: "red",  
  "font-size": "2em",  
});
```

Styling - addClass

22

- Using **css** method is considered poor design
- Managing inline-styles at runtime is complex
- You should consider adding a CSS class to the element and set the styles on this class

```
<style>
  .btn {
    color: red;
    font-size: 2em;
  }
</style>
```

```
$(document).ready(function () {
  $("button").addClass("btn");
});
```

- Use **hasClass** to determine if a CSS class is present on an element

Attributes

23

- Get/set the value of a specific attribute

```
$("#div").attr("checked")
```

- Returned value may be a
 - ▣ string – Attribute's value
 - ▣ undefined – Attribute is not present on the element
- Attribute may be empty
- Hence, below code is problematic

```
if (!$("input").attr("xxx")) {  
    // Not sure if attribute is missing or just empty  
}
```

```
<input type="checkbox" xxx />
```

Size & Position

24

- **width/height** – Content size
 - ▣ Read/write API
- **innerWidth/innerHeight** – Include padding
- **outerWidth/outerHeight** – Include border
 - ▣ Send true to include margin too
- **position** – Relative to its non static parent
 - ▣ Read-only
- **offset** – Relative to the document
 - ▣ Read/write

DOM Traversal

25

- Search inside a DOM element

```
$(".items").find(".item")
```

- Get only direct children

```
$(".items").children(".item");
```

- Get the list of parent elements

```
$(".items").parents();
```

- Get sibling elements

```
$(".items").siblings();
```

DOM Traversal (2)

26

□ Previous sibling element

```
$("#header").prev()
```

□ Next sibling element

```
$("#header").next()
```

□ Get the list of parent elements

```
$(".items").parents();
```

□ The first matching ancestor

```
$(".items").closest("li");
```

DOM Creation

27

- Construct HTML string and send it to \$
- Get back a reference to the newly created DOM element
- The newly created element is detached from the DOM
- Insert the new element into the document

```
var button = $("<button>Click Me</button>");  
$("body").append(button);
```

DOM Creation Techniques

28

□ Append HTML string

```
$("#body").append("<button>Click Me</button>");
```

□ Create and appendTo

```
$("#<button>Click Me</button>").appendTo("#body");
```

□ Other techniques

- Before/insertBefore

- after/insertAfter

- prepend/prependTo

- replaceWith/replaceAll

Moving Element

29

- No special API
 - ▣ Select existing element
 - ▣ Append it to another → It will be moved not copied

```
$("#button").appendTo("div");
```

- If target is an array the moved element will be removed and then copied to all targets

Removing Element

30

- Clear content – The element itself is not removed

```
$("#header").empty()
```

- Totally Remove an element

```
$("#header").remove()
```

- Remove an element but keep any jQuery related data
 - ▣ Event handlers
 - ▣ Attached user data

```
$("#header").detach();
```

Event Handling

31

- Native DOM offers the **addEventListener** API
- jQuery offers several methods instead
 - ▣ bind/unbind
 - ▣ delegate/undelegate
 - ▣ live/die
 - ▣ on/off
- on/off is the latest and can be used instead of all others

bind

32

- Accepts the following
 - ▣ Event name – click, blur, focus, ...
 - ▣ Data object – User defined state
 - Is used rarely
 - ▣ Callback – A function to be invoked when event is raised

```
var input = $("input");
input.bind("click", function () {
    alert("Button was clicked");
});
```


unbind

33

- Registering event handler means that the DOM holds a reference back to your objects
- This reference holds your objects alive
- When building SPA it is important to clear event handlers to allow GC collection

```
// specific handler
input.unbind("click", handler);

// all handlers for a specific event
input.unbind("click");

// all handlers for all events
input.unbind();
```

Bind Shortcuts

34

- For some common events, jQuery offers shortcut methods instead of bind
 - click
 - dblclick
 - blur
 - focus
 - keywodn/keyup/keypress
 - mouseup/mousedown
 - Change
 - More ...

```
$("#button").click(function () {  
    console.log("Button was clicked");  
});
```

What did happen to my this ?

35

- Inside a DOM event handler the **this** keyword points to the DOM element that raised the event

```
$("#button").click(function () {  
    // outputs true  
    alert(this.nodeName == "BUTTON");  
});
```

- Consider wrapping it inside jQuery object
 - ▣ Get back access to all jQuery APIs

```
$("#button").click(function () {  
    var button = $(this);  
    alert(button.text());  
});
```

Event Handler inside a Class

36

- Writing Object Oriented JavaScript and handling DOM event is tricky

```
function HomeView(element) {  
  this.element = element;  
  this.buttonLogin = this.element.find("button.login");  
  this.buttonLogin.click(this.login_Clicked);  
}  
  
HomeView.prototype.login_Clicked = function () {  
  // Exception is thrown here, why ?  
  this.buttonLogin.attr("disabled", "disabled");  
}  
  
$(document).ready(function () {  
  var homeView = new HomeView($(".home-view"));  
});
```

```
<body>  
  <div class="home-view">  
    <button class="login">Login</button>  
  </div>  
</body>
```

Event Object

37

- An optional object that is sent to the event handler
- Contains information about the event itself
 - ▣ Mouse position, Keyboard state, ...
- jQuery normalizes the event, ensuring that all standard properties exist

```
$(document).ready(function () {  
    $("button").click(function (e) {  
        // True if alt key was pressed during button click  
        console.log(e.altKey);  
    });  
});
```

Event Object is a Clone

38

- jQuery clones the browser's original event object and normalizes it
- To get access to non standard fields that are not copied by jQuery
- Use **originalEvent**

```
$("#button").click(function (e) {  
    console.log(e.originalEvent.dataTransfer);  
});
```

- Or, ask jQuery to always copy field from original object

```
jQuery.event.props.push("dataTransfer");
```

Challenge

39

- Suppose we have a table with 5000 rows
- We want to handle dblclick on each row
- Naïve solution

```
$("#table tr").dblclick(function () {  
    alert("Row was double clicked");  
});
```

- Bad performance
 - ▣ Selecting 5000 DOM elements into memory
 - ▣ Invoking addEventListener 5000 times

Solution

40

- Register dblclick handler only once
 - ▣ On the root table element
- The browser propagates dblclick events from a DOM element to its parents hierarchy

```
$("table").dblclick(function () {  
    alert("Row was double clicked");  
});
```

- Cons
 - ▣ Not clear which row was clicked
 - ▣ The handler is invoked even when clicking on non row element
 - ▣ Not every DOM event is propagated

delegate

41

- Installs event handler on a root object
- The handler is invoked only when source DOM element matches a specified selector

```
$("#table").delegate("tr", "click", function () {  
    // this is row not a table  
    var row = $(this);  
});
```

- The this reference points to the clicked row
 - Not to the table

live

42

- A deprecated method
- Same as delegate but the root object is always the document
- Therefore less optimized
- You might still encounter it in old jQuery based code
- **live** was removed from jQuery 1.9 !!!

```
$("#tr").live("click", function () {  
    var row = $(this);  
});
```

on

43

- A replacement for bind and delegate
- An overloaded method
- **bind** like usage

```
$("#tr").on("click", function () {  
    console.log("Row was clicked");  
});
```

- **delegate** like usage

```
$("#table").on("click", "tr", function () {  
    console.log("Row was clicked");  
});
```

Summary

44

- jQuery is a de-facto standard for DOM manipulation
- Many 3rd party libraries are using it
 - ▣ Telerik
 - ▣ Angular
- Is a library, not a framework
 - ▣ Easy to integrate into existing code
 - ▣ However, does not help you with modeling/layering