# Introducing jQuery

## Setup

Obtain latest version of jQuery from <http://jquery.com>. Installing jQuery is as easy as placing it within your web application and using the HTML <script> tag to include it in your pages:

<html>  
<head>  
 <script src="jquery-1.4.js"></script>  
</head>  
</html>

## jQuery Fundamentals

At it’s core, jQuery focuses on retrieving elements (see CSS selectors) from HTML pages and performing operations upon them.

### The jQuery Wrapper

jQuery makes use of CSS selectors

To collect a group of elements, we pass the selector to the jQuery function:

$(selector) or

jQuery(selector)

The $ function returns a JavaScript object containing an array of DOM elements in the order in which they are defined in the HTML document. This object possesses a large number of predefined methods that can act on the collected group of elements. It is known as a wrapper because it wraps collected elements with extended functionality.

A feature of a large number of these methods is that when they are finished, they return the same group of elements ready for another action:

$(selector).hide().addClass(“removed”)

### Utility Functions

jQuery $() function also serves as a *namespace prefix* for a handful of general-purpose utility functions.

Eg: var trimmed = $.trim(someString)

### The document ready handler

Behaviour is typically separated from structure so we’re performing operations on page elements outside of the document markup that creates them. We need a way to wait until DOM elements are fully realized before these operations execute. jQuery provides a simple means to trigger the execution of code once the DOM tree has loaded (without waiting for external resources)

$(document).ready( function() { alert(“hello world”) } )

### Making DOM elements

We can create DOM elements on the fly by passing the $() function a string that contains the HTML markup for those elements : $(“<p>Hello world</p>”)

# Selecting Elements Upon Which to Act

## Selecting elements for manipulation

jQuery uses CSS syntax and extends it with some custom means to perform both common and complex selections.

See SelectingElements.html

### Controlling the context

$() function actually takes 2 arguments. When first argument is a selector, the second argument denotes the context of the operation. The context defaults to applying that selector to every element in the DOM.

Subtrees of the DOM can be specified as the context using a DOM element reference, jQuery selector or a wrapped set.

Eg: $(selector,’div#DOMsubtree’)

### Using basic CSS selectors

See CSS3  
Some simple CSS selector examples

|  |  |
| --- | --- |
| Example | Description |
| a | Matches all anchor ( <a> ) elements |
| #specialID | Matches the element with the id value of specialID |
| .specialClass | Matches all elements with the class specialClass |
| a#specialID.specialClass | Matches the element with the id value specialID if it’s an anchor tag and has class specialClass |
| p a.special class | Matches all anchor elements with the class specialClass that are descendants of <p> elements |

### Using child, container, and attribute selectors

#### Child selectors

ul.myList > li > p : selects only paragraphs that are *direct* children of list elements which in turn are *direct* children of <ul> elements that have the class myList

See child.html

#### Attribute selectors

a[href^=’http://’] : selects all links with an href value beginning with ‘http://’ ie external sites

a[href$=’.pdf’] : all pdfs

See attribute.html

|  |  |
| --- | --- |
| Selector | Description |
| \* | Matches any element |
| E | Matches all elements with tag name E |
| E F | Matches all elements with tag name F that are descendants of E |
| E > F | Matches all elements with tag name F that are direct children of E |
| E + F | Matches all elements with tag name F that are immediately preceded by sibling E |
| E ~ F | Matches all elements with tag name F preceded by any sibling E |
| E.C | Matches all elements with tag name E with class name C |
| E#I | Matches all elements with tag name E with id of I |
| E[A] | Matches all elements with tag name E that have attribute A of any value |
| E[A=V] | Matches all elements with tag name E that have attribute A whose value is exactly V |
| E[A^=V] | Matches al elements with tag name E that have attribute A whose value starts with C |
| E[A$=V] | Matches all elements with tag name E that have attribute A whose value ends with V |
| E[A!=V] | Matches all elements with tag name E that have attribute A whose value doesn’t match the value of V, or that lack attribute A completely |
| E[A\*=V] | Matches all elements with tag name E that have attribute A whose value contains V |

### Selecting by position

a:first selects first <a> element on page (see :last)

p:odd selects every odd paragraph element (see :even)

ul li:last-child selects last <li> child of each <ul> element (see :first-child)

CSS specification refers to these types of selectors as *pseudo-classes*, but jQuery refers to them as *filters* because they filter a base selector. Easy to spot as they all start with colon (:)  
See position.html

|  |  |
| --- | --- |
| Selector | Description |
| :first | Matches the first match within the context. li a:first returns the first link that’s a descendant of a list item |
| :last | Matches the first match within the context. li a:last returns the last link that’s a descendant of a list item |
| :first-child | Matches the first child element within the context. li:first-child returns the first item of each list |
| :last-child | Matches the last child element within the context. li:last-child returns the last item of each list |
| :only-child | Returns all elements that have no siblings |
| :even | Matches even elements within the context. li:even returns every even list item |
| :odd | Matches odd elements within the context. li:odd returns every odd list item |
| :eq(n) | Matches the nth matching element |
| :gt(n) | Matches matching elements after and excluding the nth matching element |
| :lt(n) | Matches matching elements before and excluding the nth matching element |

### Using custom jQuery filter selectors

|  |  |
| --- | --- |
| Selector | Description |
| :button | Selects only button elements |
| :checkbox | Selects only checkbox elements |
| :checked | Selects only checkboxes or radio elements in checked state |
| :contains(mytext) | Selects only elements containing the text mytext |
| :disabled | Selects only elements in disabled state |
| :enabled | Selects only elements in enabled state |
| :file | Selects only file input elements |
| :has(selector) | Selects only elements that contain at least one element that matches the specified selector |
| :header | Selects only elements that are headers |
| :hidden | Selects only elements that are hidden |
| :image | Selects only image input elements |
| :input | Selects only form elements |
| :not(selector) | Negates the specified selector |
| :parent | Selects only elements that have children (inc text), but not empty elements |
| :password | Selects only password elements |
| :radio | Selects only radio elements |
| :reset | Selects only reset buttons |
| :selected | Selects only <option> elements that are in selected state |
| :submit | Selects only submit buttons |
| :text | Selects only text elements |
| :visible | Selects only elements that are visible |

## Generating new HTML

See GeneratedHTML.html

## Managing the wrapped element set

We can refine, extend or subset the set of wrapped elements we wish to operate on.

### Determining size of wrapped set

$(selector).size()

### Obtaining elements from wrapped set

$(selector).get(index)  
$(selector).eq(index)  
$(selector).first  
$(selector).last

### Slicing and dicing the wrapped set

May want to augment set by adding elements or reducing to a subset

#### Adding elements

$(selector).add(selector)

See AddElements.html

#### Removing elements

$(selector).not(selector)  
$(selector).filter(expression)

See RemoveElements.html

#### Obtaining subsets of wrapped set

$(selector).slice(x,y)  
$(selector).has(test)

#### Translating elements of wrapped set

$(selector).map( function() { … } )

See Translating.html

#### Traversing a wrapped set

$(selector).each( function(n) { … } )

See Traversing.html

### Getting wrapped sets using relationships

jQuery allows us to get new wrapped sets from an existing set, based on the hierarchical relationships of the wrapped elements to the other elements within the HTML DOM

See Relationships.html

|  |  |
| --- | --- |
| Method | Description |
| children([selector]) | Returns a wrapped set consisting of all unique children of the wrapped elements |
| closest([selector]) | Returns a wrapped set containing the single nearest ancestor that matches the passed expression, if any |
| contents() | Returns a wrapped set of the contents of the elements, which may include text nodes, in the wrapped set. |
| next([selector]) | Returns a wrapped set consisting of all unique next siblings of the wrapped elements |
| nextAll([selectr]) | Returns a wrapped set containing all the following siblings of the wrapped elements |
| nextUntil([selector]) | Returns a wrapped set of all the following siblings of the elements of the wrapped elements until, but not including, the element matched by the selector. |
| offsetParent() | Returns a wrapped set containing the closest relatively or absolutely positioned parent of the first element in the wrapped set |
| parent([selector]) | Returns a wrapped set consisting of the unique direct parents of all wrapped elements |
| parents([selector]) | Returns a wrapped set consisting of the unique ancestors of all wrapped elements. This includes the direct parents as well as the remaining ancestors all the way up to, but not including, the document root |
| parentsUntil([selector]) | Returns a wrapped set of all ancestors of the elements of the wrapped elements up until, but not including, the element matched by the selector. If no matches are made to the selector, or if the selector is omitted, all ancestors are selected. |
| prev([selector]) | Returns a wrapped set consisting of all unique previous siblings of the wrapped elements |
| prevAll([selector]) | Returns a wrapped set containing all the previous siblings of the wrapped elements |
| prevUntil([selector]) | Returns a wrapped set of all preceding siblings of the elements of the wrapped set until, but not including, the element matched by the selector. If no matches are made to the selector, or if the selector is omitted, all previous siblings are selected |
| siblings([selector]) | Returns a wrapped set consisting of all unique siblings of the wrapped elements |

### Managing jQuery chains

Consider the following statement:

$(‘img’).filter(‘[title]’).hide()

Two wrapped sets are generated within this statement: the original set of all <img> in the DOM and a second set consisting of only <img> elements which possess title attributes.

What if we want to apply a method, such as adding a class name, to the original wrapped set *after* it’s been filtered? We can’t tack it onto the existing chain as that would only affect titled images.

end() allows us to backup to the previous wrapped set:

$(‘img’).filter(‘[title]’).hide().end().addClass(‘anImage’)

andSelf() merges the two topmost sets into a single wrapped set

# Bringing pages to life with jQuery

## Working with element properties and attributes

Basic components we manipulate when it comes to DOM elements are the properties and attributes assigned to those elements. These are initially assigned to Javascript object instances that represent the DOM elements.

**Properties** are intrinsic to Javascript objects and each has a name and a value.

**Attributes** aren’t a native Javascript concept, but one that only applies to DOM elements. Attributes represent the values that are specified on the markup of DOM elements.

Consider the following:

<img id=”myimage” src=”image.jpg” alt=”an image” class=”someClass” title=”this is an image”/>

Img element attributes : id, src, alt, class, title

A Javascript object instance can represent this DOM element. Attributes are gathered into a list and stored as a property called attributes. In addition to storing the attributes in this list, the object is given a number of properties, including some that represent the attributes of the element’s markup. As such, the attribute values are reflected not only in the attributes list but also in a handful of properties.

### Manipulating element properties

Use native Javascript notation to obtain or modify properties of elements. The trick is getting the element references in the first place:

* Array indexing on wrapped set. $(selector)[n]
* Get() to return individual element by index or toArray() which returns an array of entire set of elements
* Each() or map() where individual elements are available in the callback functions
* Eq() method or :eq() filter
* Via callback functions to some methods ( not(), filter() ) that set elements as function context of callback

See SettingElementProperties.html

### Fetching attribute values

attr( attribute name ) obtains value assigned to specified attribute for first element in matched set.

Why deal with attributes at all when accessing properties is so easy? Attr() is more than a wrapper around Javascript getAttribute() and setAttribute() methods. JQuery provides access to some commonly used properties that have been a thorn in the side due to browser dependency

Eg: cellspacing, class, colspan, cssFloat etc

Also, the write variant of attr() has some of it’s own handy features …

### Setting attribute values

attr(name,value) sets named attribute to the passed value for all elements in the wrapped set.

Value parameter can be a function reference that is invoked for each element in wrapped set.  
Two parameters are passed to the function:

* Zero based index of element in wrapped set
* Current value of named attribute

Current element is also established as function context (this)

attr( attributes ) uses properties and values specified by passed object to set corresponding attributes onto all elements of matched set

See SettingAttributeValues.html

### Removing attributes

removeAttr(name)

Note that removing an attribute doesn’t remove any corresponding property from Javascript DOM element though it’s value may change.

### Storing custom data on elements

data(name,value)  
data(name)  
removeData(name)

## Changing element styling

We can style an element using existing style sheets or we can operate on the DOM element itself, applying styles directly.

### Adding and removing class names

Each element can be assigned any number of class names using a space delimited string:

<div class=”someClass anotherClass yetAnotherClass”></div>

DOM elements className property stores these classes as the same space delimited string as opposed to an array of names – cumbersome! JQuery makes this easy for us though.

addClass(names) adds class to all elements in wrapped set  
removeClass(names) removes class from each element in wrapped set  
toggleClass(names) adds or removes the class depending on presence of class  
toggleClass(names,switch) adds class name if switch evaluates to true and removes class name if switch evaluates to false.  
hasClass(name) determines if any element of matched set possesses the passed class name

### Getting and setting styles

css(name) retrieved value of css property for first element in wrapped set

css(name,value) sets css style property to specified value for each matched element. Value can be a function. See attr()

css(properties) sets css properties specified as keys in passed object:

$(‘<img>’,  
{ src:’myimage.jpg’  
})  
.css({  
 cursor:’pointer’,  
 border: ‘1px solid black’  
})

## Setting element content

### Replacing HTML or text content

Html() obtains HTML content of first element of matched set  
html(content) sets passed HTML fragment as content of all matched elements  
text() concatenates all text content of wrapped elements and returns it

### Moving and copying elements

Append(content) appends passed HTML fragment to content of all matched elements.  
prepend(content) prepends …  
before(content)  
after(content)

### Wrapping and unwrapping elements

We may want to wrap an element or series of elements in some markup

Wrap(wrapper) wraps elements of matched set with passed HTML tags or clone of passed element

$(‘a’).wrap(‘<div></div>’) wraps all links in a div  
$(‘a’).wrap( $(‘div:first’)[0] ) wraps all links in a clone of first div on page

The wrap() method works on each individual method of wrapped set. We may want to wrap all these elements as a single unit.

wrapAll(wrapper) achieves this

wrapInner(), unwrap()

### Removing elements

Remove(selector) removes elements in wrapped set from DOM

Note that the elements that have been removed from the DOM are still referenced by the wrapped set and can still be operated upon. Any jQuery data or events bound to the elements are also removed.

Detach(selector) removes elements in wrapped set from DOM but retains and bound events and jQuery data.

Empty() completely empties DOM elements of their contents

### Cloning elements

We can make copies of elements to attach elsewhere in the DOM tree.

Clone(copyHandlers) creates copies of elements of wrapped set. copyHandlers is a Boolean value that specifies if event handlers should also be copied

### Replacing elements

replaceWith(content) replaces each matched element with the specified content  
replaceAll(selector) replaces each element matched by passed selector with the content of the matched set to which this method is applied.

## Dealing with form element values

Form elements have special properties so jQuery contains a number of convenience functions.

Val() returns value attribute of first element in matched set. An array of selections is returned for a multi select element.

Val(value) sets the passed value as the value of all matched form elements

Val(values) causes any checkboxes, radiobuttons or options of <select> elements to become checked or selected if their value properties match any of the values passed in the values array

# Events

## The DOM Level 0 Event Model

Employed by most web developers. Event handlers are declared by assigning a reference to a function instance to properties of DOM elements.

See dom.0.events.html

### The event instance

When an event handler is fired, an instance of a class named Event is passed to the handler as it’s first parameter in most browsers. The properties of the event instance provide a great deal of information regarding the event that has been fired and is currently being handled.

IE is different to most browsers.

### Event bubbling

Any events are propagated to ancestors of original event target

See dom.0.propagation.html

### Affecting event propagation and semantic actions

We may want to prevent an event from bubbling up the DOM tree.

Standards compliant browsers : event.stopPropagation()

IE : event.cancelBubble=true

Under DOM Level 0 event model, almost every step we take in an event handler involves some browser specific detection.

## The DOM Level 2 Event Model

In DOM Level 0, only one event handler per element can be registered for any specific event type at a time:

someElement.onClick = doFirstThing  
someElement.onClick=doSecondThing (overwrites doFirstThing)

### Establishing Event Handlers

Event handlers (also termed listeners) are established via an element method.

Element.addEventListener(eventType, listener, useCapture)

Eg.   
var element = $(‘#example’)[0]  
element.addEventListener( ‘click’,  
 function(event) { … },  
 false )

eventType name is same as DOM level 0 event name without the ‘on’ prefix  
listener is a reference to a function or an inline function  
useCapture is a Boolean related to event propagation

See dom.2.events.html

Note that the order of the events firing cannot be guaranteed

### Event Propagation

When an event is triggered, the event first propagates from the root of the DOM tree down to the target element (capture phase) and then propagates again from the target element up to the DOM root (bubble phase).

See dom.2.propagation.html

Capture handlers hardly ever used in web pages. Main reason is that IE doesn’t support DOM Level 2 event model. It does have a proprietary model corresponding to bubble phase but doesn’t support capture phase.

jQuery makes browser disparities easier to handle

## The jQuery Event Model

* Provides a unified method for establishing event handlers
* Allows multiple handlers for each event type on each element
* Use standard event-type names: eg, click or mouseover
* Makes the Event instance available as a parameter to the handlers
* Normalizes the Event instance for most-often-used properties
* Provides unified methods for event cancelling and default action blocking

Event handlers on DOM elements established with bind() method

See jquery.events.html

In addition to the bind() method, jQuery provide a handful of shortcut methods to establish specific event handlers.

eventTypeName(listener)

|  |  |  |  |
| --- | --- | --- | --- |
| Blur | Focusin | Mousedown | Mouseup |
| Change | Focusout | Mouseenter | Ready |
| Click | Keydown | Mouseleave | Resize |
| Dblclick | Keypress | Mousemove | Scroll |
| Error | Keyup | Mouseout | Select |
| Focus | Load | Mouseover | submit |

### Removing event handlers

Unbind()

### Inspecting the event instance

Event instance has different properties across browsers.

jQuery actually passes an object of type jQuery.Event. Almost indistinguishable from original Event instance. It holds a set of normalized values and methods we can use independently of containing browser.

#### Browser-independent jQuery.Event properties

|  |  |
| --- | --- |
| Name | Description |
| **Properties** |  |
| altKey | Set to true if the Alt key was pressed when the event was triggered, false if not. |
| ctrlKey | Set to true if the Ctrl key was pressed when the event was triggered, false if not |
| currentTarget | The current element during the bubble phase. This is the same object that’s set as the function context of the event handler |
| data | The value, if any, passed as the second parameter to the bind() method when the handler was established |
| metaKey | Set to true if the Meta key was pressed when the event was triggered, false if not. The Meta key is the Ctrl key on PCs and the Command key on Macs |
| pageX | For mouse events, specifies the horizontal coordinate of the event relative to the page origin |
| pageY | For mouse events, specifies the vertical coordinate of the event relative to the page origin |
| relatedTarget | For mouse movement events, identifies the element that the cursor left or entered when the event was triggered |
| screenX | For mouse events, specifies the horizontal coordinate of the event relative to the screen origin |
| screenY | For mouse events, specifies the vertical coordinate of the event relative to the screen origin |
| shiftKey | Set to true if the Shift key was pressed when the event was triggered, false if not |
| result | The most recent non-undefined value returned from a previous event handler |
| target | Identifies the element for which the event was triggered |
| timestamp | The timestamp, in milliseconds, when the jQuery.Event instance was created |
| type | For all events, specifies the type of event that was triggered |
| which | For keyboard events, specifies the numeric code for the key that caused the event; for mouse events, specifies which button was pressed. |
| **Methods** |  |
| preventDefault() | Prevents any default semantic action (such as form submission, link redirection, checkbox state change, and so on) from occurring |
| stopPropagation() | Stops any further propagation of the event up the DOM tree. Additional events on the current target aren’t affected. |
| stopImmediatePropagation() | Stops all further event propagation including additional events on the current target |
| isDefaultPrevented() | Returns true if the preventDefault() method has been called on this instance |
| isPropagationStopped() | Returns true if the stopPropagation() method has been called on this instance |
| isImmediatePropagationStopped() | Returns true if the stopImmediatePropagation() method has been called on this instance |

### Proactively managing event handlers

DOM elements may come into and out of existence during the lifetime of a page. The ready handler isn’t going to be of much help in managing the event handlers for these dynamic elements that don’t exist when the ready handler is executed. We can manage event handlers on the fly as we use jQuery to manipulate the DOM, but we can keep all the event management code together in one place.

#### Setting up “live” event handling

live(eventType, data, listener)  
Causes the passed listener to be invoked as a handler whenever an event identified by the event type occurs on any element that matches the selector used to create the wrapped set, regardless of whether those elements exist or not when this method is called.

Eg. $(‘div’).live( ‘click’, function(event) {…} )

#### Removing “live” event handling

die(eventType,listener)  
Removes live event handlers established by live() and prevents the handler from being invoked on any future elements that may match the selector used for the call to live()

### Triggering event handlers

We may want to trigger execution of a handler under script control.

trigger(eventType, data)  
Invokes any event handlers established for the passed event type for all matched elements

triggerHandler(eventType,data)  
Invokes any event handlers established for the passed event type for all matched elements without bubbling, semantic actions, or live events

jQuery also provides convenience methods for triggering most of the event types.

eventName()  
Invokes any event handlers established for the passed event type for all matched elements

See earlier table for a list of supported methods

### Other event-related methods

#### Toggling listeners

toggle(listener1, listener2 …)  
Establishes the passed functions as a circular list of click event handlers on all elements of the wrapped set. The handlers are called in order on each subsequent click event

A common use for this convenience method is to toggle the enabled state of an element back and forth on each odd and even click.

#### Hovering over elements

See hover.html