DZone Refcardz

CONTENTS INCLUDE:

jQuery Selectors

By Bear Bibeault & Yehuda Katz

WHAT ARE JQUERY SELECTORS?

jQuery selectors are one of the most important aspects of the jQuery library. These selectors use familiar CSS syntax to allow page authors to quickly and easily identify any set of page elements to operate upon with the jQuery library methods. Understanding jQuery selectors is the key to using the jQuery library most effectively. This reference card puts the power of jQuery selectors at your very fingertips.

A jQuery statement typically follows the syntax pattern:

```
$(selector).methodName();
```

The selector is a string expression that identifies the set of DOM elements that will be collected into a matched set to be operated upon by the jQuery methods.

Many of the jQuery operations can also be chained:

```
$(selector).method1().method2().method3();
```

As an example, let's say that we want to hide the DOM element with the id value of goAway and to add class name incognito:

```
$('#goAway').hide().addClass('incognito');
```

Applying the methods is easy. Constructing the selector expressions is where the cleverness lies.

Hot Tip

The wrapped set created by the application of a selector can be treated as a JavaScript array for convenience. It is particularly useful to use array indexing to directly access elements within the wrapped set.

For example:

```
var element = $('img')[0];
```

will set the variable **element** to the first element in the matched set.

TYPES OF JQUERY SELECTORS

There are three categories of jQuery selectors: Basic CSS selectors, Positional selectors, and Custom jQuery selectors.

The Basic Selectors are known as "find selectors" as they are used to find elements within the DOM. The Positional and Custom Selectors are "filter selectors" as they filter a set of elements (which defaults to the entire set of elements in the DOM).

Basic CSS Selectors

These selectors follow standard CSS3 syntax and semantics.

Syntax	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 a sibling of tag name E.
E~F	Matches all elements with tag name F that are preceded by any sibling of tag name E.
E:has(F)	Matches all elements with tag name E that have at least one descendant with tag name F.
E.c	Matches all elements E that possess a class name of c. Omitting E is identical to *.c.
E#i	Matches all elements E that possess an id value of i. Omitting E is identical to *#i.
E[a]	Matches all elements E that posses an attribute a of any value.
E[a=v]	Matches all elements E that posses an attribute a whose value is exactly v.
E[a^=v]	Matches all elements E that posses an attribute a whose value starts with v.
E[a$=v]	Matches all elements E that posses an attribute a whose value ends with v.
E[a*=v]	Matches all elements E that posses an attribute a whose value contains v.

Examples

- $('div') selects all elements
- $('fieldset a') selects all <a> elements within <fieldset> elements

→

Basic CSS Selectors, continued

Examples

- `$('li>p')` selects all <p> elements that are direct children of <li> elements
- `$('div~p')` selects all elements that are preceded by a <p> element
- `$('p:has(b)')` selects all <p> elements that contain a <b> element
- `$('div.someClass')` selects all elements with a class name of `someClass`
- `$('.someClass')` selects all elements with class name someClass
- `$('#testButton')` selects the element with the `id` value of `testButton`
- `$('img[alt]')` selects all <img> elements that possess an `alt` attribute
- `$('a[href$=.pdf]')` selects all <a> elements that possess an `href` attribute that ends in `.pdf`
- `$('button[id*=test]')` selects all buttons whose `id` attributes contain `test`

Hot Tip

You can create the union of multiple disparate selectors by listing them, separated by commas, in a single call to `$()`. For example, the following matches all ** ** and **<p>** elements:

```
$('div,p')
```

While the following, matches all ** ** elements with a **`title`** attribute, *and* all **<img>** elements with **`alt`** attributes:

```
$('div[title],img[alt]')
```

Positional Selectors

These selectors match based upon positional relationships between elements. These selectors can be appended to any base selector (which we'll denote by B) to filter the matches based upon position. If B is omitted, it is assumed to be * (the pool of all elements).

Syntax	Description
B:first	Selects the first element on the page matching the base selector B.
B:last	Selects the last element on the page matching the base selector B.
B:first-child	Selects all elements from B that are first children.
B:last-child	Selects all elements from B that are last children.
B:only-child	Selects all elements from B that are only children.
B:nth-child(n)	Selects all elements from B that are n-th ordinal children. Starts at 1.
B:nth-child(odd|even)	Selects all elements from B that are even or odd ordinal children. The first child is considered odd (ordinal 1).
B:nth-child(Xn+Y)	Selects all elements from B that match the formula. X denotes an ordinal multiplier, while Y denotes an offset. Y may be omitted if 0. See the following examples.
B:even	Selects the even elements within the set of elements defined by B.

Positional Selectors, continued

Syntax	Description
B:odd	Selects the odd elements within the set of elements defined by B.
B:eq(n)	Selects the n-th element within the set of elements defined by B. Starts at 0.
B:gt(n)	Selects elements within the set of elements defined by B that follow the n-th element (exclusive). Starts at 0.
B:lt(n)	Selects elements within the set of elements defined by B that precede the n-th element (exclusive). Starts at 0.

Examples

- `$('p:first')` selects the first <p> element on the page
- `$('img[src$=.png]:first')` selects the first <img> element on the page that has a `src` attribute ending in `.png`
- `$('button.small:last')` selects the last <button> element on the page that has a class name of `small`
- `$('li:first-child')` selects all <li> elements that are first children within their lists
- `$('a:only-child')` selects all <a> elements that are the only element within their parent
- `$('li:nth-child(2)')` selects all <li> elements that are the second item within their lists
- `$('tr:nth-child(odd)')` selects all odd <tr> elements within a table
- `$('div:nth-child(5n)')` selects every 5th element
- `$('div:nth-child(5n+1)')` selects the element after every 5th element
- `$('.someClass:eq(1)')` selects the second element with a class name of `someClass`
- `$('.someClass:gt(1)')` selects all but the first two elements with a class name of `someClass`
- `$('.someClass:lt(4)')` selects the first four elements with a class name of `someClass`

Hot Tip

Note that the **`:nth-child`** selectors begin counting at 1, while the **`:eq`**, **`:gt`** and **`:lt`** selectors begin with 0.

jQuery Custom Selectors

These selectors are provided by jQuery to allow for commonly used, or just plain handy, selections that were not anticipated by the CSS Specification. Like the Positional Selectors, these selectors filter a base matching set (which we denote with B). Omitting B is interpreted as the set of all elements. These selectors may be combined; see the examples for some powerful selector combinations.

Syntax	Description
B:animated	Selects elements from the base set B that are currently under animated control via one of the jQuery animation methods.
B:button	Selects elements of B that are of any button type; that is: `button`, `input[type=submit]`, `input[type=reset]` or `input[type=button]`.
B:checkbox	Selects elements of B that are of type `input[type=checkbox]`.

jQuery Custom Selectors, continued

Syntax	Description
B:enabled	Selects form elements from B that are in enabled state.
B:file	Selects elements of B that are of type `input[type=file]`.
B:header	Selects elements from B that are of the header types: that is `<h1>` through `<h6>`.
B:hidden	Selects elements of B that are hidden.
B:image	Selects elements of B that are of type `input[type=image]`.
B:input	Selects form input elements from B; that is, `<input>`, `<select>`, `<textarea>` and `<button>` elements.
B:not(f)	Selects elements of B that do not match the filter selector specified by `f`. A filter selector is any selector beginning with : (colon), A base set B cannot be specified as part of `f`.
B:parent	Selects elements of B that are parents of non-empty element children.
B:password	Selects elements of B that are of type `input[type=password]`.
B:radio	Selects elements of B that are of type `input[type=radio]`.
B:reset	Selects elements of B that are of type `input[type=reset]` or `button[type=reset]`.
B:selected	Selects elements of B that are in selected state. Only `<option>` elements posses this state.
B:submit	Selects elements of B that are of type `input[type=submit]` or `button[type=submit]`.
B:text	Selects elements of B that are of type `input[type=text]`.
B:visible	Selects form elements from B that are not hidden.

Examples

- `$('img:animated')` selects all `<img>` elements that are undergoing animation
- `$(':button:hidden')` selects all button type elements that are hidden
- `$('input[name=myRadioGroup]:radio:checked')` selects all radio elements with the name attribute value of myRadioGroup that are checked
- `$(':text:disabled')` selects all text fields that are disabled
- `$('#xyz p :header')` selects all header type elements within `<p>` elements that are within an element with an `id` value of `xyz`. Note the space before `:header` that prevents it from binding directly to the `p`.
- `$('option:not(:selected)')` selects all unselected `<option>` elements
- `$('#myForm button:not(.someClass)')` selects all buttons from the <form> with the `id` of `myForm` that do not possess the class name `someClass`.
- `$('select[name=choices] :selected')` selects the selected `<option>` elements within the `<select>` element named choices.
- `$('p:contains(coffee)')` selects all `<p>` elements that contain the text `coffee`

Used either separately, or in combination, the jQuery selectors give you a great deal of power to easily create a set of elements that you wish to operate upon with the jQuery methods.

MATCHED SET METHODS

While the jQuery selectors give us great flexibility in identifying which DOM elements are to be added to a matched set, sometimes there are match criteria that cannot be expressed by selectors alone. Also, given the power of jQuery method chaining, we may wish to adjust the contents of the matched set **between** method invocations.

For these situations, jQuery provides methods that operate not upon the elements within the matched set, but on the matched set itself. This section will summarize those methods.

Adding New Elements

For adding new elements to a matched set, the `add()` method is provided:

`add(expression)`	
`expression`	(String) A selector expression that specifies the DOM elements to be added to the matched set, or an HTML string of new elements to create and add to the set. (Element) A reference to an existing element to add. (Array) Array of references to elements to add.

The `add()` method returns a **new** matched set that is the union of elements in the original wrapped set and any elements either passed directly as the `expression` argument, or matched by the selector of the `expression` argument.

Consider:

```
$('div').add('p').css('color','red');
```

This statement creates a matched set of all `<div>` elements, then creates a new matched set of the already matched `<div>` elements and all `<p>` elements. The second matched set's elements (all `<div>` and all `<p>` elements) are then given the CSS `color` property of "red".

You may think this is not all that useful because the same could have been achieved with:

```
$('div,p').css('color','red');
```

But now consider:

```
$('div').css('font-weight','bold').add('p').
css('color','red');
```

Here the first created matched set of `<div>` elements is assigned a bold rendition, and then the second matched set, with `<p>` elements added, is colored red.

jQuery chaining (in which the `css()` method returns the matched set) allows us to create efficient statements such as this one that can accomplish a great deal with little in the way of script.

More Examples

```
$('div').add(someElement).css('border','3px solid
pink');
```

```
$('div')
  .add([element1,element2])
  .css('border','3px solid pink');
```

Removing Matched Elements

What if we want to **remove** elements from the matched set? That's the job of the `not()` method:

`not(expression)`	
`expression`	(String) A selector expression that specifies the DOM elements to be removed from the matched set. (Element) A reference to an existing element to remove. (Array) Array of references to elements to remove.

Like `add()`, this method creates and returns a **new** matched set, except with the elements specified by the `expression` argument removed. The argument can be a jQuery selector, or references to elements to remove.

Examples

```
$('body *').css('font-weight','bold')
   .not('p').css('color','red');
```

Makes all body elements bold, then makes all but <p> elements red.

```
$('body *').css('font-weight','bold')
   .not(anElement).css('color','red');
```

Similar to the previous except the element referenced by variable `anElement` is not included in the second set (and therefore not colored red).

Avoid a typical beginner's mistake and never confuse the **`not()`** method, which will remove elements from the matched set, with the **`remove()`** method, which will remove the elements in the matched set from the HTML DOM!

Finding Descendants

Sometimes it's useful to limit the search for elements to descendants of already identified elements. The `find()` method does just that:

`find(expression)`	
`expression`	(String) A selector expression that specifies which descendant elements are to be matched.

Unlike the previously examined methods, `find()` only accepts a selector expression as its argument. The elements within the existing matched set will be searched for descendants that match the expression. Any elements in the original matched set that match the selector are not included in the new set.

Example

```
$('div').css('background-color','blue')
   .find('img').css('border','1px solid aqua');;
```

Selects all elements, makes their background blue, selects all <img> elements that are descendants of those elements (but not <img> elements that are not descendants) and gives them an aqua border.

Filtering Matched Sets

When really fine-grained control is required for filtering the elements of a matched set, the `filter()` method comes in handy:

`filter(expression)`	
`expression`	(String) A selector expression that specifies which elements are to be retained. (Function) A function used to determine if an element should be included in the new set or not. This function is passed the zero-based ordinal of the element within the original set, and the function context **`(this)`** is set to the current element. Returning **`false`** as the function result causes the element to not be included in the new set.

The `filter()` method can be passed either a selector expression (comma-separated if more than one is desired) or a function. When passed a selector, it acts like the inverse of `not()`, retaining elements that match the selector (as opposed to excluding them). When passed a function, the function is invoked for each element and decisions that cannot be expressed by selectors can be made regarding the exclusion or inclusion of each element.

Examples

```
$('.bashful').show()
   .filter('img[src$=.gif]').attr('title','Hi there!');
```

Selects all elements with class name `bashful`, makes sure that they are visible, filters the set down to just GIF images, and assigns a `title` attribute to them.

```
$('img[src^=images/]').filter(function(){
  return $(this).attr('title').match(/.+@.+\.com/)!= null;
   })
   .hide();
```

Selects images from a specific folder, filters them to only those whose `title` attribute matches a rudimentary .com email address, and hides those elements.

Slicing and Dicing Matched Sets

Rather than matching elements by selector, we may sometimes wish to slice up a matched set based upon the position of the elements within the set. This section introduces two methods that do that for us.

Both of these methods assume zero-based indexing.

`slice(being,end)`	
`begin`	(Number) The beginning position of the first element to be included in the new set.
`end`	(Number) The end position of the first element to **not** be included in the new set. If omitted, all elements from `begin` to the end of the set are included.

Examples

```
$('body *').slice(2).hide();
```

Selects all body elements, then creates a new set containing all but the first two elements, and hides them.

```
$('body *').slice(2,3).hide();
```

Selects all body elements, then creates a new set containing the third element in the set and hides it. Note that the new set contains just one element: that at position 2. The element at position 3 is not included.

Slicing and Dicing Matched Sets, continued

eq(position)	
position	(Number) The position of a single element to be included in the new set.

The `eq(n)` method can be considered shorthand for `slice(n,n+1)`.

Matching by Relationship

Frequently we may want to create new matched sets based upon relationships between elements. These methods are similar enough that we'll present them en masse in the following table:

Method	Description
children(expression)	Creates a new matched set containing all unique children of the elements in the original matched set that match the optional expression.
next(expression)	Creates a new matched set containing unique following (next) siblings of the elements in the original matched set that match the optional expression. Only immediately following siblings are returned.
nextAll(expression)	Creates a new matched set containing unique following (next) siblings of the elements in the original matched set that match the optional expression. All following siblings are returned.
parent(expression)	Creates a new matched set containing unique immediate parents of the elements in the original matched set that match the optional expression.
parents(expression)	Creates a new matched set containing all ancestors of the elements in the original matched set that match the optional expression.
prev(expression)	Creates a new matched set containing unique preceding siblings of the elements in the original matched set that match the optional expression. Only immediately preceding siblings are returned.
prevAll(expression)	Creates a new matched set containing unique preceding siblings of the elements in the original matched set that match the optional expression. All preceding siblings are returned.
siblings(expression)	Creates a new matched set containing unique siblings of the elements in the original matched set that match the optional expression.
contents()	Creates a new matched set containing all unique children of the elements in the original matched set including text nodes. When used on an <iframe>, matches the content document.

For all methods that accept a filtering expression, the expression may be omitted in which case no filtering occurs.

Translating Elements

There may be times that you want to translate the elements within a matched set to other values. jQuery provides the `map()` method for this purpose.

map(callback)	
callback	(Function) A callback function called for each element in the matched set. The return values of the invocations are collected into an array that is returned as the result of the `map()` method. The current element is set as the function context `(this)` for each invocation.

For example, let's say that you wanted to collect the values of all form elements within a form named myForm:

```
var values = $('#myForm :input').map(function(){
  return $(this).val();
});
```

Hot Tip

The map() function returns a jQuery object instance. To convert this to a normal JavaScript array, you can use the get() method without parameters:

```
var values = $('#myForm :input').map(function(){
  return $(this).val();
}).get();
```

In this case, **values** references a JavaScript array rather than a jQuery wrapped object.

Controlling Chaining

All of the methods examined create **new** matched sets whose contents are determined in the manner explained for each method. But what happens to the original? Is it dismissed?

It is *not*. When a new wrapped set is created it is placed on the top of a `stack` of sets, with the top-most set being the one to which any methods will be applied (as we have seen in the examples). But jQuery allows you to "pop" the top-most set off that stack so that you can apply methods to the original set. It does this with the `end()` method:

end()	
(no arguments)	

Consider a previous example:

```
$('div').add('p').css('color','red');
```

As we recall, this creates a matched set of `<div>` elements, then creates a new set that also contains the `<p>` elements. Since this latter set is at the top of the stack when the `css()` method is called, it is the second set that is affected. Now consider:

```
$('div').add('p').css('color','red').end().hide();
```

After the `css()` method is called, the `end()` method pops the second set off the stack "exposing" the original set of just `<div>` elements, which are then hidden.

Another useful method to affect how chaining the sets operates is the `andSelf()` method:

andSelf()	
(no arguments)	

Calling `andSelf()` creates yet another new matched set that is the union of the top two matched sets on the stack. This can be useful for performing an action on a set, creating a new

→

Controlling Chaining, continued

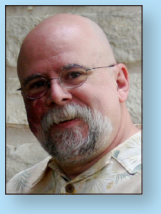
distinct set, and then applying a method (or methods) to them all. Consider:

```
$('div').css('background-color','yellow')
   .children('img').css('border','4px ridge maroon')
   .andSelf().css('margin','4em');
```

All elements are selected and their background set to yellow. Then, the <img> children of those elements are selected and have a border applied. Finally, the two sets are merged, and a wide margin is applied to all elements and their <img> children.

Between jQuery selectors and the jQuery methods that allow us to manipulate the matched sets, we can see that jQuery gives us some powerful tools to select the DOM elements that we can then operate upon with the many jQuery methods (as well as the dozens and dozens of jQuery plugins) that are available to us.

ABOUT THE AUTHORS



Bear Bibeault

Bear Bibeault has been writing software for over three decades, starting with a Tic-Tac-Toe program written on a Control Data Cyber supercomputer via a 100-baud teletype. He is a Software Architect and Technical Manager for a company that builds and maintains a large financial web application used by the accountants that many of the Fortune 500 companies keep in their dungeons. He also serves as a "sheriff" at the popular JavaRanch.com.

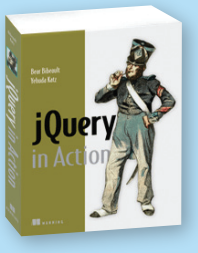
Publications: *jQuery in Action, Ajax in Practice, Prototype and Scriptaculous in Action* (Manning)
Notable Projects: "Sheriff" at **JavaRanch.com**, FrontMan Web Application Controller

Yehuda Katz

Yehuda Katz has been involved in a number of open-source projects over the past several years. In addition to being a core team member of the jQuery project, he is also a core member of Merb, an alternative to Ruby on Rails (also written in Ruby). He speaks about jQuery and Ruby at a number of regional conferences, and is the JavaScript expert on the Merb team. He recently joined EngineYard working on the Merb project full-time.

Publication: *jQuery in Action* (Manning)
Notable Projects: **Visual jQuery.com**, jQuery Plugin Coordinator, Merb, DataMapper ORM
Web site: **www.yehudakatz.com**

RECOMMENDED BOOK



jQuery in Action is a fast-paced introduction and guide to the jQuery library. It shows you how to traverse HTML documents, handle events, perform animations, and add Ajax to your web pages using jQuery. You learn how jQuery interacts with other tools and how to build jQuery plugins.

BUY NOW
books.dzone.com/books/jquery-in-action

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Core Seam
Core CSS: Part III
Hibernate Search
Equinox
EMF
XML
JSP Expression Language
ALM Best Practices
HTML and XHTML

Available:

Essential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor
RSS and Atom
GlassFish Application Server
Silverlight 2

Visit **refcardz.com** for a complete listing of available Refcardz.

Design Patterns
Published June 2008

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513
888.678.0399
919.678.0300
Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-06-6
ISBN-10: 1-934238-06-6
50795
9 781934 238066
$7.95

Version 1.0