2
Selector Expressions

Each action we perform with jQuery requires a target. For example, in order to hide or show an element on the page, first we must find that element. To do so, we rely on jQuery's **selector expressions**.

Borrowing from CSS 1–3 and then adding its own, jQuery offers a powerful set of tools for matching a set of elements in a document. In this chapter, we'll examine every selector expression that jQuery makes available in turn.

CSS selectors

The following selectors are based on the Cascading Style Sheet specifications (1–3), as outlined by the W3C. For more information about the specifications, visit `http://www.w3.org/Style/CSS/#specs`.

Element (T)

Select all elements that have a tag name of `T`.

Examples

- `$('div')` selects all elements with a tag name of `div` in the document
- `$('em')` selects all elements with a tag name of `em` in the document

Description

JavaScript's `getElementsByTagName()` function is called to return the appropriate elements when this expression is used.

ID (#myid)

Select the unique element with an ID equal to `myid`.

Examples

- `$('#myid')` selects the unique element with `id="myid"`
- `$('p#myid')` selects a single paragraph with an ID of `myid`; in other words, the unique element `<p id="myid">`

Description

Each ID value must be used only once within a document. If more than one element has been assigned the same ID, queries that use that ID will only select the *first* matched element in the DOM. However, this behavior should not be relied on. A document with more than one element using the same ID is invalid.

In our second example, it might not be immediately clear why someone might want to specify a tag name associated with a particular ID, as that `id` value needs to be unique anyway. However, some situations in which parts of the DOM are user-generated may require a more specific expression to avoid false positives. Furthermore, when the same script is run on more than one page, it might be necessary to identify the element ID as the pages could be associating the same ID with different elements. For example, page A might have `<h1 id="title">` while page B has `<h2 id="title">`.

For ID selectors such as the preceding examples, jQuery uses the JavaScript function `getElementById()`, which is extremely efficient. When another selector is attached to the ID selector, like in the second example, jQuery performs an additional check before identifying the element as a match.

As always, remember that as a developer, your time is typically the most valuable resource. Do not focus on optimization of selector speed unless it is clear that performance needs to be improved.

Class (.myclass)

Select all elements that have a class of `myclass`.

Examples

- `$('.myclass')` selects all elements that have a class of `myclass`
- `$('p.myclass')` selects all paragraphs that have a class of `myclass`
- `$('.myclass.otherclass')` selects all elements that have a class of both `myclass` and `otherclass`

Description

For class selectors, jQuery uses JavaScript's native `getElementsByClassName()` function if the browser supports it. Otherwise, it checks the `.className` attribute of each element.

As a CSS selector (for example, in a stylesheet), the multiple-class syntax used in the third example is supported by all modern web browsers, but *not* by Internet Explorer versions 6 and below. This makes the syntax especially handy for applying styles cross-browser through jQuery.

Descendant (E F)

Select all elements matched by `F` that are descendants of an element matched by `E`.

Examples

- `$('#container p')` selects all paragraph elements that are descendants of an element that has an ID of `container`
- `$('a img')` selects all `<img>` elements that are descendants of an `<a>` element

Description

Descendants of an element are that element's children, grandchildren, great-grandchildren, and so on. For example, in the following HTML code, the `<img>` element is a descendant of the `<span>`, `<p>`, `<div id="inner">`, and `<div id="container">` elements:

```
<div id="container">
  <div id="inner">
    <p>
      <span><img src="example.jpg" alt="" /></span>
    </p>
  </div>
</div>
```

Child (E > F)

Select all elements matched by `F` that are children of an element matched by `E`.

Examples

- `$('li > ul')` selects all `<ul>` elements that are children of an `<li>` element
- `$('.myclass > code')` selects all `<code>` elements that are children of an element with the class `myclass`

Description

As a CSS selector, the child combinator is supported by all modern web browsers including Safari, Mozilla/Firefox, Opera, Chrome, and Internet Explorer 7 and above; but notably *not* by Internet Explorer versions 6 and below. The first example is a handy way of selecting all nested unordered lists (that is, except the top level), and jQuery makes it possible to do this in a cross-browser fashion.

The child combinator (E > F) can be thought of as a more specific form of the descendant combinator (E F) in that it selects only first-level descendants. Therefore, in the following HTML code, the `<img>` element is a child only of the `<span>` element:

```
<div id="container">
  <div id="inner">
    <p>
      <span><img src="example.jpg" alt="" /></span>
    </p>
  </div>
</div>
```

Adjacent sibling (E + F)

Select all elements matched by `F` that *immediately* follow and have the same parent as an element matched by `E`.

Examples

- `$('ul + p')` selects all `<p>` elements that immediately follow a sibling `<ul>` element
- `$('#myid + .myclass')` selects the element with `class="myclass"` that immediately follows a sibling element with `id="myid"`

Description

One important point to consider with both the adjacent sibling combinator (E + F) and the general sibling combinator (E ~ F, covered next) is that they only select *siblings*. Consider the following HTML code:

```
<div id="container">
  <ul>
    <li></li>
    <li></li>
  </ul>
  <p>
    <img/>
  </p>
</div>
```

- `$('ul + p')` selects `<p>` because it immediately follows `<ul>` and the two elements share the same parent, `<div id="container">`
- `$('p + img')` selects nothing because `<p>` is one level higher in the DOM tree than `<img>`
- `$('li + img')` selects nothing because even though `<li>` and `<img>` are on the same level in the DOM tree, they do not share the same parent

General sibling (E ~ F)

Select all elements matched by `F` that follow and have the same parent as an element matched by `E`.

Examples

- `$('ul ~ p')` selects all `<p>` elements that follow a sibling `<ul>` element
- `$('#myid ~ .myclass')` selects all elements with `class="myclass"` that follow a sibling element with `id="myid"`

Description

One important point to consider with both the adjacent sibling combinator (E + F) and the general sibling combinator (E ~ F) is that they only select *siblings*. The notable difference between the two is their respective reach. While the former reaches only to the *immediately* following sibling element, the latter extends that reach to *all* of the following sibling elements.

Consider the following HTML code:

```
<ul>
  <li class="first"></li>
  <li class="second"></li>
  <li class="third></li>
</ul>
<ul>
  <li class="fourth"></li>
  <li class="fifth"></li>
  <li class="sixth"></li>
</ul>
```

- `$('li.first ~ li')` selects `<li class="second">` and `<li class="third">`
- `$('li.first + li')` selects `<li class="second">`

Multiple expressions (E, F, G)

Select all elements matched by any of the selector expressions `E`, `F`, or `G`.

Examples

- `$('code, em, strong')` selects all `<code>`, `<em>`, and `<strong>` elements
- `$('p strong, .myclass')` selects all `<strong>` elements that are descendants of a `<p>` element, as well as all elements that have a class of `myclass`

Description

This multiple expression combinator is an efficient way to select disparate elements. An alternative to this combinator is the `.add()` method described in Chapter 3*, DOM Traversal Methods*.

Numbered child (:nth-child(n/even/odd/expr))

Select all elements that are the nth child of their parent.

Examples

- `$('li:nth-child(2)')` selects all `<li>` elements that are the second child of their parent
- `$('p:nth-child(odd)')` selects all `<p>` elements that are an odd-numbered child of their parent (first, third, fifth, and so on)
- `$('.myclass:nth-child(3n+2)')` selects all elements with the class `myclass` that are the (3n+2)th child of their parent (second, fifth, eighth, and so on)

Description

As jQuery's implementation of `:nth-child(n)` is strictly derived from the CSS specification, the value of `n` is "1-based," meaning that the counting starts at 1. However, for all other selector expressions, jQuery follows JavaScript's "0-based" counting. Therefore, given a single `<ul>` containing two `<li>`s, `$('li:nth-child(1)')` selects the first `<li>` while `$('li:eq(1)')` selects the second.

As the two look so similar, the `:nth-child(n)` pseudo-class is easily confused with `:nth(n)`, a synonym for `:eq(n)`. However, as we have just seen, the two can result in dramatically different matched elements. With `:nth-child(n)`, all children are counted, regardless of what they are, and the specified element is selected only if it matches the selector attached to the pseudo-class. With `:nth(n)`, only the selector attached to the pseudo-class is counted, not limited to children of any other element, and the nth one is selected. To demonstrate this distinction, let's examine the results of a few selector expressions given the following HTML code:

```
<div>
  <h2></h2>
  <p></p>
  <h2></h2>
  <p></p>
  <p></p>
</div>
```

- `$('p:nth(1)')` selects the second `<p>` because numbering for `:nth(n)` starts with `0`
- `$('p:nth-child(1)')` selects nothing because there is no `<p>` element that is the first child of its parent
- `$('p:nth(2)')` selects the third `<p>`
- `$('p:nth-child(2)')` selects the first `<p>` because it is the second child of its parent

In addition to taking an integer, `:nth-child(n)` can take `even` or `odd`. This makes it especially useful for table-row striping solutions when more than one table appears in a document. Again, given the preceding HTML snippet:

- `$('p:nth-child(even)')` selects the first and third `<p>` because they are children 2 and 4 (both even numbers) of their parent

The third example selector illustrates the most complicated usage of `:nth-child()`. When a simple mathematical expression of the form `an+b` is provided with integers substituted for `a` and `b`, `:nth-child()` selects the elements whose positions in the list of children are equal to `an+b` for some integer value of `n`.

Further description of this unusual usage can be found in the W3C CSS specification at `http://www.w3.org/TR/css3-selectors/#nth-child-pseudo`.

First child (:first-child)

Select all elements that are the first child of their parent element.

Examples

- `$('li:first-child')` selects all `<li>` elements that are the first child of their parent element
- `$(.myclass:first-child')` selects all elements with the class `myclass` that are the first child of their parent element

Description

The `:first-child` pseudo-class is shorthand for `:nth-child(1)`.
For more information on `:X-child` pseudo-classes, see the *Description* for the *Numbered child* selector.

Last child (:last-child)

Select all elements that are the last child of their parent element.

Examples

- `$('li:last-child')` selects all `<li>` elements that are the last child of their parent element
- `$(.myclass:last-child')` selects all elements with the class `myclass` that are the last child of their parent element

Description

For more information on `:X-child` pseudo-classes, see the *Description* for the *Numbered child* selector.

Only child (:only-child)

Select all elements that are the only child of their parent element.

Examples

- `$(':only-child')` selects all elements in the document that are the only child of their parent element
- `$('code:only-child')` selects all `<code>` elements that are the only child of their parent element

Not (:not(E))

Select all elements that do not match the selector expression `E`.

Examples

- `$(':not(.myclass)')` selects all elements in the document that do not have the class `myclass`
- `$('li:not(:last-child)')` selects all `<li>` elements that are not the last child of their parent elements

Empty (:empty)

Select all elements that have no children (including text nodes).

Examples

- `$(':empty')` selects all elements in the document that have no children
- `$('p:empty')` selects all `<p>` elements that have no children

Description

One important thing to note with `:empty` (and `:parent`) is that *child elements include text nodes*.

The W3C recommends that the `<p>` element should have at least one child node, even if that child is merely text (see `http://www.w3.org/TR/html401/struct/text.html#edef-P`). On the other hand, some other elements are empty (that is, have no children) by definition; for example, `<input>`, `<img>`, `<br>`, and `<hr>`.

Universal (*)

Select all elements.

Examples

- `$('*')` selects all elements in the document
- `$('p > *')` selects all elements that are children of a paragraph element

Description

The `universal` selector is especially useful when combined with other expressions to form a more specific selector expression.

Attribute selectors

The CSS specification also allows elements to be identified by their attributes. While not widely supported by browsers for the purpose of styling documents, these attribute selectors are highly useful and jQuery allows us to employ them regardless of the browser being used.

When using any of the following attribute selectors, we should account for attributes that have multiple, space-separated values. As these selectors see attribute values as a single string, `$('a[rel=nofollow]')`; for example, will select `<a rel="nofollow" href="example.html">Some text</a>` but *not* `<a rel="nofollow self" href="example.html">Some text</a>`.

Attribute values in selector expressions can be written as bare words or surrounded by quotation marks. Therefore, the following variations are equally correct:

- Bare words: `$('a[rel=nofollow self]')`
- Double quotes inside single quotes: `$('a[rel="nofollow self"]')`
- Single quotes inside double quotes: `$("a[rel='nofollow self']")`
- Escaped single quotes inside single quotes: `$('a[rel=\'nofollow self\']')`
- Escaped double quotes inside double quotes: `$("a[rel=\"nofollow self\"]")`

The variation we choose is generally a matter of style or convenience. The authors choose to omit quotation marks in this context for clarity, and the examples that follow reflect this preference.

Attribute ([foo])

Select all elements that have the `foo` attribute, with any value.

Examples

- `$('[rel]')` selects all elements that have a `rel` attribute
- `$('.myclass[style]')` selects all elements with the class `myclass` that have a `style` attribute

Attribute equals ([foo=bar])

Select all elements that have the `foo` attribute with a value exactly equal to `bar`.

Examples

- `$('[name=myname]')` selects all elements that have a `name` value exactly equal to `myname`
- `$('a[rel=nofollow]')` selects all `<a>` elements that have a `rel` value exactly equal to `nofollow`

Description

For more information on this attribute selector, see the introduction to *Attribute selectors*.

Attribute does not equal ([foo!=bar])

Select all elements that do *not* have the `foo` attribute, or have a `foo` attribute but with a value other than `bar`.

Examples

- `$('a[rel!=nofollow]')` selects all `<a>` elements that either have no `rel` attribute, or have one with a value other than `nofollow`
- `$('input[name!=myname]')` selects all `<input>` elements that either have no `name` attribute, or have one with a value other than `myname`

Description

As these selectors see attribute values as a single string, the first example *will* select `<a rel="nofollow self" href="example.htm">Some text</a>`. Consider the *Attribute contains word* selector for alternatives to this behavior.

Attribute begins with ([foo^=bar])

Select all elements that have the `foo` attribute with a value *beginning* exactly with the string `bar`.

Examples

- `$('[id^=hello]')` selects all elements that have an ID beginning with `hello`
- `$('input[name^=my]')` selects all `<input>` elements that have a `name` value beginning with `my`

Description

This selector can be useful for identifying elements in pages produced by server-side frameworks that produce HTML with systematic element IDs.

Attribute ends with ([foo$=bar])

Select all elements that have the `foo` attribute with a value *ending* exactly with the string `bar`.

Examples

- `$('[id$=goodbye]')` selects all elements that have an ID ending with `goodbye`
- `$('input[name$=phone]')` selects all `<input>` elements that have a `name` value ending with `phone`

Attribute contains ([foo*=bar])

Select all elements that have the `foo` attribute with a value *containing* the substring `bar`.

Examples

- `$('[style*=background]')` selects all elements that have a `style` value containing `background`
- `$('a[href*=example.com]')` selects all `<a>` elements that have an `href` value containing `example.com`

Description

This is the most generous of the jQuery attribute selectors that match against a value. It will select an element if the selector's string appears anywhere within the element's attribute value. Therefore, `$('p[class*=my]')` will select `<p class="yourclass myclass">Some text</p>`, `<p class="myclass yourclass">Some text</p>` and `<p class="thisismyclass">Some text</p>`.

Compare this selector with the *Attribute contains word* selector, which is more appropriate in many cases.

Attribute contains word ([foo~=bar])

Select all elements that have the `foo` attribute with a value containing the word `bar`, delimited by spaces.

Examples

- `$('[class~=myclass]')` selects all elements that have the class of `myclass` (and optionally other classes as well).
- `$('a[rel~=nofollow]')` selects all `<a>` elements with a `rel` value including `nofollow`.

Description

This selector matches the test string against each word in the attribute value, where a "word" is defined as a string delimited by whitespace. The selector matches if the test string is exactly equal to any of the words. Thus, the first example is equivalent to `$('.myclass')`.

This selector is similar to the *Attribute contains* selector, but substring matches within a word do not count. Therefore, the second example matches `<a rel="nofollow" href="example.html">Some text</a>` as well as `<a rel="nofollow self" href="example.html">Some text</a>`, but does not match `<a rel="nofollowself" href="example.html">Some text</a>`.

Attribute contains prefix ([foo|=bar])

Select all elements that have the `foo` attribute with a value either equal to `bar`, or beginning with `bar` and a hyphen (`-`).

Examples

- `$('[id|=hello]')` selects all elements with an ID of `hello` or an ID that begins with `hello-`
- `$('a[hreflang|=en]')` selects all `<a>` elements with an `hreflang` value of `en` or beginning with `en-`

Description

This selector was introduced into the CSS specification to handle language attributes. The second example, for instance, matches `<a href="example.html" hreflang="en">Some text</a>` as well as `<a href="example.html" hreflang="en-UK">Some text</a>`.

Form selectors

The following selectors can be used to access form elements in a variety of states. When using any of the form selectors other than `:input`, providing a tag name as well is recommended (for example, `input:text` rather than `:text`).

- **Form element (:input)**: Select all form elements (`<input>` (all types), `<select>`, `<textarea>`, `<button>`)
- **Text field (:text)**: Select all text fields (`<input type="text">`)
- **Password field (:password)**: Select all password fields (`<input type="password">`)
- **Radio button (:radio)**: Select all radio button fields (`<input type="radio">`)
- **Checkbox (:checkbox)**: Select all checkbox fields (`<input type="checkbox">`)
- **Submit button (:submit)**: Select all submit inputs and button elements (`<input type="submit">`, `<button>`)
- **Image button (:image)**: Select all image inputs (`<input type="image">`)
- **Reset button (:reset)**: Select all reset buttons (`<input type="reset">`)
- **Standard button (:button)**: Select all button elements and input elements with a type of `button` (`<button>`, `<input type="button">`)
- **File upload (:file)**: Select all file upload fields `(<input type="file">)`
- **Enabled form element (:enabled)**: Select all form elements that are enabled (that is, they do not have the `disabled` attribute and users can interact with them)

- **Disabled form element (:disabled)**: Select all form elements that are disabled (that is, they have the `disabled` attribute and users cannot interact with them)
- **Checked box (:checked)**: Select all form elements—checkboxes and radio buttons—that are checked
- **Selected option (:selected):** Select all form elements (effectively, `<option>` elements) that are currently selected

Custom selectors

The following selectors were added to the jQuery library in an attempt to address common DOM traversal needs not met by the CSS specification.

Element at index (:eq(n))

Select the element at index `n` within the matched set.

Examples

- `$('li:eq(2)')` selects the third `<li>` element
- `$('.myclass:eq(1)')` selects the second element with the class `myclass`

Description

The selector `:nth(n)` exists as a synonym of this selector.

The index-related selector expressions (including this selector and the others that follow) **filter** the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use 0-based indexing, these selectors reflect that fact. This is why `$('.myclass:eq(1)')` selects the second element in the document with the class `myclass`, rather than the first. In contrast, `:nth-child(n)` uses 1-based indexing to conform to the CSS specification.

Greater than (:gt(n))

Select all elements at an index greater than `n` within the matched set.

Examples

- `$('li:gt(2)')` selects all `<li>` elements following the third one
- `$('.myclass:gt(1)')` selects all elements with the class `myclass` following the second one

Description

See the *Description* for *Element at index*, for important details regarding the indexing used by this selector.

Less than (:lt(n))

Select all elements at an index less than `n` within the matched set.

Examples

- `$('li:gt(2)')` selects all `<li>` elements preceding the third one
- `$('.myclass:gt(1)')` selects all elements with the class `myclass` preceding the second one

Description

See the *Description* for *Element at index* for important details regarding the indexing used by this selector.

First (:first)

Select the first element within the matched set.

Examples

- `$('li:first')` selects the first `<li>` element
- `$('.myclass:first')` selects the first element with the class `myclass`

Description

The `:first` pseudo-class is shorthand for `:eq(0)`. It could also be written as `:lt(1)`.

See the *Description* for *Element at index* for important details regarding the indexing used by this selector.

Last (:last)

Select the last element within the matched set.

Examples

- `$('li:last)` selects the last `<li>` element
- `$('.myclass:last)` selects the last element with the class `myclass`

Description

While `:first` has equivalent selectors—`nth(0)` and `eq(0)`—the `:last` pseudo-class is unique in its ability to select only the last element in the set of matched elements.

See the *Description* for *Element at index* for important details regarding the indexing used by this selector.

Even element (:even)

Select all elements with an even index within the matched set.

Examples

- `$('li:even')` selects the even-indexed elements within the set of `<li>` elements
- `$('.myclass:even')` selects the even-indexed elements within the set of elements that have the class `myclass`

Description

See the *Description* for *Element at index* for important details regarding the indexing used by this selector. In particular, note that the 0-based indexing means that, counter-intuitively, `:even` selects the first element, third element, and so on within the matched set.

Odd element (:odd)

Select all elements with an odd index within the matched set.

Examples

- `$('li:odd')` selects the odd-indexed elements within the set of `<li>` elements
- `$('.myclass:odd')` selects the odd-indexed elements within the set of elements that have the class `myclass`

Description

See the *Description* for *Element at index* for important details regarding the indexing used by this selector. In particular, note that the 0-based indexing means that, counter-intuitively, `:odd` selects the second element, fourth element, and so on within the matched set.

Is parent (:parent)

Select all elements that are the parent of another element, including text nodes.

Examples

- `$(':parent')` selects all elements that are the parent of another element, including text nodes
- `$('td:parent')` selects all `<td>` elements that are the parent of another element, including text nodes

Description

One important thing to note regarding use of `:parent` (and `:empty`) is that *child elements include text nodes*.

The W3C recommends that the `<p>` element have at least one child node, even if that child is merely text (see `http://www.w3.org/TR/html401/struct/text.html#edef-P`). On the other hand, some other elements are empty (that is, have no children) by definition; for example, `<input>`, `<img>`, `<br>`, and `<hr>`.

Contains text (:contains(text))

Select all elements that contain the specified text.

Examples

- `$('p:contains(nothing special)')` selects all `<p>` elements that contain the text `nothing special`
- `$('li:contains(second)')` selects all `<li>` elements that contain the text `second`

Description

The matching text can appear directly within the selected element in any of that element's descendants, or a combination thereof. Therefore, the first example would still select the following paragraph:

```
<p>This paragraph is <span>nothing <strong>special</strong></span></p>
```

As with attribute value selectors, text inside the parentheses of `:contains()` can be written as bare words or surrounded by quotation marks. *The text must have matching case to be selected*.

Contains element (:has(E))

Select all elements that contain an element matching `E`.

Examples

- `$('p:has(img)')` selects all `<p>` elements that contain an `<img>` element as a descendant
- `$('.myclass:has(#myid)')` selects all elements with the class `myclass` that contain a descendant with ID `myid`

Description

This expression matches an element if an element matched by `E` exists anywhere among the descendants, and not just the direct children. For example, the first example matches the `<p>` element in the following HTML code:

```
<div id="container">
  <div id="inner">
    <p>
      <span><img src="example.jpg" alt="" /></span>
    </p>
  </div>
</div>
```

Visible (:visible)

Select all elements that are visible.

Examples

- `$('li:visible')` selects all `<li>` elements that are visible
- `$('.myclass:visible')` selects all elements with the class `myclass` that are visible

Description

The `:visible` selector matches items that are currently visible on the page. Rather than relying on the CSS properties assigned to the element, such as its `display` and `visibility`. jQuery determines whether the element is visible by testing its current width and height.

Elements can be considered hidden for several reasons:

- They have a `display` value of `none`
- They are form elements with `type="hidden"`
- Their `width` and `height` are explicitly set to 0
- An ancestor element is hidden, so the element is not shown on the page

If the element satisfies any of these conditions, it will not be matched by the `:visible` selector.

Hidden (:hidden)

Select all elements that are hidden.

Examples

- `$('li:hidden')` selects all `<li>` elements that are hidden
- `$('.myclass:hidden')` selects all elements with the class `myclass` that are hidden

Description

The `:hidden` selector matches items that are currently hidden on the page.
For details on how this determination is made, see the *Description* for *:visible*.

Header element (:header)

Select all elements that are headers, such as `<h1>` or `<h2>`.

Examples

- `$(':header')` selects all header elements
- `$('.myclass:header')` selects all header elements with the class `myclass`

Currently animating (:animated)

Select all elements that are in the progress of an animation at the time the selector is run.

Examples

- `$(':animated')` selects all elements that are currently animating
- `$('.myclass:animated')` selects all elements with the class `myclass` that are currently animating