FULL SAIL
UNIVERSITY

programming for web applications

12

jQuery selectors and manipulators

content.Manipulation

content.**Manipulation**

Get/Set Contents

.html()	Gets all the html contents of the set. (getter)
.html(*text*)	Replaces the html content of the set with new code. (setter)

- Note that **these methods interpret html** (which includes tags).
- The **get** method will return a string of all the html content ***(cannot be chained)***. It is the equivalent of the **.innerHTML** property, from Javascript.
- The **set** method returns the original matched set (including any changes), and **can be chained**. *Just like innerHTML, this replaces all of the html inside the element.*

content.**Manipulation**

Get/Set Contents

.text()	Gets all the text contents of the set *(concatenates all of the text inside any elements and* ***returns a single string*** *as a result).* ***(getter)***
.text(*text*)	Replaces the content of the set with text. **(setter)**

- One of the benefits of the text set-method is that it will automatically encode HTML entities correctly. For example:

```
$("selector").text("Miyagi > Karate Kid")
```

becomes...

```
"Miyagi &gt; Karate Kid"
```

content.**Manipulation**

Creating New Elements

- These next methods we'll discuss **create new elements in the DOM.** The key difference between all of them is **where they add the new content.**
- These are extremely **useful for adding content to a target set**, using parent-child relationships to designate the point of entry.

content.**Manipulation**

target.append(*content*)	Appends the contents (can be html, text, or element sets) as the last child of the *target* set.

- If the *content* is a string of html or text, it is **created** and **then appended** inside.
- With **append**, the original *target* is the returned object that can be chained on.
  - *This code would animate the paragraphs, not just the anchors.*

```
$('#desc').append('<a href="#">link</a>').animate();
```

content.**Manipulation**

content.appendTo(*selector*)	Moves all the elements specified by *content* and appends them to the elements of the *selector*

```
$('<a href="">Link</a>').appendTo('#nav').animate();
```

- **appendTo** is similar to the reverse of **append**, with the *target* in the factory.
- The benefit of this is in chaining. Previous, the target of further chains was the element being appended *on*. Now, the targets are the elements *being* appended.
- In the above example, the link is being made into #nav, and then animated.

content.**Manipulation**

Creating Elements

- Keep in mind, with **appendTo,** the new html is the factory target.. so all chained methods will occur on the newly created markup. Use **append** if you want the jquery target to be the insertion point instead.

```
$('<a href="">Go Home</a>')
    .appendTo("#nav")
    .slideDown()
;
```

```
$('#nav')
    .append("<a href="">Go Home</a>")
    .slideDown()
;
```

content.**Manipulation**

Creating Elements

- Similar to the append method are **prepend**, which **adds the content to the beginning of the target elements**.

target.prepend(*content*)	Moves all elements of *content* into the beginning of the *target* set. Returns the *target* for chaining.
content.prependTo(*target*)	Moves all elements of *content* into the beginning of the *target* set. Returns the *content* for chaining.

content.**Manipulation**

Creating Elements

- Similar again, these **before** and **after** methods **insert the content as siblings of the target, instead of as a child of the target.**

target.before(*content*)	Moves all elements of *content* into positions of siblings of *target*, in front of *target*. Returns ***target*** for chaining.
content.insertBefore(*target*)	Moves all elements of *content* into positions of siblings of *target*, in front of *target*. Returns ***content*** for chaining.

target.after(*content*)	Moves all elements of *content* into positions of siblings of *target*, after the *target*. Returns ***target*** for chaining.
content.insertAfter(*target*)	Moves all elements of *content* into positions of siblings of *target*, after the *target*. Returns ***content*** for chaining.

content.**Manipulation**

Wrapping Elements

- The methods we've looked at so far *(before, after, append, etc..)*, these all move or insert content ***into*** something.
- These next few methods will instead take a target of content, and ***wrap*** **it with some html markup**.
- For example, if I wanted to target all links on a page and wrap them each individually in a li with a class:

```
$("a").wrap('<li class="anchor"></li>');
```

content.**Manipulation**

Replacing Elements

- These next 2 methods instead take a *target* and replace all of its contents with the new specified *content.*
- As usual, the key difference between these is *what* is being returned...

target.replaceWith(*content*)	Replaces all of the contents of *target* with *content*. This method returns the *target* for chaining, even though it has been removed from the DOM.
content.replaceAll(*target*)	Replaces all of the contents of *target* with *content*. Returns *content* as the target of further chaining.

```
$('<p>New Paragraph</p>').replaceAll('span').animate();
```

```
$('span').replaceWith('<p></p>').appendTo('#blah').animate();
```

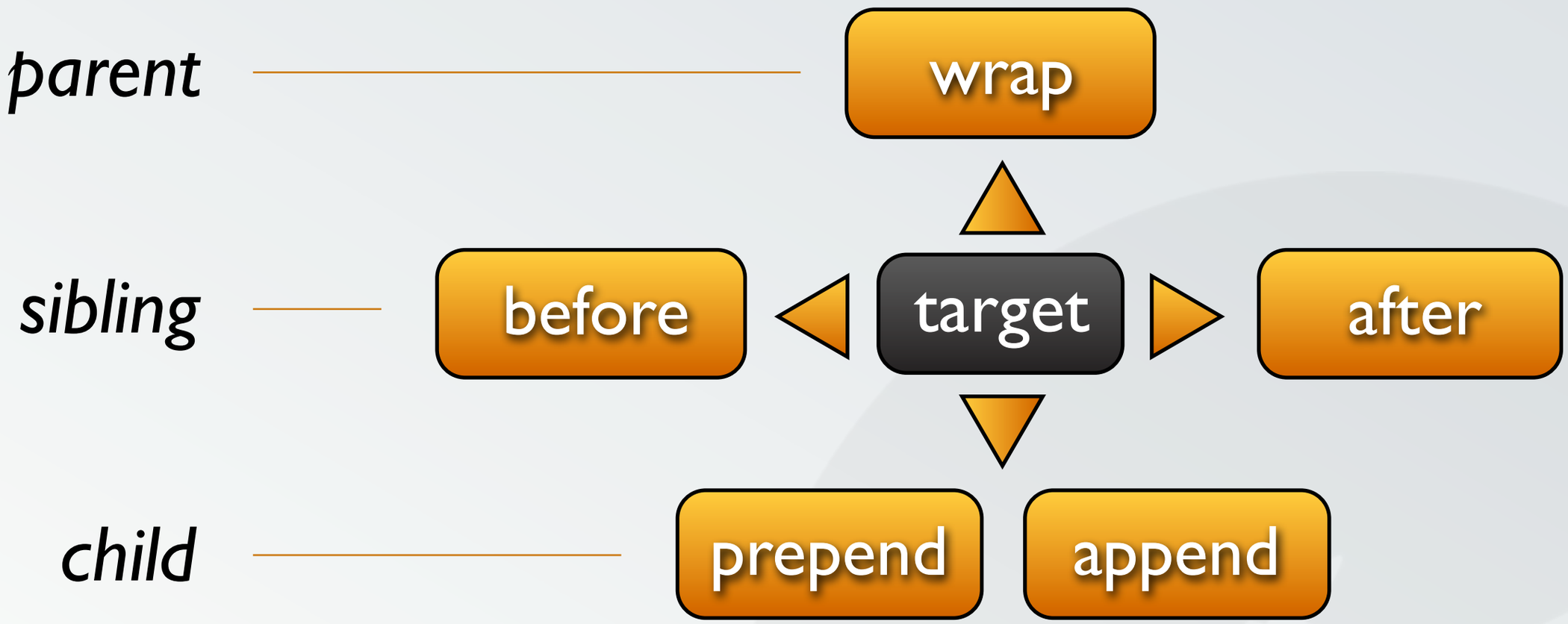
content.**Manipulation**

Wrapping Elements

target.wrap(*wrapper*)	Takes each element in the *target* set and wraps each individually with the html of *wrapper*.
target.wrapAll(*wrapper*)	Takes the entire *target* set as a whole and wraps the entire blocks with the html of *wrapper*.
target.wrapInner(*wrapper*)	Takes each element in the *target* set and wraps the ***contents*** of each individually with the html of *wrapper*.

- Each of these returns the *target*. The *wrapper* can be any set of complex html elements, but must be validly closed:

```
$("p").wrapInner("<span><strong><em></em></strong></span>");
```


parent
wrap
sibling
before
target
after
child
prepend
append

content.**Manipulation**

Removing Elements

target.empty()	Removes all elements inside of the matched set.
target.remove()	Removes all elements of the matched set.
target.detach()	Same as *remove*, except it saves jQuery data on the element instead of discarding.

- While **empty** leaves the original set selectable *(can be chained)*, **remove** destroys the set and its descendants.

```
$("p").empty();
$("p").remove();
```

content.**Manipulation**

Copying Elements

- The clone method is a unique manipulation method with only 1 argument:

target.clone(*boolean*)	Creates a clone of the *target* element(s), including all children and text. If *boolean* is true, it also copies all events throughout. *(default: false)*.

- With this method, the returned element is the *clone*, not the original. In addition, the clone is not yet inserted into the DOM, until one of our other methods is used *(such as appendTo)*

```
$("p").clone().appendTo("#mydiv").animate();
```

```
$("a#mine").clone(true).appendTo("#mydiv").animate();
```