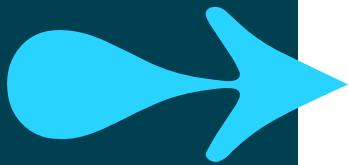


INTRODUCTION



What is the DOM?

- The DOM and HTML tree

Selecting elements

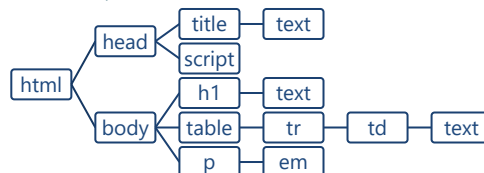
- Basic Selectors
- CSS Selector patterns

Arrays of selected objects

Creating new elements

QA What is the Document Object Model?

- HTML documents have a hierarchical structure that form the DOM
 - Every element, except <html> is contained within another
 - Creating a parent/child relationship



- A DOM tree contains two types of elements
 - Nodes.
 - Text.

HTML documents have what is called a hierarchical structure. Each element (or tag) except the top <html> tag is contained in another element, its parent. This element can in turn contain child elements. You can visualise this as a kind of organisational chart or family tree.

HTML has valid rules for how this DOM fits together and renders the markup delivered from the server into a client side DOM. For all intents and purposes, the DOM is a complex array.

JavaScript provides a series of inbuilt functions to manipulate the structure of the HTML in your browser. It is fair to say that nothing rendered in the page is safe from your grubby little mitts as you start to traverse the page with JavaScript. You can manipulate existing items and create new ones.

QA HTML markup to DOM object (1)

- Consider the following HTML

```

```

- The tag has a type of and four attributes
 - `id`
 - `src`
 - `alt`
 - `title`
- The element is read and interpreted by the browser into a DOM
 - Each element becomes a NodeList object
 - Assigned a property based on the html attribute

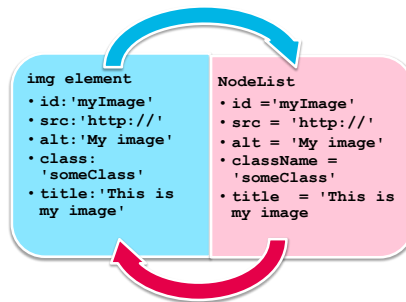
4

HTML gives the browser instructions on how to render the order and set the properties of the objects. Each element in HTML can have zero or more attributes assigned to it. So, in the example above each element has properties and attributes assigned to it.

These properties and attributes are initially assigned to the DOM elements as a result of parsing their HTML markup and can be changed dynamically under script control. To make sure we have our terminology and concepts straight, consider the following HTML markup for an image element:

QA HTML markup to DOM object (2)

- HTML is translated into DOM elements, including the attributes of the tag and the properties created from them.



5

For the most part, the name of a JavaScript attribute property matches that of any corresponding attribute, but there are some cases where they differ. For example, the class attribute in this example is represented by the `className` attribute property.

Every element mapped into the DOM is, in fact, an object literal of key value pairs as we discussed in the previous chapter. If we can select the element we want from the DOM, we can manipulate it in exactly the same way.

Selecting elements

- HTML DOM elements can be selected via JavaScript
 - Single elements can be selected in the following ways:

```
let x = document.getElementById('id');
```

```
let y = document.querySelector('#id');
```

- Multiple elements can be selected using the following approaches:


```
let allP = document.getElementsByTagName('p');
```

```
let allA = document.querySelectorAll('div > a');
```

6

To manipulate a DOM object, you need to gain access to it. To achieve this, we create a variable that points to the required object. Up until recently, this was achieved with a set of elements of the document object starting with getElement...

ECMAScript5 introduces a new way of doing things using the **querySelector** approach. These methods leverage the CSS3 selector engine. The power and flexibility of these selectors are astounding. However, legacy browsers (IE7<) do not support these methods and they will not work. We will look at strategies to deal with legacy browsers later in the course.



Basic Selectors

- CSS Selectors allow us to obtain almost any DOM element

| Selector | Definition |
|-----------------------------------|--|
| 'a' | This selector matches all link (<a>) elements. |
| #specialID | This selector matches elements that have an id of specialID. |
| '.specialClass' | This selector matches elements that have the class of specialClass. |
| 'a#specialID.specialClass' | This selector matches links with an id of specialID and a class of specialClass. |
| 'p a.specialClass' | This selector matches links with a class of specialClass declared within <p> elements. |

7

For applying styles to page elements, web developers have become familiar with a small, but powerful and useful, group of selection methods that work across all browsers. Those methods include selection by an element's ID, CSS class name, tag name, and the DOM hierarchy of the page elements.

Here are some examples to give you a quick refresher:

- `a` – This selector matches all link (<a>) elements
- `#specialID` – This selector matches elements that have an id of specialID
- `.specialClass` – This selector matches elements that have the class of specialClass
- `a#specialID.specialClass` – This selector matches links with an id of specialID and a class of specialClass
- `p a.specialClass` – This selector matches links with a class of specialClass declared within <p> elements

We can mix and match the basic selector types to select fairly fine-grained sets of elements. In fact, the most fancy and creative websites use a combination of these basic options to create their dazzling displays.



Child, container and attribute selectors (1)

- These selectors are part of the CSS specification
- Only exceptionally old browsers won't support them (pre-IE8)

| Selector | Description |
|----------|--|
| * | Matches any element |
| E | Matches all element 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 F immediately preceded by sibling E |
| E~F | Matches all elements F preceded by any sibling E |

8

For more advanced selectors, the **querySelector** and **querySelectorAll** methods use the next generation of CSS supported by Mozilla Firefox, Internet Explorer 8, Safari and other modern browsers. These advanced selectors include selecting the direct children of some elements, elements that occur after other elements in the DOM, and elements with attributes matching certain conditions.

Suppose we want to select an external hyperlink. Using basic CSS selectors, we might try something :

```
ul.myList li a.
```

Unfortunately, that selector would grab all links because they all descend from a list element.

A more advanced approach is to use child selectors, where a parent and its direct child are separated by the right angle bracket character (>). This selector matches only links that are direct children of an element. If a tag was further embedded, say within a within a <p>, that tag would not be selected. Going back to our example, consider a selector such as:

```
ul.myList > li > a
```

This selector selects only links that are direct children of list elements, which are in turn direct children of elements that have the class myList.

Attribute selectors example

- Use attribute selectors with care as they can be expensive
 - A complex search pattern
- ^= operator finds attributes starting with a value

```
document.querySelectorAll('a[href^="http"]');
```

- \$= operator finds attributes ending with a value

```
document.querySelectorAll('a[href$=".doc"]');
```

- *= operator finds attributes containing the value

```
document.querySelectorAll('a[href*="name"]');
```

9

Attribute selectors are also extremely powerful. If we return to our example, where we wish to provide a different behaviour to links that point to external pages, we can easily achieve this.

Any external hyperlink will start with the prefix `http://` if we use the selector:

```
a[href^=http://]
```

The ^ symbol specifies that the match must occur at the beginning of the value.

QA Selecting by position

- Elements can be selected by position in relation to other elements

| Selector | Description |
|-----------------------------|--|
| :first-of-type | The first match of an element on a page. li a:first-of-type returns the first link also under a list item. |
| :last-of-type | The last match of the page. li a:last-of-type returns the last link also under a list item. |
| :first-child | The first child element. li:first-child returns the first item of each list. |
| :last-child | The last child element. li:last-child returns the last item of each list. |
| :only-child | Returns all elements that have no siblings. |
| :nth-child(n) | The nth child element. li:nth-child(2) returns the second list item of each list. |
| :nth-child(even odd) | Even or odd children. li:nth-child(even) returns the even children of each list. |

10

Sometimes, we'll need to select elements by their position on the page or in relation to other elements. We might want to select the first link on the page, or every other paragraph, or the last list item of each list. jQuery supports mechanisms for achieving these specific selections, for example:

a:first-of-type

This format of selector matches the first <a> element on the page.

What about picking every other element?

p:nth-child(odd)

This selector matches every odd paragraph element. As we might expect, we can also specify that evenly ordered elements be selected with.

p:nth-child(even)

As above but even p elements.

li:last-child

chooses the last child of parent elements. In this example, the last child of each element is matched.

Creating new content – DOM programming

- The DOM can have new objects added to it using JavaScript

```
let e1 = document.createElement('p');
```

- This creates the `<p>` part of the html but not its text
 - The text node is part of DOM as well as the markup

```
let text = document.createTextNode('stuff');
```

- Then you must append the text node to the element
 - Then the element to the DOM tree

```
e1.appendChild(text);  
document.querySelector('#id').appendChild(e1);
```

QA Creating new content – innerHTML and textContent

- In the olden days, IE broke the DOM programming standard
 - All browsers now support **innerHTML** and **innerText** (prefer **textContent** over **innerText**)
- These functions allow us to add to the DOM in a quick and dirty way
 - The entire JavaScript string is parsed into a HTML element
 - Beware that older browsers can face injection attacks!

```
let el = document.querySelector('#id');  
el.innerHTML = "<em>cool</em>";
```

12

From <https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent>

Internet Explorer introduced [node.innerText](#). The intention is similar but with the following differences:

- While **textContent** gets the content of all elements, including [<script>](#) and [<style>](#) elements, **innerText** does not
- **innerText** is aware of style and will not return the text of hidden elements, whereas **textContent** will
- As **innerText** is aware of CSS styling, it will trigger a reflow, whereas **textContent** will not
- Unlike **textContent**, altering **innerText** in Internet Explorer (up to version 11 inclusive) not only removes child nodes from the element, but also *permanently destroys* all descendant text nodes (so it is impossible to insert the nodes again into any other element or into the same element anymore)



QuickLab 10

- Creating new content using the DOM

REVIEW



What is the DOM?

- The DOM and HTML tree

Selecting elements

- Basic Selectors
- CSS Selector patterns

Arrays of selected objects

Creating new elements