Overview

- The **Document Object Model** (DOM) is a programming interface for HTML or XML documents.
- Models document as a tree of nodes.
- Nodes can contain text and other nodes.
- Nodes can have attributes which include style and behavior attributes.
- Possible to get all nodes of a particular type, specific `class` or `id`.

Document Object Model

- API to access parsed HTML/XML documents.
- Can be used from any language, but within browsers the only language commonly supported currently is JavaScript.
- Datatypes include `document`, `element`, `attribute`.

Document

- Current document available as `document` property of global `window` object.
- Properties include `location` (URL, giving `href`, `protocol`, `hostname`, `port`, `pathname`, `search`, `hash`), `contentType`, `body`, `cookie` (cookie defs separated by `;`).
- Methods include `getElementsByTagName()`, `getElementsByName()`, `getElementById()`, `getElementsByClassName()`, `querySelector()`, `querySelectorAll()`.
- Allows updating document content dynamically *Dynamic HTML* (DHTML).

Element

- Represents an individual HTML element.
- Properties include `id`, `classList`, `innerHTML` (markup within element), `attributes` (map `NamedNodeMap` of attributes).
- Methods include `getAttribute()`, `getAttributeNames()`, `removeAttribute()`, `setAttribute()`.

Stylesheets

- Current best practice is to relegate presentation to stylesheets.
- Can be specified using **external** stylesheets, using `<link>` elements.
- Can also be specified using **internal** stylesheets using `<style>` elements.
- Can also be specified **inline** for an individual element using `style` attribute.
- Precedence (in descending order) inline, internal, external.

Cascading Style Sheets

- *Cascading Style Sheets* (CSS) specifies priority rules (cascade) between different style declarations which may apply to a element.
- A CSS stylesheet consists of a set of rules.
- A rule consists of a selector followed by a brace delimited set of CSS declarations separated by ;.

```
p .highlight {
 background-color: yellow;
 color: blue
}
```

- Will not cover CSS declarations.

Simple CSS Selectors

Universal Selector * selects all elements; usually used in conjunction with other selectors.

HTML Element Names Simply specify name of HTML element. Examples p, a, table.

Class Selectors Name of class preceeded by a .. Examples .highlight, .important.

ID Selectors ID of element preceeded by #. Examples include #form1, #table1. Note that ID must be unique in document.

[attr] Selects all elements having attribute attr. Examples [href],

Combining Selectors

Constrain	Can follow selector by class or id selectors (without spaces). `p.chemical` matches p elements having class `chemical`.
Descendent	Simply write selectors adjacent to each other separated by a space. Example: `.chemical p` selects all p elements which are descendents of a element which has class `chemical`.
Child	Write selectors separated by a >. Example: `.chemical > p` selects all p elements which are direct children of a element which has class `chemical`.

Combining Selectors Continued

Sibling	Write selectors separated by a `~`. Example: `.chemical ~ p` selects all p elements which follow (not necessarily immediately) a element which has class `chemical`.
Adjacent Sibling	Write selectors separated by a `+`. Example: `.chemical + p` selects all p elements which immediately follow a element which has class `chemical`.

Unobstrusive JavaScript

Different technologies used for different concerns:

Content	HTML used for content.
Presentation	CSS used for styling.
Behavior	JavaScript used to specify behavior.

- Do not mix technologies.
- Best practice is to split out into separate `*.html`, `*.css` and `*.js` files.
- Modern technology blurs lines between concerns; CSS 3 contains support for visual behavior traditionally achieved using JavaScript. Nevertheless it remains a good organizational principle.

Bad Code

In doc.html:

```
<a href="submit.cgi"
   onClick="checkForm(this)"
   style="font-weight: bold">
  Submit
</a>
```

- Uses CSS and JavaScript code within attributes of HTML elements.
- Maintaining file will require content, presentational and programming skills.

Better Code

1. In `doc.html` maintained by content specialist or a *Content Management System* (CMS):‘<a href="submit.cgi" id="submit">Submit</a>‘.
2. In `doc.css` maintained by web designer `#submit { font-weight: bold; }`.
3. In `doc.js` maintained by front-end programmer: `document.¬ getElementById('submit').onclick(checkForm(this))`.

- Separate concerns, separate files, separate specialists.
- `doc.html` will need to reference `doc.css` stylesheet and `doc.js`.
- In practice, single `.css` stylesheet, `.js` file shared by multiple html documents.

Playing with the DOM

dom-play.html

JQuery

- Used by a large percentage ($> 70\%$ as of *Nov, 2018*) of public facing web sites.
- Raison d'etre was hiding browser incompatibilities re. DOM and event model.
- Less necessary now that browsers and JavaScript implementations are more standards compliant.
- Newer applications using newer frameworks will typically not use jQuery.
- Nevertheless, it remains a very popular legacy framework.

Replaces DOM Tedium

From jqia:

```
let checkedValue;
let elements =
  document.getElementsByTagName('input');
for (let i = 0; i < elements.length; i++) {
  if (elements[i].type === 'radio' &&
      elements[i].name === 'some-radio-group' &&
      elements[i].checked) {
  checkedValue = elements[i].value;
  break;
 }
}

let checkedValue =
 jQuery('input:radio[name="some-radio-
group"]:checked').
```

jQuery Object

- `jQuery` is a function which takes up to 2 arguments with different behavior based on form and type of arguments.
- `jQuery` often aliased as `$`.

On Load Handler

Need to wait for page to load before operating on DOM.
`window.onload()` handler provided by DOM:

```
window.onload = function() { //waits for page display,
  ...                         //and loading external assets
}                             //only single onload() handler
```

replaced by jQuery's `$(document).ready()` handler:

```
$(function() {   //waits for page DOM to be built;
  ...             //external assets may still be loading
});               //can have multiple ready handlers.
```

jQuery Chaining

Most `jQuery` functions return the object they operated on which allows method chaining. Hence

```
obj.fn1().fn2();
```

instead of:

```
obj.fn1();
obj.fn2();
```

jQuery CSS Style Selectors

- `$('p.chemical')` returns all p elements having class `chemical`.
- `$('#h2o.chemical')` returns all elements having id `h2o` and class `chemical`.
- `$('p .chemical')` returns all elements with class `chemical` which are descendents of p elements.
- `$('p > .chemical')` returns all elements with class `chemical` which are children of p elements.
- `$('p + .chemical')` returns all elements with class `chemical` which are immediately preceeded by a p sibling element.
- `$('p ~ .chemical')` returns all elements with class `chemical` which are preceeded by a p sibling element.

Attribute Selectors

- `$('p[data-val]')` matches elements of type `p` which have a `data-val` attribute with any value.
- `$('.chemical[data-atomic-wt="92"]')` matches elements having class `chemical` having a `data-atomic-wt` attribute having value `"92"`.
- `$('.chemical[data-name^="A"]')` matches elements having class `chemical` having a `data-name` attribute having value starting with `A`.
- `$('.chemical[data-name$="ium"]')` matches elements having class `chemical` having a `data-name` attribute having value ending with `ium`.
- `$('.chemical[data-atomic-wt!="92"]')` matches elements having class `chemical` having a `data-atomic-wt` attribute having value not equal to `"92"` or lacks `data-atomic-wt` attribute completely.

Position and Child Filters

Position filters: `:first`, `:last`, `:even`, `:odd`, `:eq(n)`, `:gt(n)`, `:lt(n)`. Indexes start at 0.

- `$('p:first')` selects first paragraph.
- `$('li:odd')` selects all li elements at odd indexes.
- `$('li:gt(4):odd')` selects all li elements at odd indexes greater than 4.
- `$('ul:first-child')` selects first child of `ul`.
- `$('ul:last-child')` selects last child of `ul`.
- `$('ul:nth-child(3)')` selects third child of `ul` (index starts at 1).
- `$('ul:nth-child(3n+1)')` selects first, fourth, ... child of `ul`.

Form Filters

`$('input[name="widget"]:checked')` selects `input` controls having name `widget` with state checked.

`$('input:disabled')` selects `input` controls which are disabled.

`$('input:focus')` selects `input` controls which are focused.

`$(':selected')` selects `option` elements which are selected.

Accessing and Updating Elements

- `addClass(names)` / `removeClass(names)` / `toggleClass(names)` will add/remove/toggle classes given by spaced-delimited `names`.
- `css(name)` / `css(name, value)` will return value css property `name` of first matched / update css property `name` of all matched to `value`.
- `html()` / `html(value)` will access matched elements inner HTML / update inner HTML of matched elements to `value`.
- `val()` / `val(value)` will return value of first matched form control / update all matched form control values to `value`.
- `attr(name)` / `attr(name, value)` will return value of attribute `name` of first matched element / update attribute `name` of all matched elements to `value`. If `value` is a function return value is new value; it is called with index and current value with `this` set to element.

Accessing and Updating Elements Continued

- `append(content)` / `prepend(content)` will append / prepend argument `content` to content of all matched elements. If `content` is a function return value is added content; it is called with index and existing element context with `this` set to element.
- `before(content)` / `after(content)` will insert `content` as sibling before / after all matched elements. The `content` argument has the same semantics as for `append()`.
- `remove()` will remove all matched elements from DOM. Can be called with optional parameter `selector` as `remove(`selector‘) which allows filtering on elements to be removed.
- `detach()` is like `remove()` except that it retains events and data associated with elements so that the elements can be inserted back. Preferred way to make changes in background.

Events

- When browser events (like key presses, mouse clicks, page loads) occur, browser calls a **event handler**.
- Historically, different browsers had different ideas of how a event was propagated between an element and its containing elements.
- DOM level 0 allows you to assign a **single** handler to each event for an element using syntax like `element.onclick = function(event) { ... }`. Problematic in that different scripts may each try to add handlers for the same event.
- In DOM level 0 event bubbles up from leaf element on which event occurs to its parent all the way up the DOM tree.
- DOM level 2 event model has a *capture phase* (before *bubble phase*) where event propagated down from the top level of the DOM tree to the leaf element causing the event.
- DOM level 2 allows adding **multiple** handers for an event using `addEventListener(eventType, handler, useCapture)`.

jQuery Event Handlers

`on(eventType, handler)`

eventType	Includes `blur`, `change`, `click`, `focus`, `keydown`, `keyup`, `keypress`, `mouse{down,enter,leave,move,out,over,up}`, `select`, `submit`.
handler	Function which handles event. First argument is the event which is a cleaned up version of the DOM event which provides event properties like the (x,y) position, target, etc. Has methods like `preventDefault()`, `stopPropagation()`. `this` accesses the DOM element to which the handler was bound.

Other Event Methods

`trigger(eventType)` Trigger event `eventType` on matched elements.

`off(eventType)` Remove all event handlers for `eventType` from matched elements.

Example

jquery-play

Example which illustrates use of input fields with a jQuery based script.

Example illustrates following concepts:

- Data-driven programming.
- The use of Immediately Invoked Function Expressions (IIFEs) for encapsulation.
- The use of jQuery.

References

jQuery Web Site

[jquia] Bear Bibeault, Yehuda Katz, and Aurelio De Rosa, *jQuery in Action*, Third Edition, Manning, 2015.