JavaScript

Using JavaScript in the Browser

April 3, 2018

Peter J. Jones

✉ pjones@devalot.com

@devalot

http://devalot.com

What's In Store

Before Lunch	After Lunch
HTML and CSS Refresher	Event Handling
The Document Object Model	Making Ajax Requests
Manipulating Page Elements	jQuery Ajax Functions
Using jQuery DOM Functions	CORS vs. JSONP
Debugging	Validating Forms

JavaScript and the Browser

How JavaScript fits in:

- HTML for content and user interface
- CSS for presentation (styling)
- JavaScript for behavior (and business logic)

What is HTML?

- Hyper Text Markup Language
- HTML is very error tolerant (browsers are very forgiving)
- That said, you should strive to write good HTML
- Structure of the UI and the content of the **view data**
- Parsed as a tree of nodes (elements)
- HTML5
  - Rich feature set
  - Semantic (focus on content and not style)
  - Cross-device compatibility
  - Easier!

Anatomy of an HTML Element

- Also known as: nodes, elements, and tags:

```
<element key="value" key2="value2">
  Text content of element
</element>
```

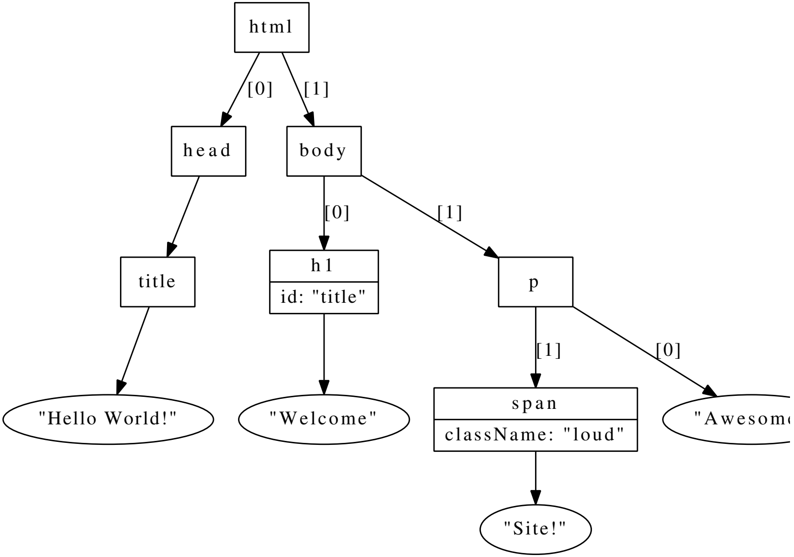
HTML Represented as Plain Text

```html
<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>
    <h1 id="title">Welcome</h1>

    <p>
      Awesome <span class="loud">Site!</span>
    </p>
  </body>
</html>
```

HTML Parsed into a Tree Structure



What is CSS?

- Cascading Style Sheets
- Rule-based language for describing the look and formatting
- Separates presentation from content
- Can be a separate file or inline in the HTML
- Prefer using a separate file

What Does CSS Look Like?

```css
p {
  background-color: white;
  color: blue;
  padding: 5px;
}

.spoiler {
  display: none;
}

p.spoiler {
  display: block;
  font-weight: bold;
}
```

Anatomy of a CSS Declaration

- Selectors choose which elements you want to style. A selector is followed by a body where styling properties are set:

```
selector {
  property-x: value;
  property-y: val1 val2;
}
```

- For example:

```
h1 {
  color: #444;
  border: 1px solid #000;
}
```

The Various Kinds of Selectors

- Using the element's type (name):
  - HTML: `<h1>Hello</h1>`
  - CSS: `h1 {...}`
- Using the ID attribute:
  - HTML: `<div id="header"></div>`
  - CSS: `#header {...}`
- Using the class attribute:
  - HTML: `<div class="main"></div>`
  - CSS: `.main {...}`
- Using location or relationships:
  - HTML: `<ul><li><p>One</p></li><li>Two</li></ul>`
  - CSS: `ul li p {...}`

How the Browser Processes JavaScript

- Parser continues to process HTML while downloading JS
- Once downloaded, JS is executed and *blocks* the browser
- Include the JS at the bottom of the page to prevent blocking

Getting JavaScript into a Web Page

- Preferred option:

  ```
  <script src="somefilename.js"></script>
  ```

- Inline in the HTML (yuck):

  ```
  <script>
    var x = "Hey, I'm JavaScript!";
    console.log(x);
  </script>
  ```

- Inline on an element (double yuck):

  ```
  <button onclick="console.log('Hey there');"/>
  ```

How JavaScript Affects Page Load Performance (Take Two)

- The browser blocks when executing JS files
- JS file will be downloaded then executed before browser continues
- Put scripts in file and load them at the bottom of the page

What is the DOM?

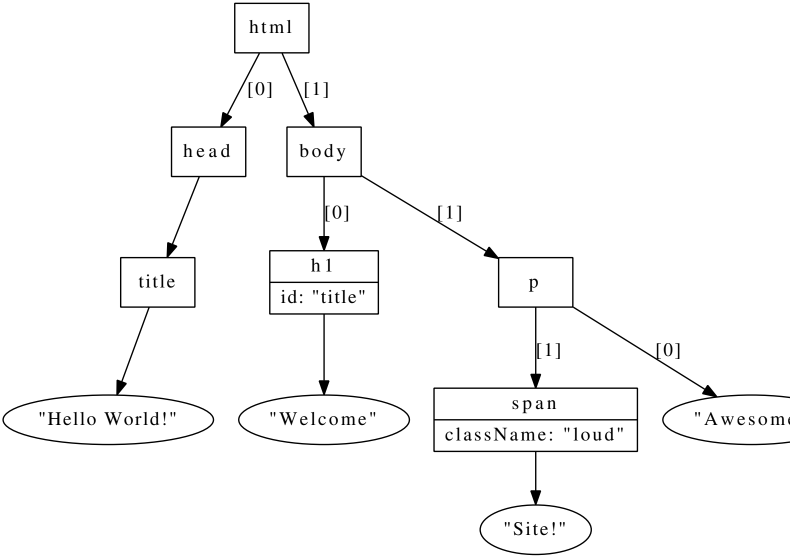
- What most people hate when they say they hate JavaScript
- The DOM is the browser's API for the document
- Through it you can manipulate the document
- Browser parses HTML and builds a tree structure
- It's a live data structure

The Document Structure

- The `document` object provides access to the document
- It's a tree-like structure
- Each node in the tree represents one of:
  - Element
  - Content of an element
- Relationships between nodes allow traversal

Looking at the Parsed HTML Tree (again)

And produce this tree structure:



Element Nodes

- The HTML:

```
<p id="name" class="hi">My <span>text</span></p>
```

- Maps to:

```
var node = {
  tagName:    "P",
  childNodes: NodeList,
  className:  "hi",
  innerHTML:  "My <span>text</span>",
  id:         "name",
  // ...
};
```

  - Attributes may **very loosely** to object properties

Working with the Document Object Model

- Accessing elements:
  - Select a single element
  - Select many elements
  - Traverse elements
- Working with elements
  - Text nodes
  - Raw HTML
  - Element attributes

Performance Considerations

- Dealing with the DOM brings up a lot of performance issues
- Accessing a node has a cost (especially in IE)
- Styling has a bigger cost (it cascades)
  - Inserting nodes
- Layout changes
  - Accessing CSS margins
  - Reflow
  - Repaint
- Accessing a `NodeList` has a cost

Accessing Individual Elements

Starting on the `document` object or a previously selected element:

`document.getElementById("main");` Returns the element with the given ID (e.g., `<div id="main">`).

`document.querySelector("p span");` Returns the *first* element that matches the given CSS selector.
The search is done using depth-first pre-order traversal.

Accessing a List of Elements

Starting on the `document` object or a previously selected element:

`document.getElementsByTagName("a");` Returns a `NodeList` containing *all* `<a>` elements.

`document.getElementsByClassName("highlight");` Returns a `NodeList` containing *all* elements that have a `class` attribute set to `foo` (e.g., `<div class="highlight">`).

`document.querySelectorAll("p span");` Returns a `NodeList` containing *all* elements that match the given CSS selector.

Traversal Functions

`parentNode`	The parent of the specified element.
`previousSibling`	The element immediately preceding the specified element.
`nextSibling`	The element immediately following the specified element.
`firstChild`	The first child element of the specified element.
`lastChild`:	The last child element of the specified element.
`childNodes`	A `NodeList` containing the direct decedents (children) of the specified element.

But. . .

DOM Living Standard (WHATWG)

Supported in IE $>=$ 9:

`children`: All *element* children of a node (i.e. no text nodes).

`firstElementChild`: First *element* child.

`lastElementChild`: Last *element* child.

`childElementCount`: The number of children that are *elements*.

`previousElementSibling`: The previous sibling that is an *element*.

`nextElementSibling`: The next sibling that is an *element*.

The `nodeType` Property

Interesting values for the `element.nodeType` property:

Value	Description
1	Element node
3	Text node
8	Comment node
9	Document node

Creating New Nodes

`document.createElement("a");` Creates and returns a new node without inserting it into the DOM.
In this example, a new <a> element is created.

`document.createTextNode("hello");` Creates and returns a new text node with the given content.

Adding Nodes to the Tree

```
var parent = document.getElementById("customers"),
    existingChild = parent.firstElementChild,
    newChild = document.createElement("li");
```

`parent.appendChild(newChild);` Appends `newChild` to the end of `parent.childNodes`.

`parent.insertBefore(newChild, existingChild);` Inserts `newChild` in `parent.childNodes` just before the existing child node `existingChild`.

`parent.replaceChild(newChild, existingChild);` Removes `existingChild` from `parent.childNodes` and inserts `newChild` in its place.

`parent.removeChild(existingChild);` Removes `existingChild` from `parent.childNodes`.

HTML and Text Content

```
var element = document.getElementById("foo"),
    name    = "bar";
```

`element.innerHTML` Get or set the element's decedents as HTML.

`element.textContent`: Get or set *all* of the text nodes (including decedents) as a
single string.

`element.nodeValue` If `element` is a text node, comment, or attribute node, returns
the content of the node.

`element.value` If `element` is a form input, returns its value.

Exercise: DOM Manipulation

1. Open the following files in your text editor:
   - `src/www/js/flags/flags.js`
   - `src/www/js/flags/index.html` (read only!)
2. Open the `index.html` file in your web browser.
3. Complete the exercise.

What is jQuery?

- A utility library that makes DOM manipulation easier
- Includes an Ajax and Promise library as well
- Used to be an important part of cross-browser development

Using jQuery

- Load the library into your HTML using a `<script>` tag
- Use the `$` function to access the API, or
- Call `jQuery.noConflict();` then access the API through the `jQuery` API

Fetching Elements with jQuery

```
$("#view li").each(function() {
  console.log($(this).text());
});
```

- When the $ function is given a CSS selector as its argument, it returns a jQuery object holding all of the matching elements.
- When given a DOM node, the $ function wraps it into a jQuery object.
- The `text` and `html` functions act like `textContent` and `innerHTML` respectively.

Creating New Elements

```
artists.forEach(function(a) {
  $("<li>").text(a).appendTo("#view ul");
});
```

- When the $ function is given an HTML fragment it turns it into a node and wraps it into a jQuery object.
- `appendTo` is similar to `appendChild` except the receiver is the node to add, and the first argument is the parent.

Introduction to Debugging

- All modern browsers have built-in JavaScript debuggers
- We've been using the debugging console the entire time!

Browser Debugging with the Console

- The `console` object:
  - Typically on `window` (doesn't always exist)
  - Methods
    - `log`, `info`, `warn`, and `error`
    - `table(object)`
    - `group(name)` and `groupEnd()`
    - `assert(boolean, message)`

Accessing the Debugger

- In the browser's debugging window, choose **Sources**
- You should be able to see JavaScript files used for the current site

Setting Breakpoints

There are a few ways to create breakpoints:

- Open the source file in the browser and click a line number
- Right-click the line number to create conditional breakpoints
- Use the `debugger;` statement in your code

Stepping Through Code

- After setting breakpoints, you can reload the page (or trigger a function)
- Once the debugger stops on a breakpoint you can step through the code using the buttons in the debugger
    - Step In: Jump into the current function call and debug it
    - Step Over: Jump over the current function call
    - Step Out: Jump out of the current function

Console Tricks

- `$_` the value of the last evaluation
- `$0`—`$4` last inspected elements in historical order
- `$("selector")` returns first matching node (CSS selector)
- `$$("selector")` returns all matching nodes
- `debug(function)` sets a breakpoint in `function`
- `monitor(function)` trace calls to `function`

Events Overview

- Single-threaded, but asynchronous event model
- Events fire and trigger registered handler functions
- Events can be click, page ready, focus, submit (form), etc.

So Many Events!

- UI: load, unload, error, resize, scroll
- Keyboard: keydown, keyup, keypress
- Mouse: click, dblclick, mousedown, mouseup, mousemove
- Touch: touchstart, touchend, touchcancel, touchleave, touchmove
- Focus: focus, blur
- Form: input, change, submit, reset, select, cut, copy, paste

Using Events (the Basics)

1. Select the element you want to monitor
2. Register to receive the events you are interested in
3. Define a function that will be called when events are fired

Event Registration

Use the addEventListener function to register a function to be called when an event is triggered:

Example: Registering a click handler:

```
var main = document.getElementById("main");

main.addEventListener("click", function(event) {
  console.log("event triggered on: ", event.target);
});
```

Note: Don't use older event handler APIs such as onClick!

Event Handler Call Context

- Functions are called in the context of the DOM element
- I.e., `this === eventElement`
- Use `bind` or the `var self = this;` trick

Event Propagation

- By default, events propagate from the target node upwards until the root node is reached (bubbling).
- Event handlers can stop propagation using the `event.stopPropagation` function.
- Event handlers can also stop the browser from performing the default action for an event by calling the `event.preventDefault` function

Example: Event Handler

```
main.addEventListener("click", function(event) {
  event.stopPropagation();
  event.preventDefault();

  // ...
});
```

Event Delegation

- Parent receives event instead of child (via bubbling)
- Children can change without messing with event registration
- Fewer handlers registered, fewer callbacks
- Relies on some event object properties:
    - `event.target`: The element the event triggered for
    - `event.currentTarget`: Registered element (parent)

Event Handling: A Complete Example

```
node.addEventListener("click", function(event) {
  // `this' === Node the handler was registered on.
  console.log(this);

  // `event.target' === Node that triggered the event.
  console.log(event.target);

  // Add a CSS class:
  event.target.classList.add("was-clicked");

  // You can stop default browser behavior:
  event.preventDefault();
});
```

Exercise: Simple User Interaction

1. Open the following files in your text editor:
   - `src/www/js/events/events.js`
   - `src/www/js/events/index.html` (read only!)
2. Open the `index.html` file in your web browser.
3. Complete the exercise.

Event Loop Warnings

- Avoid blocking functions (e.g., `alert`, `confirm`)
- For long tasks use eteration or web workers
- Eteration: Break work up using `setTimeout(0)`

Event “Debouncing”

- Respond to events in intervals instead of in real-time
- Reuse a timeout object to process events in the future

```
var input   = document.getElementById("search"),
    output  = document.getElementById("output"),
    timeout = null;

var updateSearchResults = function() {
  output.textContent = input.value;
};

input.addEventListener("keydown", function(e) {
  if (timeout) clearTimeout(timeout);
  timeout = setTimeout(updateSearchResults, 100);
});
```

Listening for Events Using jQuery

```
$("#view").click(function(event) {
  console.log(event.target, "was clicked");
});

$("#reload").on("click", function(event) {
  $("#view").html("");
  load();
});
```

- Use can use the `on` function or one of the shortcut functions such as `click`

Ajax Basics

- Asynchronous JavaScript and XML
- API for making HTTP requests
- Handled by the `XMLHttpRequest` object
- Introduced by Microsoft in the late 1990s
- Why use it? Non-blocking server interaction!
- Limited by the same-origin policy

Ajax: Step by Step

1. JavaScript asks for an HTTP connection
2. Browser makes a request in the background
3. Server responds in XML/JSON/HTML
4. Browser parses and processes response
5. Browser invokes JavaScript callback

Sending a Request, Basic Overview

```
var req = new XMLHttpRequest();

// Attach event listener...

req.open("GET", "/example/foo.json");
req.send(null);
```

Knowing When the Request Is Complete

```
var req = new XMLHttpRequest();

req.addEventListener("load", function(e) {
  if (req.status == 200) {
    console.log(req.responseText);
  }
});
```

Popular Data Formats for Ajax

- HTML: Easiest to deal with
- XML: Pure data, but verbose
- JSON: Pure data, very popular

Ajax with HTML

- Easiest way to go
- Just directly insert the response into the DOM
- Scripts will **not** run

Ajax with XML

More work to extract data from XML:

```
request.addEventListener("load", function() {
  if (request.status >= 200 && request.status < 300) {
    var data = request.responseXML;
    var messages = data.getElementsByTagName("message");

    for (var i=0; i<messages.length; ++i) {
      console.log(messages[i].innerHTML);
    }
  }
});
```

What is JavaScript Object Notation (JSON)?

- Built-in methods:
  - `JSON.stringify(object);`
  - `JSON.parse(string);`
- Example:

```
{
  "messages": [
    {"text": "Hello", "priority": 1},
    {"text": "Bye",   "priority": 2}
  ],
  "sender": "Lazy automated system"
}
```

Ajax with JSON

- Sent and received as a string
- Needs to be serialized and de-serialized:

```
req.send(JSON.stringify(object));

// ...

var data = JSON.parse(req.responseText);
```

Should You Use the XHR API?

- It is best to use an abstraction for `XMLHttpRequest`
- They usually come with better:
  - `status` and `statusCode` handling
  - Error handling
  - Callback registration
  - Variations in browser implementations
  - Additional event handling (progress, load, error, etc.)
- So, use a library like jQuery

Exercise: Making Ajax Requests

1. Open the following files:
    - `src/www/js/artists/artists.js`
    - `src/www/js/artists/index.html` (read only!)
2. Open `http://localhost:3000/js/artists/`
3. Complete the exercise.

Same-origin Policy and Cross-origin Requests

- By default, Ajax requests must be made on the same domain
- Getting around the same-origin policy
  - A proxy on the server
  - JSONP: JSON with Padding
  - Cross-origin Resource Sharing (CORS) (>= IE10)

Recommendation: Use CORS.

Introducing JSONP

- Browser doesn't enforce the same-origin policy for resources (images, CSS files, and JavaScript files)
- You can emulate an Ajax call to another domain that returns JSON by doing the following:
  1. Write a function that will receive the JSON as an argument
  2. Create a `<script>` element and set the `src` attribute to a remote domain, include the name of the function above in the query string.
  3. The remote server will return JavaScript (not JSON)
  4. The JavaScript will simply be a function call to the function you defined in step 1, with the requested JSON data as its only argument.

Example: JSONP

1. Define your function:

```
function myCallback (someObject) { /* ... */ }
```

2. Create the script tag:

```
<script src="http://server/api?jsonp=myCallback">
</script>
```

3. The browser fetches the URL, which contains:

```
myCallback({answer: "Windmill"});
```

4. Your function is called with the requested data

Making Ajax Requests Using jQuery

```
$.getJSON("/api/artists")
  .then(function(artists) {
    var template = $("#template").html();
    var view = Mustache.render(template, {artists: artists});
    $("#view").html(view);
  })
  .fail(function(error) {
    console.error("bloody hell: ", error);
  });
```

- The `then` function was added in jQuery 1.8
- The `catch` function wasn't added until jQuery 3.0

Using the `fetch` Function

```
fetch("/api/artists", {credentials: "same-origin"})
  .then(function(response) {
    return response.json();
  })
  .then(function(data) {
    updateUI(data);
  })
  .catch(function(error) {
    console.log("Ug, fetch failed", error);
  });
```

Browser Support and Documentation

Browsers:

- IE (no support)
- Edge $>=$ 14
- Firefox $>=$ 34
- Safari $>=$ 10.1
- Chrome $>=$ 42
- Opera $>=$ 29

Docs:

- Living Standard
- MDN

Forms

```html
<form action="https://www.google.com" method="get">
  <label>
    Search: <input type="search"
                   name="q"
                   placeholder="Type Here"
                   required>
  </label>

  <input type="submit" value="Search">
</form>
```

See: `src/examples/html/form.html`

Form Input Types

- `button`
- `checkbox`
- `color`
- `date`
- `datetime-local`
- `email`
- `file`
- `hidden`
- `image`
- `month`
- `number`
- `password`
- `radio`
- `range`
- `reset`
- `submit`
- `tel`
- `text`
- `time`
- `url`
- `week`

See the HTML spec for details on the input types.

Exercise: Writing Forms

1. Open the following file:

   `src/www/html/form.html`

2. Let's write some HTML!

Form Validation in the Browser

Validation attributes:

- `max`: Maximum number or date
- `maxlength`: Maximum number of characters
- `min`: Minimum number or date
- `minlength`: Minimum number of characters
- `pattern`: Regular expression `value` must match
- `required`: Input must have a value
- `title`: Describe the `pattern` conditions

CSS Pseudo Classes:

- `:valid`: Element's value is valid
- `:invalid`: Element's value is invalid
- `:optional`: No value is required
- `:required`: A value is required

Exercise: Form Styling

1. Open (and edit) the following files in your text editor:
   - `src/www/css/form/form.css`
   - `src/www/css/form/index.html`
2. Follow the directions in the CSS file
3. Open the HTML file in your browser and confirm your changes

JavaScript Form Validation Workflow

When validating a form with JavaScript:

1. Listen for `input` events on the form inputs and validate the specific input that changed
2. Listen for `submit` events on the form itself and validate the entire form, halting the submission if there are errors

Exercise: JavaScript Form Validation

1. Open (and edit) the following files in your text editor:
    - `src/www/css/form/form.js`
    - `src/www/css/form/index.html`
2. Follow the directions in the JavaScript file
3. Open the HTML file in your browser and confirm your changes