jQuery & AJAX part 1

IMY 220 ● Lecture 13

AJAX

Each web browser tends to supply slightly different methods for interacting dynamically with the server.

Luckily, jQuery handles these differences.

AJAX

jQuery makes using AJAX easy

There are only a handful of AJAX functions in jQuery, and most of those are just useful wrapper functions.

- load
- getJSON
- ajax (next lecture)

Loading Remote HTML

The easiest of the jQuery Ajax functions: load

The load method will grab an HTML file off the server and insert its contents within the current web page.

You can load either static HTML files or dynamic pages that generate HTML output.

load

```
$('div:first').load('test.html');
```

This dynamically inserts the entire contents of the test.html file into the first div on the page.

You can use any selector to decide where the HTML should go, and you can even load it into multiple locations at the same time.

Picking HTML with Selectors

The load action lets you specify a jQuery selector as part of the URL string.

Only page elements that match the selector will be returned.

The format for using selectors with load is very simple: you just add the selector string after the filename you wish to load, separated with a space:

```
$('#target').load('page.html div:first');
```

NB

Do not use this method for all your webpages. Your website still needs to consist of separate webpages.

Use this when you have different pages that contain similar information.

Limitations of the load Function

For security reasons, the content you load must be stored on the same domain as the web page from which your script is running.

Web browsers typically stop you from making requests to third-party servers in order to prevent cross-site scripting attacks: evil scripts being maliciously injected into the page.

Read more about XSS attacks here: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting

Enhancing Hyperlinks with Hijax

You normally have standard hyperlinks that will link to separate pages.

With Ajax you can now “intercept” the links when a user clicks on them, and instead of sending the user to a new page, you can load the information below the links.

This way, any users who visit our site without JavaScript will still be able to visit the new pages as normal links.

- Progressively enhancing hyperlinks in this manner is sometimes called hijax (hijack the hyperlinks with Ajax)

Enhancing Hyperlinks with Hijax

```
<ul id="movieList">
    <li><a href="Metropolis.html">Metropolis</a></li>
    <li><a href="BladeRunner.html">Blade Runner</a></li>
    <li><a href="TheThing.html">The Thing</a></li>
    <li><a href="Alien.html">Ailen</a></li>
</ul>
<div id="movie">
    Click on a movie above to find out more!
</div>
```

Metropolis.html:

```
<h1>Metropolis</h1>
<p id="description">
    Metropolis is a 1927 German expressionist science-fiction drama
    film directed by Fritz Lang
</p>
```

Enhancing Hyperlinks with Hijax

Select all the links inside the unordered list

Prevent the default event from occurring (which would be to follow the link and load the target page).

Grab the original destination of the link (by retrieving the href attribute of the link we clicked on), and pass it on to the load function.

```
$('#movieList a').on("click", function(e){
      let url = $(this).attr('href');
      $('#movie').load(url);
      e.preventDefault();
});
```

Enhancing Hyperlinks with Hijax

This code works perfectly, but injecting the entire contents of the page turns out to be a bit problematic.

The problem is that we don’t necessarily want to load the entire page via Ajax, just the bits we’re interested in.

Enhancing Hyperlinks with Hijax

The selector you use can be as complex as you like, letting you pick out very specific parts of the page to pull in.

```
$('#movieList ul a').on("click", function(e){
      let url = $(this).attr('href') + ' #description';
      $('#movie').html('loading...').load(url);
      e.preventDefault();
});
```

Be sure to include a space before the hash to separate it from the filename.

Now only the description div is loaded in.

For the loading indicator: replace the target element’s text with “loading …” before you call load.

Note: The Entire Page Is Still Loaded

You might think that specifying a selector could be a way to reduce the bandwidth of your Ajax calls.

It doesn’t work that way, unfortunately.

Regardless of whether or not you add a selector, the entire page is returned, and the selector is run against it.

When finished loading...

Since .load is an AJAX method, you can’t manipulate elements or add events synchronously, for example:

```
$('#movie').load('otherPage.html div.otherContent');

$('div.otherContent').addClass("myClass");
// This will not work, since div.otherContent does not
// yet exist when the above is called
```

When finished loading...

Instead, you must use callback methods when chaining events, for example:

```
$('#movie').load('otherPage.html div.otherContent', function(){
      $("div.otherContent").addClass("myClass");
      // I can now do whatever I want with the content that I’ve
      // loaded, since I know that it has finished loading
});
```

When finished loading...

Or, using a Promise...

```
let loadPage = url => {
    return new Promise((resolve, reject) => {
        $('#movie').load(`${url} div.otherContent`, resolve);
    });
}

$('#movieList').on('click', 'a', function(e){
    let url = $(this).attr('href');
    loadPage(url).then(() => {
        $('div.otherContent').addClass('myClass');
    });
    e.preventDefault();
});
```

When finished loading...

Note that loadPage is defined as a function which returns a Promise

Because of this, we have to execute loadPage before we can call .then, since the result of this function is the Promise we call .then on

getJSON

jQuery provides us with a method for fetching JSON data: $.getJSON

The basic version of this method accepts a URL and a callback function.

The URL is a service that returns data in JSON format.

```
$.getJSON('person.json', person => {
     console.log(person.firstName);
});
```

getJSON

Or, using Promises…

```
let loadPerson = new Promise((resolve, reject) => {
     $.getJSON('person.json', json => {
          resolve(json);
     });
});

loadPerson.then(
     person => console.log(person.firstName)
);
// since we don’t ever call reject on the Promise
// we can omit the second parameter function
```

Check if JSON file is valid

```
let getPerson = new Promise((res, rej) => {
     $.getJSON('/foo/bar.json')
     .done(data => {
          resolve(data);
     })
     .fail(() => {
          reject("error");
     });
});
```

Loading External Scripts with $.getScript

Cramming desktop-style functionality into our web apps is great, but it suffers a downside: as our applications become larger, download time increases proportionately.

We need our application to be as snappy as possible.

$.getScript

The $.getScript function will load and execute a JavaScript file via Ajax.

How does this help us out?

> We can load a minimal amount of our code when the page loads and then pull in further files as we need them.

This is particularly useful if we require any plugins that have a sizeable footprint, but regardless of size you should try to delay the loading of your code until it’s absolutely necessary.

$.getScript

Example: Color plugin (for Color Animation)

```
$.getScript('http://github.com/jquery/jquerycolor',
() => {
      $('body').animate({'background-color': '#fff'},
      'slow');
});
```

$.ajax

All of the jQuery’s Ajax functions uses the $.ajax method.

All of jQuery’s Ajax functions are simply wrappers around the $.ajax method.

The $.ajax method is the heart of jQuery’s Ajax abilities and is the most powerful and customizable Ajax method available to us.

Next lecture…

References

jQuery: Novice to Ninja by Earle Castledine and Craig Sharkie

https://api.jquery.com/load/

https://api.jquery.com/jQuery.getJSON/

https://api.jquery.com/jQuery.getScript/