Client-Side Web Programming: JavaScript (Part 3)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - JavaScript libraries for client-side web programming
  - jQuery
  - JavaScript arrow functions

Agenda

- **JavaScript libraries**
- jQuery
- Arrow functions

JavaScript Libraries: Motivation

- **Problem 1:**
  - Many incompatibilities among browsers
  - JavaScript code should account for all/most/many variations
- **Problem** 2:
  - JavaScript/AJAX code uses common patterns and often is repetitive

JavaScript Libraries: Motivation

Problem 1:

Many incompatibilities among browsers

DOM, JavaScript, and HTML versions may differ

JavaScript code should account for all/most/many variations

Becoming less important

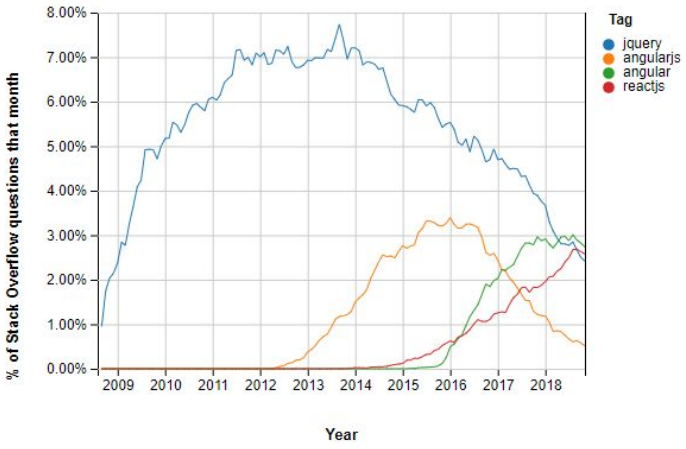
Problem 2:

JavaScript/AJAX code uses common patterns and often is repetitive

JavaScript Libraries: Motivation

- **Solution:**
  - Use a JavaScript **library**
    - **jQuery**, AngularJS, Angular, **React**, …

JavaScript Libraries



As of May 2019, according to
https://divante.com/blog/top-10-popular-javascript-frameworks-2019/

6

JavaScript Libraries

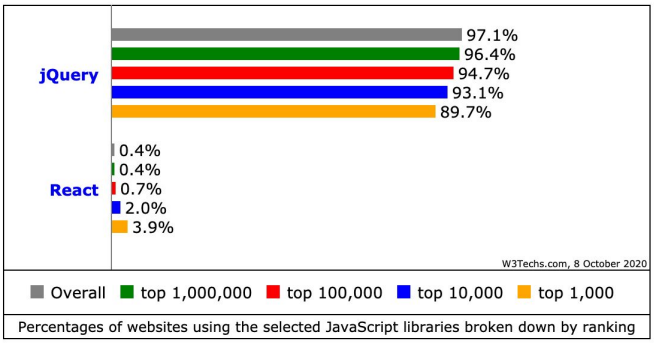
[see slide]

Be careful

Is it the case that:

(1) React is now more popular than jQuery. Or
(2) jQuery is older than React, and so programmers have fewer questions about jQuery
(3) jQuery is much easier to use than React, and so programmers have fewer questions about jQuery

JavaScript Libraries



As of October 2020, according to
https://w3techs.com/technologies/comparison/js-jquery,js-react

7

JavaScript Libraries

[see slide]

In fact, in an absolute sense jQuery is much more popular than React

It would be interesting to see such data for NEW applications

Agenda

- JavaScript libraries
- **jQuery**
- Arrow functions

jQuery

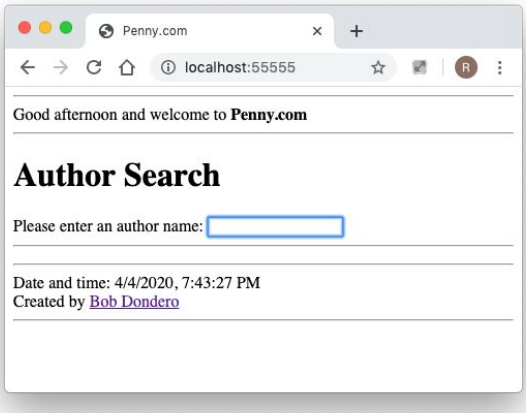
- **Who**: John Resig
- **When**: 2006
- **Why**: Simplify JavaScript syntax for finding, selecting, and manipulating DOM nodes

jQuery: Fundamentals

- jQuery statement syntax:
  - **`$(selector).action()`**
    - **`$`**
      - Indicates that this is a jQuery statement
    - ***`selector`***
      - Selects HTML element(s)
      - As in CSS; covered soon
    - ***`action`*`()`**
      - Specifies an action to be performed on the selected element(s)

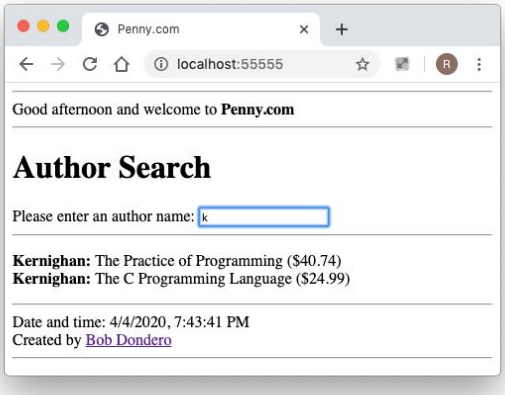
jQuery: Example

- See **PennyJQuery** app



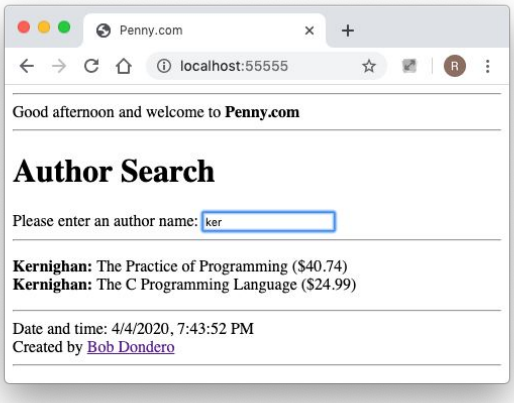
jQuery: Example

- See **PennyJQuery** app



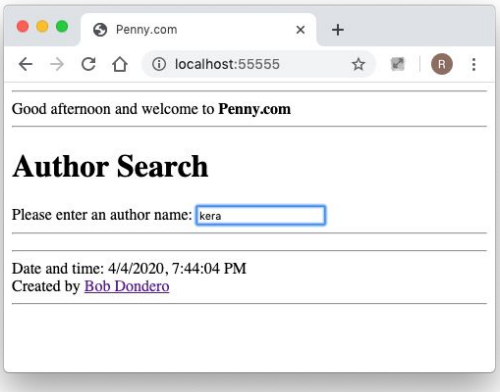
jQuery: Example

- See **PennyJQuery** app



jQuery: Example

- See **PennyJQuery** app



jQuery: Example

- See **PennyJQuery** app (cont.)
  - runserver.py
  - penny.sql, penny.sqlite
  - book.py, database.py
  - **penny.py** (same as previous)
  - **index.html**

15

jQuery: Example

[see slide]

Code notes: penny.py
Same as previous

Code notes: index.html
Uses jQuery to access DOM nodes, perform AJAX

See dynamic trace in following slides

jQuery: Example

- See **PennyJQuery** app (cont.)
  - Dynamic trace:
    - **Step 1**: Browser sends HTTP request to app for / or /index
    - **Step 2**: App receives HTTP request, calls index(), and delivers page (index.html) to browser

jQuery: Example

- See **PennyJQuery** app (cont.)
  - Dynamic trace (cont.):
    - **Step 3**: Browser receives HTTP response, renders page, interprets JavaScript code, calls `setup()`, and installs `getResults()` as event handler for `input` events on input element
    - **Step 4**: User types 'k' in input element

jQuery: Example

- See **PennyJQuery** app (cont.)
  - Dynamic trace (cont.):
    - **Step 5**: Browser calls `getResults()`, sends HTTP request to app for /searchresults?author='k', and remains responsive to user input
    - **Step 6**: App receives HTTP request, calls `search_results()`, and delivers HTML fragment (book list) to browser

jQuery: Example

- See **PennyJQuery** app (cont.)
  - Dynamic trace (cont.):
    - **Step 7**: Browser receives HTML fragment, queues call of `handleResponse(response)`

jQuery: Example

- See **PennyJQuery** app (cont.)
  - Dynamic trace (cont.):
    - **Step 8**: Eventually, browser calls `handleResponse(response)`, assigns response's text to `innerHTML` of `resultsParagraph`
    - **Step 9**: Browser displays book list

jQuery: Fetching the Library

- **How to fetch the jQuery library…**
- **Option 1**
  - Command browser to fetch jQuery library from jQuery site
  - (Pro) Likely to be cached in browser
  - (Con) What if not cached and jQuery site is down???!!!

jQuery: Fetching the Library

- **Option 2**
  - Ahead of time, download jQuery your to your website
  - Command browser to fetch jQuery library from your website
  - (Con) Less likely to be cached in browser
  - (Pro) Always works!

jQuery: Observation

- jQuery makes it easy to access DOM

Without jQuery:

```
let author =
   document.getElementById('authorInput').value;
```

With jQuery:

```
let author = $('#authorInput').val();
```

=> access by id

jQuery: Observation

- jQuery makes it easy to program AJAX

  Without jQuery:

```
function processReadyStateChange()
{  ...
   let resultsParagraph =
      document.getElementById('resultsParagraph');
   resultsParagraph.innerHTML = this.responseText;
}

function getResults()
{  ...
   request = new XMLHttpRequest();
   request.onreadystatechange = processReadyStateChange;
   request.open('GET', url);
   request.send();
}
```

jQuery: Observation

With jQuery:

```
function handleResponse(response)
{
   $('#resultsParagraph').html(response);
}
function getResults()
{  ...
   request = $.ajax(
      {  type: 'GET',
         url: url,
         success: handleResponse
      }
   );
}
```

jQuery: Observation

- jQuery makes it easy to write apps for old browsers
  - Old versions of Microsoft IE don't implement:
    - XMLHttpRequest objects
    - addEventListener() method
  - jQuery detects and compensates

26

jQuery: Observation

[see slide]

Old versions of Microsoft IE use ActiveXObject objects instead of XMLHttpRequest objects

Old versions of Microsoft IE use attachEvent() instead of addEventListener()

jQuery: Summary

- jQuery summary
  - Useful
    - Easy to access DOM
    - Easy to use AJAX
    - Handles old browsers
  - Easy to learn and use
    - Especially if you know CSS
    - Lots of web info
  - Extremely popular

jQuery: Commentary

- jQuery commentary
  - Don't use "raw" JavaScript in browsers
  - Use either
    - jQuery, or
    - A more elaborate library

Some advanced topics...

29

In this lecture the topics beyond this point are advanced topics

They’re certainly worth knowing, but…

You won’t need to know them to complete the assignments

You won’t need to know them to complete your project -- unless you choose to use them!

Agenda

- JavaScript libraries
- jQuery
- **Arrow functions**

Aside: this

- Recall from JavaScript lectures…
- **Question**: How is `this` bound within a function `f()`?
- **Answer**: Depends upon how `f()` is called:

Function Call	Binding of `this`
`f()`	In `f()`, `this` is `undefined`
`obj.f()`	In `f()`, `this` is bound to `obj`
`new f()`	In `f()`, `this` is bound to a new empty object
`f.call(obj)`	In `f()`, `this` is bound to `obj`

Arrow Functions

- ***Arrow function def expressions***
  - Informally ***arrow functions***
  - Arrow functions vs non-arrow functions:
    - More succinct
    - Same semantics - mostly!!!

Arrow Functions

- See **arrow.js**
  - Arrow functions
    - With 1 parameter
    - With 2 parameters
    - With 0 parameters

```
$ node arrow.js
25
25
25
25
25
30
30
30
30
hi
hi
hi
$
```

33

Arrow Functions

First let's look at some arrow functions in JavaScript programs running on Node.js

Then we'll look at some arrow functions in JavaScript programs running in the browser, in React applications

[see slide]

Arrow Functions

- See **arrow.js** (cont.)
  - In this program…
    - Arrow functions have **same** semantics as non-arrow functions
  - In other programs…
    - Arrow functions and non-arrow functions have **different** semantics

Aside: setInterval & setTimeout

- In browsers:
  - `window.setInterval(f, ms);`
    - Call `f` every `ms` milliseconds
    - We have seen
  - `window.setTimeout(f, ms);`
    - Call `f` after `ms` milliseconds
- In Node.js:
  - `setInterval(f, ms);`
    - Call `f` every `ms` milliseconds
  - `setTimeout(f, ms);`
    - Call `f` after `ms` milliseconds
    - We will use now…

Arrow Functions

- **Fact 1**: In a non-arrow function…
  - The value of `this` is determined **dynamically**
    - Based upon the call
    - `o.f()`
      - In the function `this` is bound to `o`
    - `f()`
      - In the function `this` is undefined

Arrow Functions

- See **arrow1.js**
  - See example execution
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls non-arrow function
      - As `f()`, not as `o.f()`
    - In non-arrow function, `this` is undefined

Arrow Functions

- **Fact 2**: In a non-arrow function…
  - The value of an ordinary variable is determined **statically**
    - Based upon program block structure

Arrow Functions

- See **arrow2.js**
  - See example execution
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls non-arrow function
    - In non-arrow function, `this` is undefined
      - But the non-arrow function doesn't use `this`!

Arrow Functions

- **Fact 3**: In an arrow function…
  - The value of `this` (and any ordinary variable) is determined **statically**
    - Based upon program block structure

Arrow Functions

- See **arrow3.js**
  - See example execution
  - Notes:
    - Global code calls `main()`
    - `main()` calls `blueCar.writeColor()`
    - `blueCar.writeColor()` calls `setTimeout()`
    - `setTimeout()` calls arrow function
    - In arrow function, `this` is bound to `blueCar`

Arrow Functions: Summary

- **Question**: Why use arrow functions?
- **Answer 1**: They're more succinct
- **Answer 2**: `this` is defined statically

- Arrow functions often are appropriate as callback functions

Summary

- We have covered:
  - JavaScript libraries for client-side web programming
  - jQuery
  - JavaScript arrow functions