Single Page App Day

Brought to you by Chariot Solutions

What are Single Page Applications?

2.1

Single Page Application Division of Labor

- Rich user interface
- Client-side logic
- Hold state, cache data
- Make requests to server(s)

- Respond to client requests
- Handle business logic
- Generally stateless
- Provide security

2.2

Three popular SPAs

- `Angular` - Google-created *Framework* that includes APIs for many chores
- `React` - A Component library written from the ground up for a single purpose, has many "friends"
- `VueJS` - A "better, simpler Angular?"

Why an SPA?

- Rich front-end experience
  - Snappier responses
  - No full-page refreshes
  - Better indications of activity
- Full client application
  - Leverage browser computing power
  - Reduce server complexity
  - Have a true conversation with the server

2.4

Mad SPA Skills...

"Modern" JavaScript

- A moving target...
- `JavaScript` is actually officially ECMAScript
- `ECMAScript 2018` is the current version
- Most developers develop in `ECMAScript 2015` or higher and translate it to an older version

Say it: ECMAScript = ES = JavaScript


3.3

Babel - babeljs.io

`Babel` "transpiles" ECMAScript 2018 → ECMAScript 5 (which every browser interprets) or higher...

- Compiles down to ECMAScript 5
- Can compile to newer versions of ECMAScript

Design and UI Skills

- User Experience Design
- HTML and Cascading Style Sheets

3.5

JavaScript Skills

- Shed the jQuery knowledge, it’s obsolete
- JavaScript is NOT Java
- The event loop controls performance

3.6

Key JavaScript Skills (this IS a JavaScript day...)


3.7

Prototypes ARE Inheritance Model

```
1  var prototype = {
2    sayHello: function() {
3       console.log(this.name);
4    }
5  };
6
7  var obj = Object.create(prototype);
8  obj.name = 'Ken';
9
10 console.log(obj.sayHello());
11
12 // output: 'Ken'
```

3.8

Class definitions (i.e. constructor functions)

```
1  class Person {
2    constructor(first, last) {
3       this.first = first;
4       this.last = last;
5    }
6  }
```

Turns into...

```
1  function Person(first, last) {
2    this.first = first;
3    this.last = last;
4  }
```

Classes create constructor functions!

3.9

Classes and Inheritance

```
1  class Customer extends Person {
2
3    constructor(first, last, orders) {
4      super(first, last);
5      this.orders = orders;
6    }
7
8    reportOrders() {
9      return `Customer ${this.first} ${this.las
10             has ${orders.length} orders`;
11   }
12 }
```

The Customer Prototype

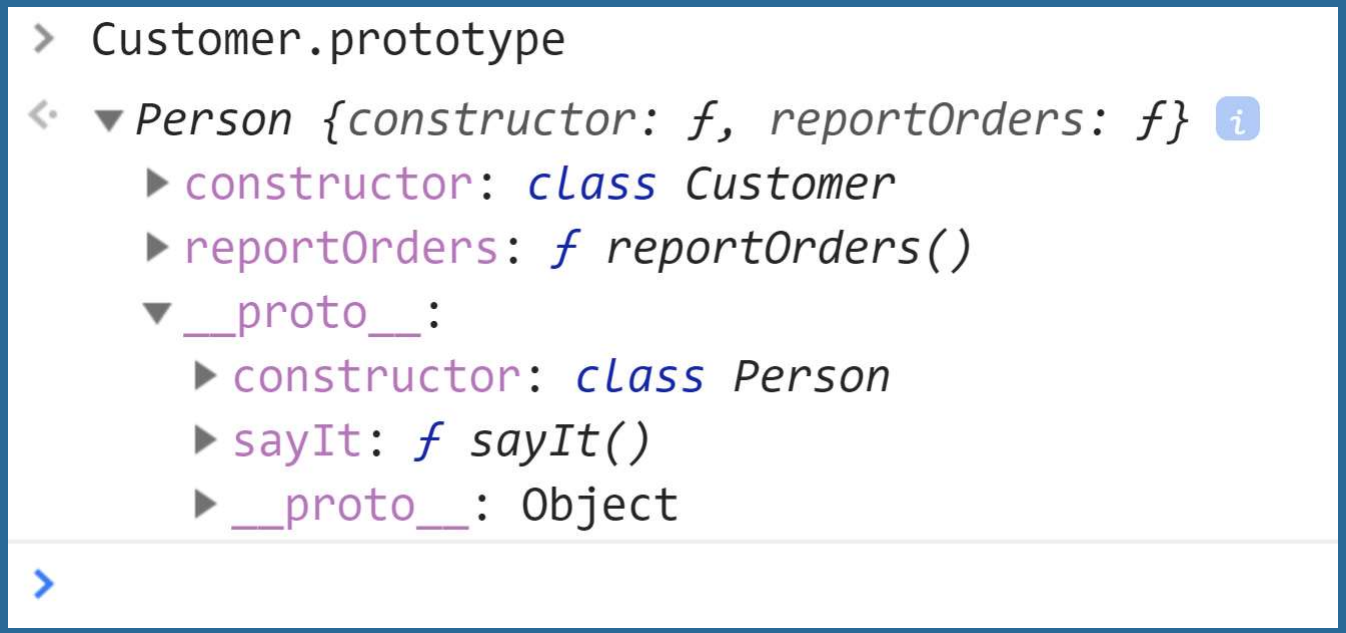

Figure 1. Prototypes are objects…

JavaScript is *function driven* and supports objects…

3.11

`this` is a mess...

Without arrow functions