

Intro to Client-Side Apps With AngularJS

Philly Tech Week Dev Days, 5 April 2014



Agenda

- Software
- Background: HTML and JavaScript
- Tutorial: AngularJS PhoneCat
- ???

AngularJS

<http://angularjs.org/>

- Templating
- Data Binding
- “Single-page” Applications

JQuery

<http://jquery.com/>

- AJAX
- DOM Manipulation
- Event Handling

Node.JS

<http://nodejs.org/>

- JavaScript-based, Event-driven WebServer
- Simple HTTP server out-of-the-box
- Add-on modules provide additional capabilities
- *Used for AngularJS PhoneCat tutorial*

Sandboxes

JSFiddle: <http://jsfiddle.net/>

Plunker: <http://plnkr.co/>

Intro to Client-Side Apps With AngularJS

HTML

Hypertext Markup Language

A markup language uses *markup codes* interspersed with *text content*

In HTML, markup codes are called *tags*

Tags start with `<`, end with `>`, may contain attributes

Example: `<p color='red'>`

Well-formed HTML matches every start tag with an end tag

Example: `<i>this text is italicized</i>`

Example: Hello World

```
<!doctype html>
<html>
  <head>
    <title> Example HTML Page </title>
  </head>
  <body>
    <p>
      <b>Hello</b>,
      <i>World</i>
    </p>
  </body>
</html>
```

Common HTML Tags

<code><div></code>	Division (used to group/position other elements)
<code><p></code>	Paragraph
<code></code>	Span (used to style text within paragraph)
<code></code>	Image
<code></code>	Anchor (link)
<code></code>	Unordered (bulleted) List
<code></code>	Ordered (numbered) List
<code></code>	List item (usable only inside <code></code> or <code></code>)

Table Tags

<code><table></code>	Encloses the entire table
<code><tr></code>	Encloses a single row in the table
<code><th></code>	Table cell that's used as a header
<code><td></code>	Table cell that's used as data

*Tables should be used for tabular data, **not** formatting*

ID and Class

Every HTML tag supports `id` and `class` attributes

`id` must be unique

`class` can be used by multiple elements

A single element can have multiple classes

Used to select elements in CSS and JQuery

Cascading Style Sheets (CSS)

Separates appearance from content

Usually loaded as a separate file

May be specified in `<style>` element

Can apply different formatting depending on output device

Example: “desktop” site may use fixed header, mobile site may allow header to scroll out of view

“Responsive design”

CSS Example

Set default font for all text in page

```
BODY { font-family: Calibri, "Times New Roman", serif; }
```

Render all text inside `` in green

```
STRONG { color: green; }
```

Render everything with class “highlight” in red

```
.highlight { color: #FF0000; }
```

Hide the `<div>` with id “hideme”

```
#hideme { visibility: hidden; }
```

Intro to Client-Side Apps With AngularJS

JavaScript



JavaScript

An object-oriented + functional programming language

Introduced in 1995 with Netscape Navigator 2.0

Popularity increased in early/mid-2000s with introduction of AJAX

Increasing use as server-side language circa 2010

JavaScript is not Java

But JavaScript *is* a registered trademark of Oracle (formerly Sun)

They were released at about the same time, Netscape supported both

One early use of JavaScript was interaction with Java applets

JavaScript + Browser

Scripts can react to events, modify DOM

Libraries such as JQuery simplify this process

Loaded and executed dynamically

Either from file or inline

Script execution delays page rendering

Might try to modify parts of page that haven't rendered yet

Example: Hello World

```
<body>
  <button id='clickMe' onClick='clickHandler()'>Click me</button>
  <p id='fillMe'></p>

  <script type='text/javascript'>
    function clickHandler() {
      var button = document.getElementById("clickMe");
      button.style.visibility = "hidden";
      var para = document.getElementById("fillMe");
      para.innerHTML = "Hello, World";
    }
  </script>
</body>
```

JavaScript: The Bad Parts

Loosely typed, minimal error reporting

If something doesn't work, look for typo

`console.log` and `alert()` are your friends

Too easy to create global variables

`this` is context-dependent

“Truthiness” encompasses more than you might think

JavaScript: The Good Parts

< \$20

< 150 pages

Douglas Crockford

JavaScript Evangelist

“Discoverer” of JSON

Creator of JSLint

<http://www.crockford.com/>



Functions

The unit of modularization for JavaScript programs

Establishes scope for variables and embedded functions

“Module pattern” used to create namespace, minimize global variables

May be named or anonymous

Anonymous functions typically used as callbacks (event handlers)

Named functions are visible in enclosing scope (may be global)

Function Examples

```
// named function -- creates variable "add"
function add(x, y) {
    return x + y;
}
```

```
// anonymous function assigned to variable "doSomething"
var doSomething = function (x, y, fn) {
    return fn(x,y);
}
```

```
// function invocation, passing named function as parameter
var x = doSomething(1, 2, add);
```

```
// function invocation, passing anonymous function as parameter
var y = doSomething(1, 2, function(x,y) { return x * y; });
```

Variables

Holds a reference to some value

Number (all numbers are floating-point)

String

Boolean (although lots of things *act* like booleans)

Function (a function's name is also a variable)

Array

Object

Null

Undefined

Objects

A mutable, free-form collection of name/value pairs

Other languages call this a *dictionary* or *hash*

Objects may be nested to any depth

Members accessed by name

Dot notation: `foo.bar`

Bracket notation: `foo["bar"]`

May create piecemeal or as object literal

Object Examples

```
// create object literal with two members
var foo = { x: 12, y: 13 };

// access value from object
var z = foo.x + foo["y"];

// add new member -- a function
foo.addX = function(z) { return this.x + z };

// invoke function via object reference
foo.addX(17);
```

The Meaning of `this`

A reference to the “current” object

When a method is invoked via an object reference: the invoking object

When a function is invoked by name (sans object): the global context

Within an object literal: the global context

Best practice: when creating objects with functions, use “module” pattern

Ensures that `this` is the module

The “Module” Pattern

Object that exposes data and methods, created using anonymous function

- Establishes a namespace for embedded functions

- Allows hiding of internal data

Several variants

- Crockford returns object literal, requires invoking function

- Alternative uses `new`, relies on equivalence of object and function, does *not* hide data, but is slightly less verbose

Module Pattern Example

```
var module = (function() {  
    var x = 12;  
  
    function addX(y) { return x + y }  
  
    return {  
        addX: addX  
    }  
})();  
  
var foo = module.addX(17);    // 29  
var bar = module.x;          // undefined
```

Prototypal Inheritance

Every object has a “`prototype`” member

- The default prototype depends on the object type

- Can create new objects with specific prototype

If object does not define member, prototype checked

- Prototypes may themselves have prototypes

Updates to prototype affect all objects with that prototype

- Bad idea: change `Object.prototype` (or `String.prototype`, ...)

Prototypal Inheritance Example

```
// create the prototype
var proto = { x: 12, y: 13 };

// two objects that use the prototype, don't define their own members
var foo = Object.create(proto);
var bar = Object.create(proto);

// assigning foo.x overrides the prototype
foo.x = 17;

// this only updates bar.x (because foo now has its own x)
proto.x = 6;

// this updates both foo.y and bar.y
proto.y = 32;
```

JavaScript Object Notation

An object's data, represented as a string

Example: `{ "x": 12, "y": 13, "z": ["foo", "bar"] }`

Has long been used to return data from server

Libraries to convert to/from JSON available for most languages

With rise of JS-based servers, used to upload as well

Taking over from `url-form-encoded` data

Requires extra work on “traditional” servers

Truthiness and Logical Tests

“Truthiness” indicates presence of a value

“Not true” is literal `false`, `null`, `0`, or `undefined`

```
if (console) // may be true or false
```

```
if ("false") // true!
```

Two tests for equality

`==` converts arguments: `123 == "123" is true`

`===` does not convert: `123 === "123" is false`

Prefer `===` to avoid subtle bugs

Intro to Client-Side Apps With AngularJS

AngularJS



What is AngularJS?

A framework for building client-side applications

- Templating

- Two-way data binding

- Server provides static HTML and JSON data

Extends HTML using *directives*

- May be specified as attributes or new tags (I prefer attributes)

- Name starts with “ng”

- Multiple ways to spell (**eg**: ng-App, ngApp, ng:App)

Terminology

Model (aka `$scope`)

A “plain old JavaScript object” containing data to be rendered on page

May also have methods that interact with the model

Controller

Responsible for populating the model when page/view is loaded

View

General: the HTML (template) that is used to render the model

In single-page app: the part of the app that uses routing, partial templates

Terminology, cont

Service

A JavaScript object that provides shared functionality

Angular provides several services (eg, `$http`)

The application can create its own services, use dependency injection to install them in controllers as needed

Module

A named object that holds the app's controllers, services, directives, &c

Application code creates modules, registers them with Angular

Example

```
<!doctype html>
<html ng-app>
  <head>
    <script src="lib/angular.js"></script>
  </head>
  <body ng-controller='MyController'>
    <button ng-click='clickHandler()' ng-show='showButton'>Click me</button>
    <p ng-hide='showButton'>{{content}}</p>

    <script type='text/javascript'>
      function MyController($scope) {
        $scope.showButton = true;
        $scope.content = "";

        $scope.clickHandler = function() {
          $scope.showButton = false;
          $scope.content = "Hello, World"
        }
      }
    </script>
```

What Happens When Page Loads

1. Browser reads page, executes `angular.js` script
 - `angular.js` attaches a “page ready” event handler
2. Angular examines DOM, discovers `ngApp` and `ngController` directives, runs controller
 - `ngApp` optionally identifies module where controller is found
 - `ngController` identifies controller method; with no module, it's global
3. Angular examines page for templating commands and executes them
 - Scope of template is defined by `ngController`

ngApp

Directive that identifies the page as controlled by Angular

When Angular's post-load handler sees an `ngApp` directive, it starts processing the page

Only one `ngApp` may appear on a page; normally added to `<html>`

Optionally specifies the name of a module

The module must already have been created by application JavaScript

That module will be used to resolve controller methods, resources, &c

ngController

Associates a section of the page with specific model

You can have multiple controllers on one page -- but it's rare

Identifies a function to populate model

Preferred: the function is defined as part of an explicit module

For playtime only: if no module, then looks for global function

ngShow / ngHide

Shows or hides *current* element based on expression

Expression valuated against the scope

JavaScript “truthiness”

Changes visibility only, *does not modify DOM*

ngView

Used for single-page apps: identifies the part of a page where a partial template will be loaded

URL fragment ID (hashtag) identifies *route*

Example: `http://localhost:8000/app/index.html#/phones`

Application must explicitly configure routes

Each route has its own controller, “partial” template

Routes may include parameters, which are provided to controller

Event/Update Cycle

Angular captures events from all elements bound to model

When model changes, Angular rebuilds affected templates

Application code can trigger rebuild with `$scope.$apply()`

This is useful when integrating Angular into an application that already communicates with server using JQuery or other library

Dependency Injection

Angular tracks named objects and supplies them as parameters to controller and factory methods

Two ways to request injection:

```
// short form
```

```
myModule.controller('MyController', function($scope, $http) {  
    // ...  
})
```

```
// long form
```

```
myModule.controller('MyController', ['$scope', '$http', function(s, h) {  
    // ...  
}])
```

Must use long form if minifying!

Angular and Legacy Apps

Angular can take over part of a page, with legacy code handling the rest -- app can migrate to Angular over time

Internally, Angular uses “jQuery lite”

If application already uses “full” jQuery, Angular will use it too

Things to watch out for:

- If loading page fragments via jQuery, must explicitly bootstrap Angular
- If retrieving data via jQuery, must call `$scope.$apply()`
- Don't forget `doctype!`

Intro to Client-Side Apps With AngularJS

Tutorial

Overview

A catalog of Android phones

Progressively enhanced: from hardcoded list of names, to basic information retrieved from server, to multiple views

Available online: <http://docs.angularjs.org/tutorial>

Tutorial Workspace

Each step in tutorial has its own directory in `Workspace`

`step-00, step-01, ...`

All application files in the `app` subdirectory

Pages served by Node.js

Unpack the version of Node.js for your operating system and add to path

Run `./scripts/web-server.js` in tutorial directory

Connect to server: <http://localhost:8000/app/index.html>

step-01: Hardcoded HTML

Things to note:

- `<!doctype html>`
- Stylesheets (not used)

Things to try:

- Add class to list items, apply style

step-02: Simple Template

Things to note:

- Angular directives: `ng-app`, `ng-controller`, `ng-repeat`
- Controller to initialize model
- Module -- contains controller
- Templating (“handlebars”)

Things to try:

- Add a price field to model

step-03: Filtering

Things to note:

- Binding model variable to HTML element (`ng-model`)
- Added `filter` clause to `ng-repeat`
- Starting to use styles

Things to try:

- Restrict filter to phone name

step-05: Get Data from Server

Things to note:

- `$http` service
- Dependency injection
- Added `orderBy` clause to `ng-repeat`, filled by HTML drop-down

Things to try:

- Extract carriers from data payload, populate another drop-down, use as filter

step-07: Routing and Views

Things to note:

- Another JavaScript library: `angular-route.js`
- `index.html` no longer contains markup
- Module (`app.js`) now defines routes
- Multiple controllers
- Partial templates
- URL rewritten when page loads

Things to try:

- Bogus URLs, with and without “catch all” route

step-08: Routing and Views, cont

Things to note:

- Phone list link construction; `$routeParams`
- New data request for each detail page
- Hierarchical data
- Images use `ng-src` rather than `src`

Things to try:

- Cache phone data

step-10: Event Handlers

Things to note:

- Function to update `mainImageUrl`

Things to try:

- Add click handler that cycles main image