AngularJS, Part 2

Some Tips and Tricks

Using Server Data

Modules

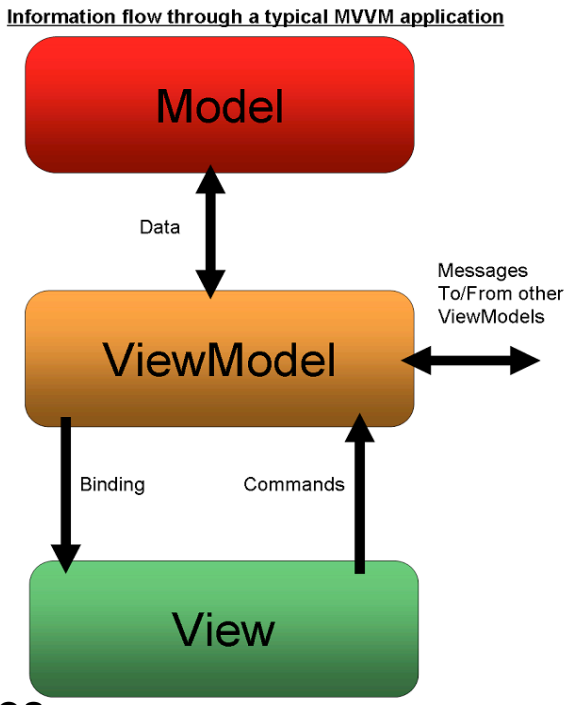
Front Controller Pattern

What does this have to do with REST?

AngularJS Reminder

Where were we?

- Angular supports SPAs – Single-Page Applications
- Angular is in the class of MVVM variants
- Angular supports 2-way data binding
  - When the ViewModel changes, the View changes
  - When the View changes, the ViewModel changes
  - Bye bye DOM manipulation and event listeners
- Angular is *intrusive*
  - We have to specify ng-app, ng-controller
  - We use special attributes, and probably special directives
  - $scope is a global *blackboard*
- Angular is *non-intrusive* (say wha???)
  - We can inject Angular into legacy code and limit or changes to the scope of the ng-app (decorated on any container element).
  - At least by using custom attributes we do not force new tags or re-interpreting of existing tags or attributes on developers



AngularJS Tips and Tricks

ng-show and ng-hide

- Allow you to dynamically display or make invisible a part of the DOM based on the outcome of an expression.
- From our MVVM perspective, this has the nice feature of allowing our widgets to appear cross-wired – a change in the state of one widget can make another widget (dis)appear
  - Hey, just like a desktop GUI!
- See Ch. 2 formValidation.html

CSS

- AngularJS allows you to apply CSS classes and styles dynamically using the {{…}} template notation
- For example, instead of making something invisible, make it gray and disable it.
- See deathRayMenu.html (Chapter 2 O'Reilly book)

AngularJS Tips and Tricks

Watches

- Observer pattern – decouple an object's change in state from other object's that want to enact a behavior based on the change
- You want to take some action in response to a change in an object property (data or computed function)

In Angular, a function on $scope:

- `$watch(watchFn, watchAction, deepWatch)`

where

`watchFn` – expression or function returning watch target

`watchAction` – what to invoke if watchFn changes

`deepWatch` – optional param, tells Angular to deep compare (expensive)

- see examples orderDiscountWatch*html (Chapter 2 of O'Reilly book)
- You can deregister a watch by calling the deregistering function which was returned on the original call to $watch

Modules

How do you normally write an application?

```
public static void main(String[] args)
```

- The runtime environment looks for a defined entry point
- As a *single-page application* (SPA), an AngularJS webpage simply adds its stuff, sets initial state, and goes.
- AngularJS has no defined entry point

Applications that grow over time need to modularize the code to support managing complexity.

- So far, we have one or more Controllers, each with its own $scope
- These objects act as “God” objects (code smell)

We aren’t going to create our own modules

- We are going to use builtin modules
  - $location – interacts with the browser’s location
  - $http – for HTTP communication (AJAX, or XHR)
  - $route – for (front) controller behavior

Using Server Data

Examples so far have hardwired data in the ViewModel ($scope) as part of the Controller constructor

- But really, we want to do some combination of:
  1. Initializing the ViewModel from dynamic server-side data
  2. Wiring some parts of the ViewModel to the Model

To do #1:

- You could generate the data initialization code dynamically on the server, but of course you would have to hit a server-side resource
- You could retrieve the data via an AJAX request

To do #2:

- Bind a function in the template that does the AJAX XHR

Example of #1

```
// full example is in talkingToServers (O'Reilly Chapter 2)
// run node server.js from same dir as this file (index.html)
// Then go to browser and type in http://localhost:3000

// rest of page left off, it is as you would expect.
<body ng-controller="ShoppingController">
    <h1>Shop!</h1>
    <table>
      <tr ng-repeat="item in items">
        <td>{{item.title}}</td>
        <td>{{item.description}}</td>
        <td>{{item.price | currency}}</td>
      </tr>
    </table>
    <script>
      function ShoppingController($scope, $http) {
        $http.get('/products').success(function(data, status, headers, config) {
          $scope.items = data;
        });
      }
    </script>
  </body>
```

Reminder: The Front-Controller Pattern

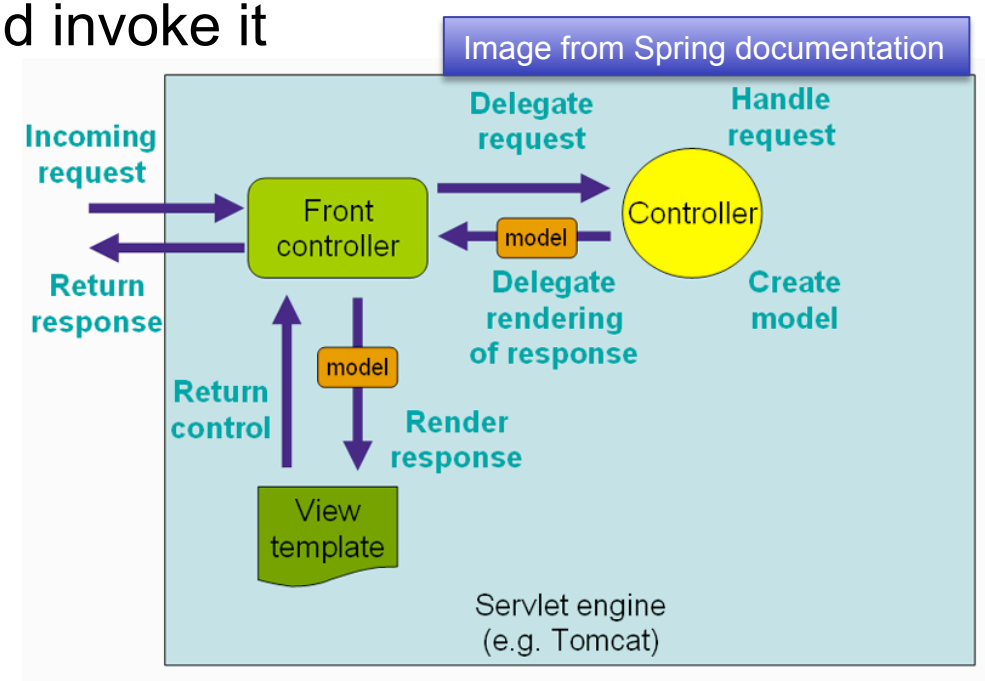
We have 2 related problems:

1. We want to bind to Model data in our ViewModel (or at least have some defined coupling to it for CRUD).
2. We have some behaviors that are distributed between the client and the server-side for a variety of reasons.

- We can do AJAX calls via $http for the 1st problem
- For the 2nd, we need a way to determine if the behavior is on the client or on the server and invoke it

The Front Controller Pattern

- Recall I recommended this as a way to manage SPA functionality versus behavior on the server
- The $route module will help us out



$route

In server-side MVC, we use URLs to determine behaviors

- One of 2 patterns typically used:
  1. For each action, use an “action” parameter
  2. For each action use a different URL (different servlet)

The former is closer to a Front Controller on the server

- But now we want to move this logic down to the client
- In AngularJS, a $route can manage sub-pages for each action and map each to a URL
- This basically violates strict SPA
- See example aMail
- We can create a $route that proxies calls to somewhere on the server
  - This is pretty much the proxy-like thing we did for data!

Summary

AngularJS is a front-end framework for creating MVC-style (MVVM) applications within the browser

- Uses a single-page application (SPA) model
- Provides a simple 2-way binding facility

Mapping the VM to the M

- Like you would do in POJ: write some AJAX
- Angular's $http module helps you with this; you simply write the initialization and/or other RUD calls and bind to $scope

Where's the C?

- As an SPA, there is no place you can point to and say, "there's the Controller"
- You can mimic such behavior using $route
  - This allows you to map new action requests to new Views as "sub-pages"
  - You can add the "F" by mapping some actions to server behaviors
  - Of course if you re-render the whole page from the server-side then you basically have to re-load the entire app (or its next step)