JQuery and AngularJS

WS Slides #3

jQuery Overview

JavaScript library by John Resig
- Cross browser: Trying to eliminate differences
- Free, Open Source
- Very good documentation

Features
- Much better DOM, DOM Events and CSS API's
- Better AJAX API
- Effects, animations
- ...

To add a JS library, just download the JS file and put it somewhere under Web Pages. Include in HTML-page in correct order

jQuery Philosophy

1. Find some DOM nodes using CSS selectors. Nodes will be "wrapped" in the jQuery object (objects enhanced with jQuery methods)

2. Do something with them using jQuery methods (instead of native JS API)
- Chain multiple method calls to process the set of elements

3. Don't need to explicitly traverse the set. Use implicit iteration (each()-method) to get final result
- Possible to get "previous" set back

jQuery API

API represented as JQuery object (abbreviated to $)

```
// Any jQuery statement/expression starts
JQuery. ...

// Same as above but quicker
$. ...
```

Deferring Execution

Any JavaScript should normally execute after DOM fully constructed

- To defer execution until DOM is ready, if using jQuery, wrap code like below

```
// Function called when DOM ready (wrapping document in jQuery)
$(document.ready(function(){
    // Code to execute when DOM ready
});
```

Or (simpler, same behaviour as above)

```
// Wrapping a function
$(function(){
    // Code to execute when DOM ready
});
```

jQuery DOM Elements

Find

```
// Ue CSS selector to get a wrapped DOM element)
var ul = $("#myList");

// Selecting in DOM (many more methods)
var children = $("#myList").children();
var parent = $("#myList").parent();
```

Add

```
// Add input element to DOM
$("<input id='btnClose' type='button'
                           value='Save'/>").appendTo('#popUp');
```

Modify

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

Delete

```
$('.productTableBody tr').remove();
```

Input Elements

Often need values of input, select and textarea elements

```
// Get some user input
var value = $("#myInput).val();
```

Manipulate Attributes and CSS

Manipulating attributes and CSS

Attribute

```
$('#greatphoto').attr('alt');  // Get
$('#greatphoto').attr('alt', 'Beijing Brush Seller'); // Set
```

CSS

```
$('#mydiv').css('color') // Get
$('#mydiv').css('color', 'green')  // Set
```

jQuery Traversing

Traversing

```
$("#myList").children().each( function(index, element) {
        $(this).css("color", "red");
});
```

Dissection

- jQuery will find all children to element with id “myList” and traverse the list
- Anonymous callback method passed to each(). Will execute for each child element, will set color for each child
- $(this) is the wrapped child element ("this" only is the current DOM element)

Wrapped or Not

Sometimes the underlying DOM nodes are returned (non wrapped)

```
// Get will unwrap and return underlying DOM node
var firstChild = $("#myList").children().get(0);

// Can’t use JQuery methods, not a wrapped object
firstChild.children()    // BAD!!!
```

When do we have a wrapped set, when do we have to original DOM node(s)?
- Check documentation!

JQuery DOM Events

Event names similar to native DOM API

// DOM level 0 names (skip “on” to get JQuery name)

onclick, The event occurs when the user clicks on an element

ondblclick, The event occurs when the user double-clicks on an element

onmousedown, The event occurs when a user presses a mouse button over an element

onmousemove, The event occurs when the pointer is moving while it is over an element

onmouseover, The event occurs when the pointer is moved onto an element

onmouseout, The event occurs when a user moves the mouse pointer out of an element

...many more...

jQuery Event Setup

Static elements

```
   // No () for listener functions!!!
   $(function(){    // Execute when DOM ready
      $("#btn1").on("click", listeners.click); // Supply listener
func.
   });
```

Dynamic elements (with id myBtn created in future)

```
 // Connect (in future)
 $(function(){
  $(document).on("dblclick","#myBtn", listeners.dblclick);
 });
```

jQuery Event Handling

Events handled by callback functions

```
// Setup
$("#txt").on("mouseenter", listeners.mouseEnter).
    on("mouseleave", listeners.mouseLeave ).
    on("focus", listeners.focus);

// Handling
var listeners = (function(){
    var color;
    mouseEnter: function(){
                color = $(this).css("background-color");
                $(this).css("background-color", "gray");
    },
    mouseLeave: function(){
                $(this).css("background-color", color);
    },
    focus: function(){
       ...
    },
})();
```

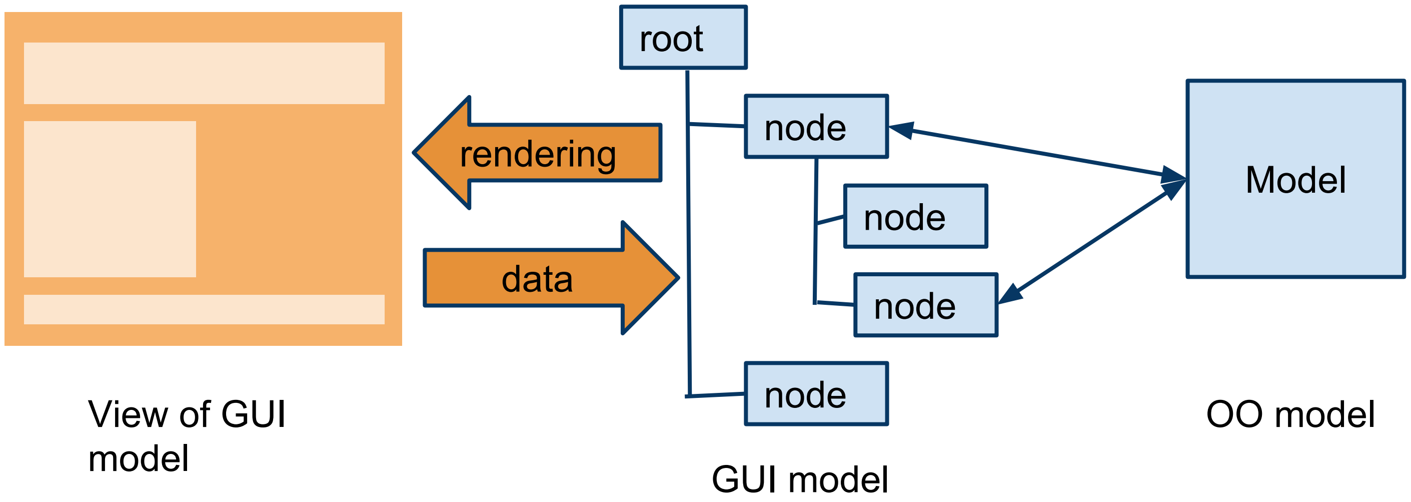
AJAX with jQuery

Possible but we prefer using Angular

What Model?

Confusing use of “model”

- We have the “real” OO model, the model of the core problem
- Also when talking GUI there’s a “model” ...
- ...the model behind the GUI, holding GUI data. Input output channel to real model



AngularJS

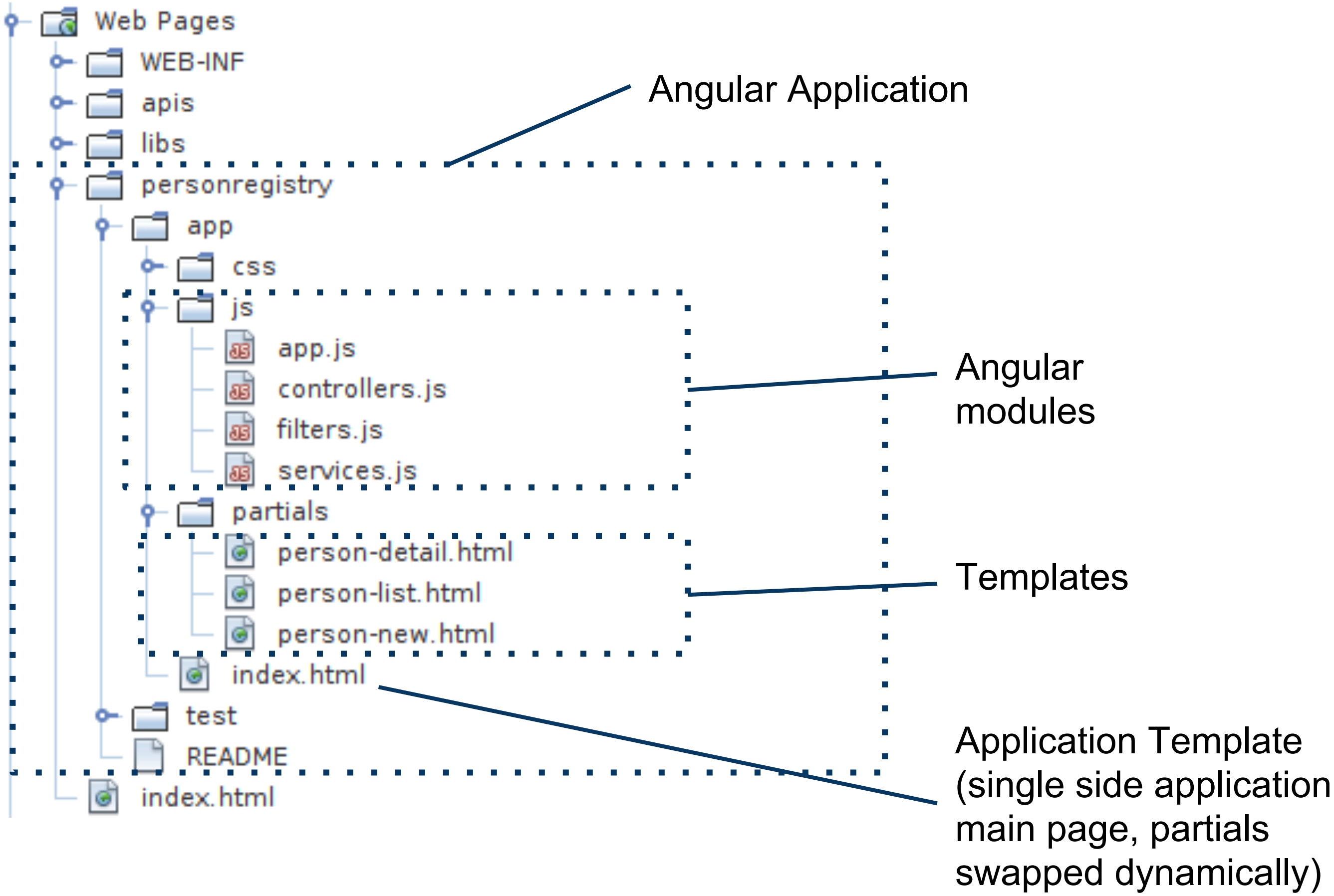
JavaScript client side MVC framework developed by Google. Conceptual Overview

Basically

-Templates = HTML enhanced with Angular attributes (smarter HTML)
- Modules
- Dependency injection
- An MVC structure
- "Data binding", JS code and HTML page automatically in sync (no need for DOM manipulation code)
- Event Handling
- AJAX and much more ...

Development Environment

Angular application development structure



- Web Pages
  - WEB-INF
  - apis
  - libs
  - personregistry
    - app
      - css
      - js
        - app.js
        - controllers.js
        - filters.js
        - services.js
      - partials
        - person-detail.html
        - person-list.html
        - person-new.html
      - index.html
    - test
    - README
  - index.html

Angular Application

Angular modules

Templates

Application Template (single side application main page, partials swapped dynamically)

Templates

HTML that contains Angular-specific elements and attributes

- **Directive** An attribute or element that augments an existing DOM element or represents a reusable DOM component
- **Markup**, The double curly brace notation **{{ }}** to bind expressions to elements is built-in Angular markup.
- **Filter**, Formats data for display.
- **Form controls**, Validates user input.

Templates are the V parts of MVC

Example Template

Part of a template with Angular **directives** and markup (and Bootstrap classes)

```
...
<table class="table">
            <thead>
                <tr>
                    <td>Id</td><td>Name</td><td>Age</td>
                </tr>
            </thead>
            <tr ng-repeat="person in persons | filter:query" ng-class-odd="'odd'"
                ng-class-even="'even'" >
                <td>{{person.age}}</td>
                <td><a href="#/persons/{{person.id}}">{{person.fname}}</a></td>
                <td>{{person.age}} </td>
            </tr>
        </table>
        <div class="btn-group">
            <button class="btn btn-default" ng-disabled="currentPage === 0"
                    ng-click="currentPage = currentPage - 1">
                Previous
            </button>
            <button class="btn btn-default"
                    ng-disabled="currentPage >= count / pageSize - 1"
                    ng-click="currentPage = currentPage + 1">
                Next
            </button>
        </div>
```

Angular Partials

Partials are portions of the page, swapped in and out

- Have seen how to use JSP partials (<jsp:include …)
- Partials need not be full pages, a div with directives and markup

In Angular solved by $routeProvider

```
// Mapping URL to partials
$routeProvider.
    when('/persons', {
        templateUrl: 'partials/person-list.html',
        controller: 'PersonListCtrl'
     })...

// In main page
 <div class="view-container">
     <!-- Include the partial for the current route -->
             <div ng-view ></div>
</div>
```

AngularJS Scope

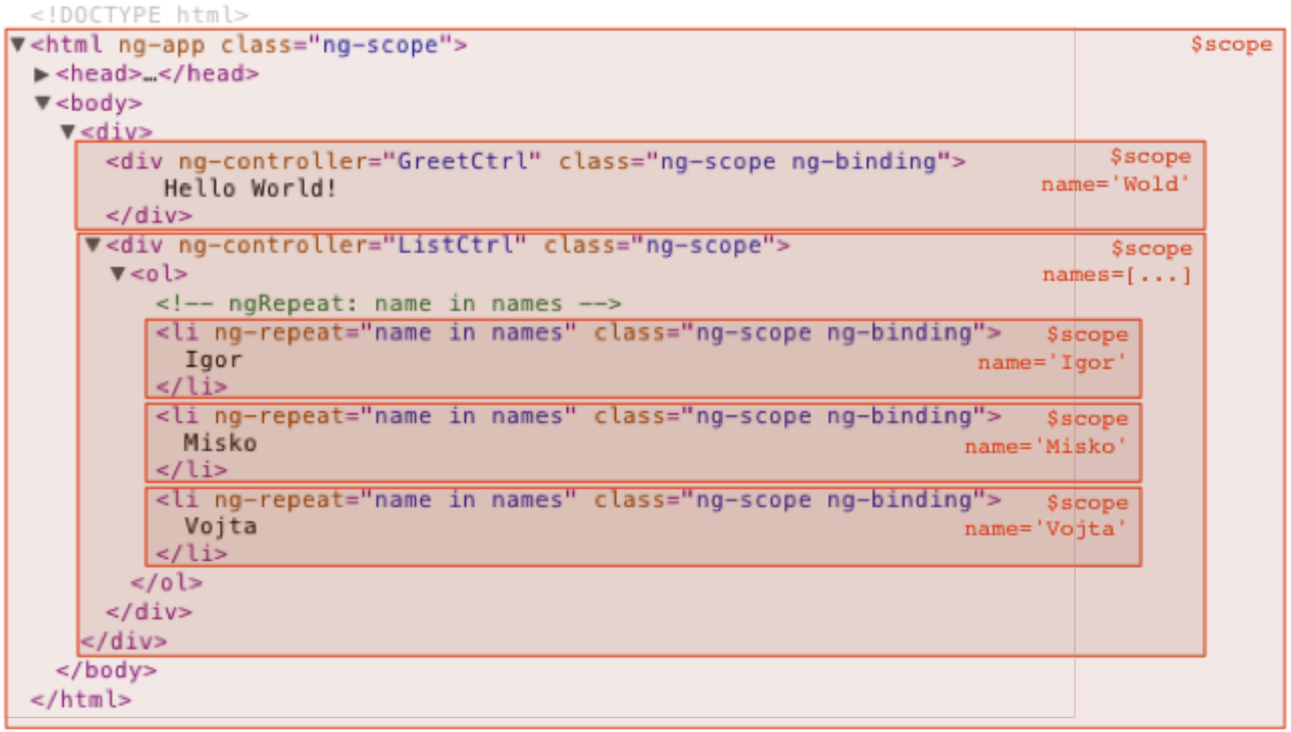
Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

Each ng-* will create a scope (subscope).

Accessible as $scope in code.

Also $rootScope

(single top level scope)



```
<!DOCTYPE html>
▼<html ng-app class="ng-scope">
 ▶<head>…</head>
 ▼<body>
  ▼<div>
     <div ng-controller="GreetCtrl" class="ng-scope ng-binding">
       Hello World!
     </div>
   ▼<div ng-controller="ListCtrl" class="ng-scope">
    ▼<ol>
       <!-- ngRepeat: name in names -->
       <li ng-repeat="name in names" class="ng-scope ng-binding">
         Igor
       </li>
       <li ng-repeat="name in names" class="ng-scope ng-binding">
         Misko
       </li>
       <li ng-repeat="name in names" class="ng-scope ng-binding">
         Vojta
       </li>
     </ol>
   </div>
  </div>
 </body>
</html>
```

Modules

Application code organized as modules

- Normally one module/js-file
- Modules have **names**
- List **dependencies** between modules
- Modules are objects, **references** to

```
// Create module (for many controllers) and get reference
// module object (file controllers.js). No dependencies
var personRegControllers = angular.module('PersonRegControllers',
[]);

// To retrieve module. NO dependency list
… = angular.module('PersonRegControllers');
```

BootStrapping

In template using ng-app

```
<html lang="en" ng-app="PersonReg">
```

- Will load module PersonReg as application “main” module
- Other modules as listed **dependencies**

```
var personReg = angular.module('PersonReg', [
     'ngRoute',
     'PersonRegControllers',  // Need these
     'PersonRegService'
]);
```

Module Functionality

Code to execute attached to modules

- Dependency injection i.e. angular supplies objects needed

```
// Add controller named PersonListCtrl to module personRegControllers
personRegControllers.controller('PersonListCtrl', ['$scope', 'PersonRegProxy',
    function($scope, PersonsRegProxy) {
          // code to execute …
    }]);


// Add configuration to module personReg
personReg.config(['$routeProvider',
    function($routeProvider) {
         // code to execute …
 }]);


// Add add factory named PersonRegProxy to module personRegService
personRegService.factory('PersonRegProxy', ['$http',
  function($http) {
        // code to execute …
 }]);
```

Controllers

Controllers (JS functions) will be part of C in MVC

- Attached to DOM using ng-controller directive or use $routeProvider

Use controllers to

- Set up the initial state of the $scope object
- Add behavior to the $scope object
- Add event listeners
- Host Observer Patterns code

Databinding and Listeners

```
// Set scoped variable values in Controller
personRegControllers.controller('PersonListCtrl', ['$scope',
                              'PersonRegProxy',
    function($scope, PersonsRegProxy) {
        $scope.orderProp = 'id';
        $scope.pageSize = '10';
        $scope.currentPage = 0;

// In template. pageSize and currentPage will be in sync! Automatically updated
<button class="btn btn-default" ng-disabled="currentPage >= count /  pageSize - 1"
          ng-click="currentPage =  currentPage + 1"> Next </button>


// Add a listener to scope (also in controller)
$scope.save = function() {
            PersonRegProxy.create($scope.person)
                    .success(function() {
                        $scope.message = "Success";
                    }).error(function() {
                ; // TODO;
            });
        };
// In template
<input type="button" value="Save" class="btn btn-default" ng-click=" save()"/>
```

Data binding works if data manipulated from inside Angular. If not have to force update using $apply(function(){ …})

Compile and Digest

Angular Compile

- Angular will traverse DOM, search for directives, attaching listeners if found
- Will set up "watches" (with callback functions) for expressions (example: keep an eye on {{person.id}} = $scope.person.id)

Angular Digest Cycle

- If any value in any scope changed …
- … Angular starts the digest cycle … check all watches.
- If value handled by the watch changed …
- … execute callback, manipulate DOM to render new value.

More on $scope

AngularJS Observer

Built in using $watch + databinding

```
// Observer
personRegControllers.controller('PersonListCtrl', ['$scope', … ,
    function($scope, PersonsRegProxy) {
       ...
       $scope.currentPage = 0;
       $scope.someValue = 45;
       ...
       $scope.$watch('currentPage', function() {
               getRange();
       });
       function getRange() {
            $scope.someValue = 99;
       }
    }]);
```

When currentPage changed getRange() called. In page “view of someValue” (DOM) updated due to data binding

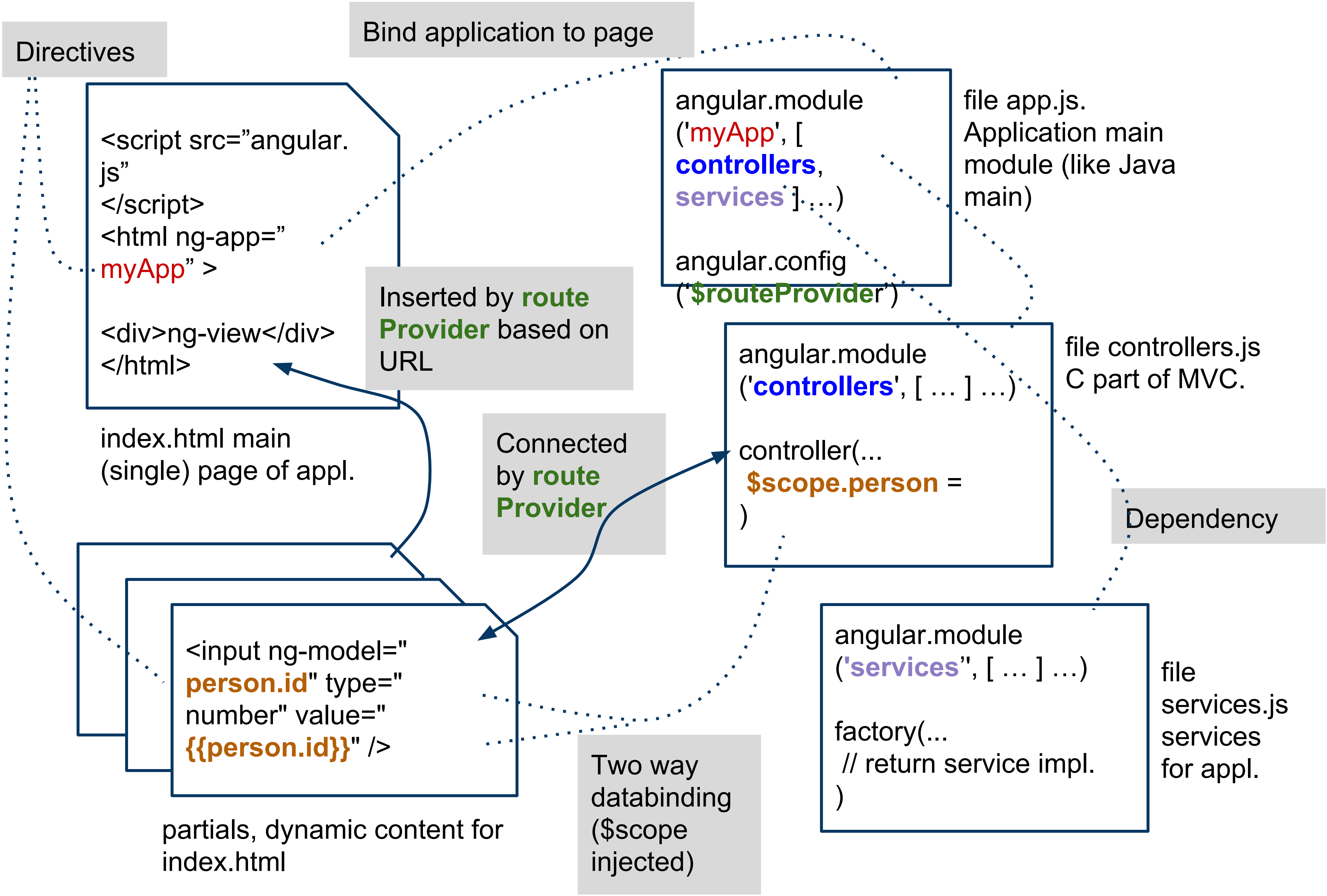
Factories

Used to create singleton instances of services

```
// Return service object named PersonRegProxy
personRegService.factory('PersonRegProxy', ['$http',
    function($http) {

        var url = 'http://localhost:8080/rest_backend/webresources/json';

        return {
            findAll: function() {
                return $http.get(url);
            },

            // …
            delete: function(id) {
                return $http.delete(url + "/" + id);
            },
            count: function() {
                return $http.get(url + "/count");
            }
        };
    }]);
```

Object returned (close to module pattern)

AngularJS App Overview



Directives

Bind application to page

```
<script src=”angular.
js”
</script>
<html ng-app=”
myApp” >

<div>ng-view</div>
</html>
```

index.html main (single) page of appl.

Inserted by **route Provider** based on URL

```
angular.module
('myApp', [
controllers,
services ] …)

angular.config
(‘$routeProvider’)
```

file app.js. Application main module (like Java main)

Connected by **route Provider**

```
angular.module
('controllers', [ … ] …)

controller(...
 $scope.person =
)
```

file controllers.js C part of MVC.

Dependency

```
<input ng-model="
person.id" type="
number" value="
{{person.id}}" />
```

partials, dynamic content for index.html

Two way databinding ($scope injected)

```
angular.module
('services”, [ … ] …)

factory(...
 // return service impl.
)
```

file services.js services for appl.

AngularJS Location

Used for navigation. The $location service exposes the current URL in the browser address bar, so you can

- Watch and observe the URL
- Change the URL

Synchronizes the URL with the browser when the user

- Changes the address bar
- Clicks the back or forward button (or clicks a History link)
- Clicks on a link

Represents the URL object as a set of methods (protocol, host, port, path, search, hash)

AJAX with AngularJS

```
// Create a service module
var someService = angular.module('someService', [] );
// Proxy more to come… Dependency on $http
someService.factory('someServiceProxy', ['$http',
   // $http object injected
    function($http) {
        var url = ...
        return {
            findAll: function() {
                return $http.get(url); // Call will return apromise
            }
           };
    }]);

// In control. Promise has methods success, error and more.
someServiceProxy.findAll()
    .success(function(){
       // Handle response
    }).error(function(){
       // Handle exception
    });
```

AngularJS cont.

Very good intro tutorial

Other good tutorial

Developers Guide

API (a bit messy)