

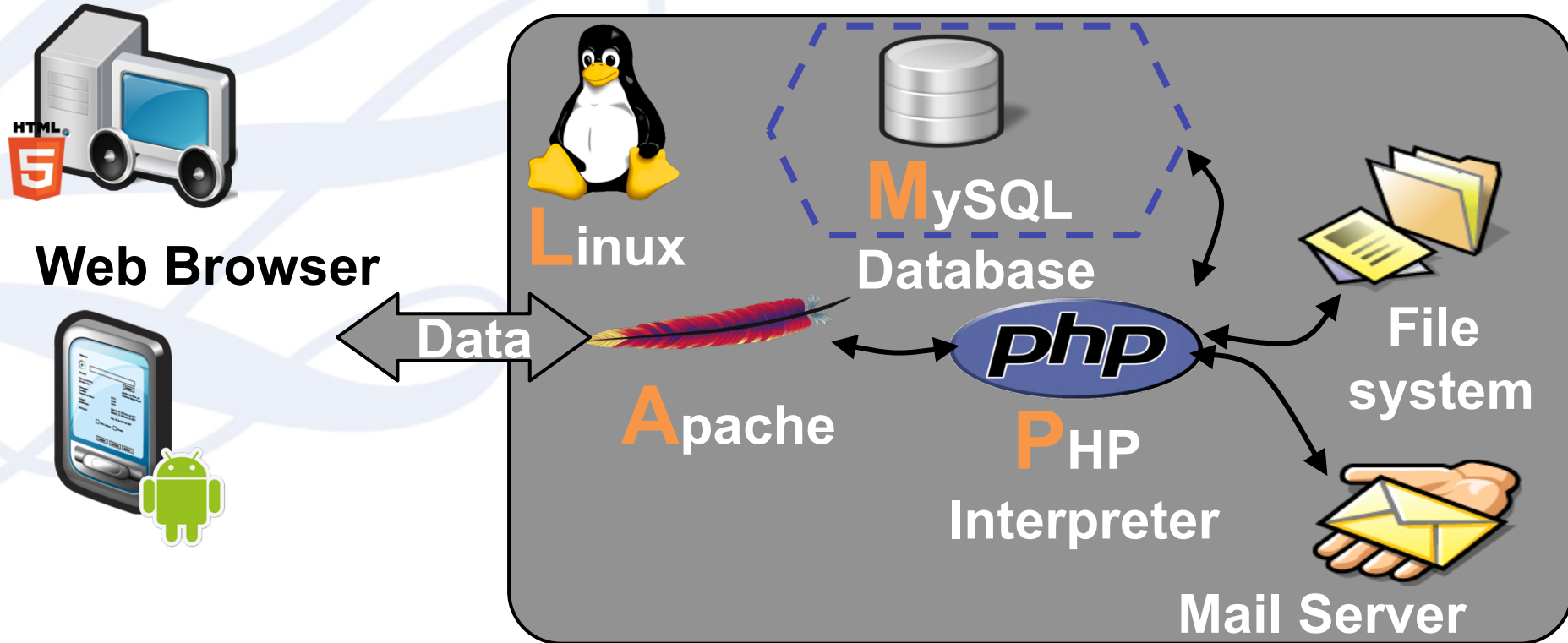
305CDE Developing the Modern Web 2

Tutorial 6

Introduction to AngularJS

Dr. Jianhua Yang, 04/11/2014

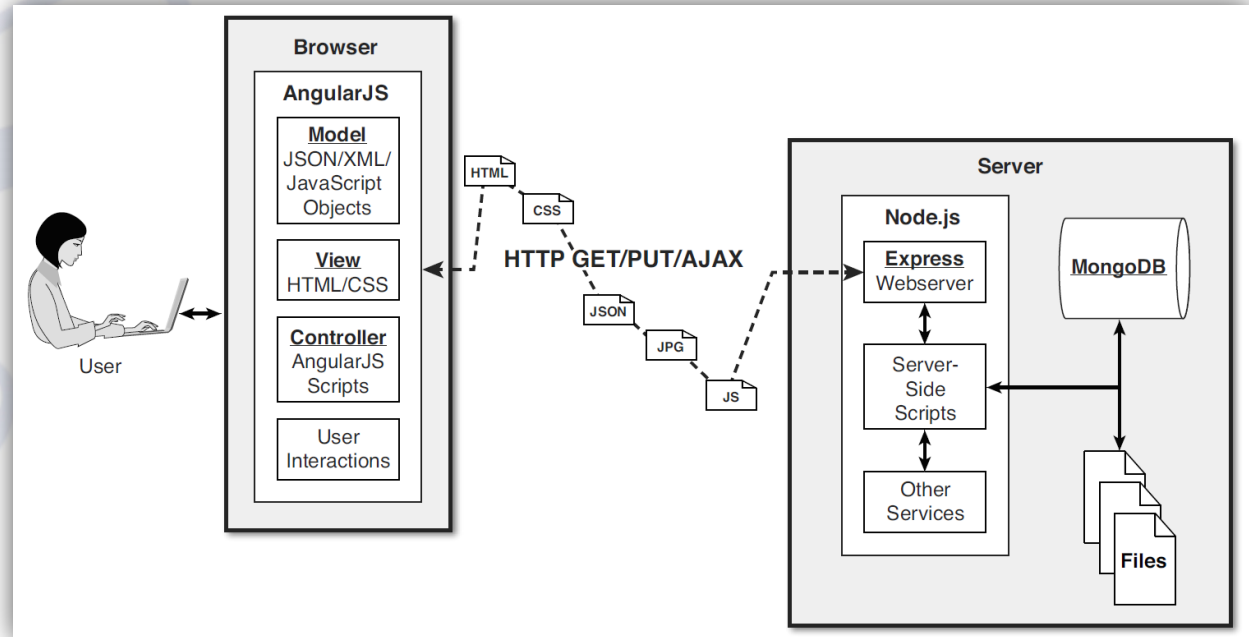
Backgrounds



Backgrounds

“Developers Love MEAN”

- MongoDB
- Express
- AngularJS
- Node.js

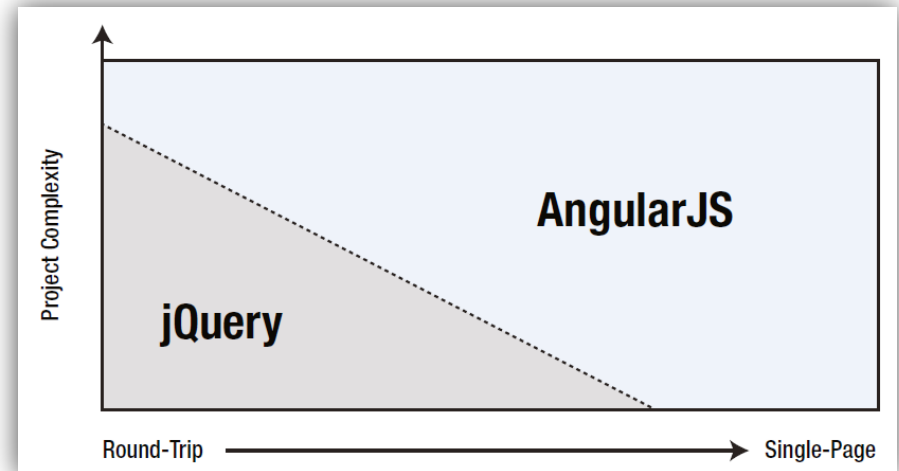


Backgrounds

“AngularJS enables you to move some or all of this processing and logic out to the browser, sometimes leaving the server just passing data from the database.

Something that AngularJS has been specifically designed for is **Single Page Application** functionality.”

- Simon D. Holmes



Roadmap (for next 6 weeks)

1. Introduction to AngularJS + API
2. Forms and services + API
3. Server communication + API
4. Filters and routing + API
5. More directives + API
6. Review + API



Coventry
University

How to master AngularJS?

 **stackoverflow**

QuestionsTagsUsersBadgesUnanswered

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

How to master AngularJS? [closed]

266
★
374

I'm pretty new to AngularJS and I find it a bit awkward. The easy stuff is very easy, but the advanced things are significantly harder (directives, provider / service / factory...)

The [documentation](#) isn't very helpful for someone who's just starting to learn those things; and I find myself constantly searching for directives for things I need instead of writing my own.

I tried [Ember.js](#) and I was much more productive with it, but the API is still being changed significantly so I prefer skipping it for now.

Are there any better resources to get into AngularJS properly?

angularjs

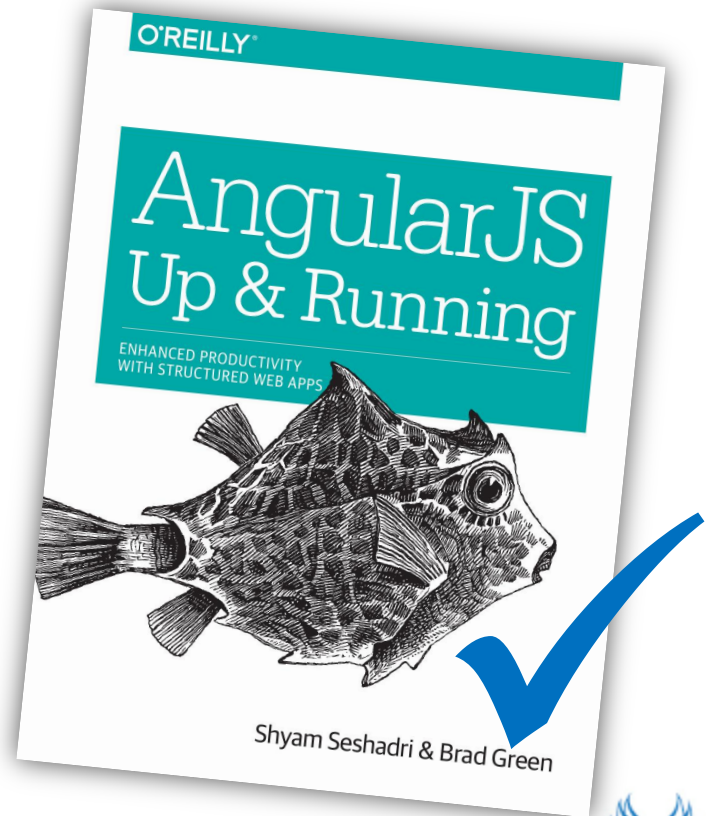
share

edited Sep 11 '13 at 11:25
 **Voles**
1,958 ● 3 ● 20 ● 43

asked Jan 15 '13 at 8:38
 **Gal Ben-Haim**
5,476 ● 7 ● 23 ● 59

The “textbook”

- This (➔) serves as our “textbook”
- It **overrides most** existing resources (online/offline), as AngularJS introduced some new features in release 1.2.
- The manual **overrides all!**



305CDE Developing the Modern Web 2

Lecture 6

Introduction to AngularJS

Dr. Jianhua Yang, 05/11/2014

Jianhua.Yang@coventry.ac.uk

COM Programming Support: ECG-03, Tues. 13:00-15:00



Contents

- Two-way data binding
- Directives
- Modules
- Controllers
- Model View Controller (MVC)
- \$http service

Two-way data binding

The image illustrates two-way data binding in AngularJS. It features three overlapping windows:

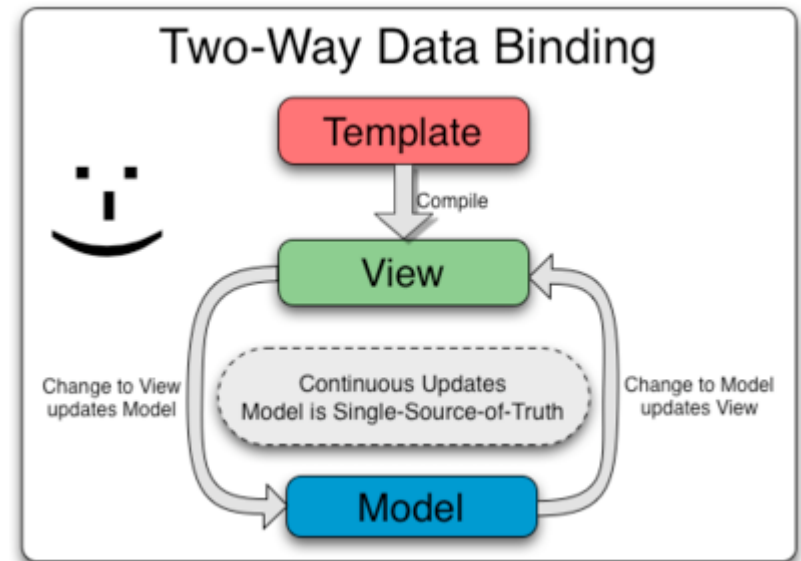
- Brackets Editor:** Displays the source code for `HelloAngularJS.html`. The code includes an AngularJS application with a text input and a greeting message that updates based on the input value.
- Browser Window (Left):** Shows the initial state where the input field is empty and the greeting is "Hello, !".
- Browser Window (Right):** Shows the state after the user has entered "AngularJS" into the input field, resulting in the updated greeting "Hello, AngularJS!".

Source Code (HelloAngularJS.html):

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <div ng-app="">
5       <p>
6         Name:
7         <input type="text" ng-model="name" value="John">
8       </p>
9       <p>
10        Hello, {{name}}!
11      </p>
12    </div>
13    <script
14      src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.
15      n.js"></script>
16  </html>
```

Two-way data binding

- Data binding works in the direction from our model to the UI and from the UI to the model.
- Any view changes triggered by a user are immediately reflected in the model, and any changes in the model are instantly propagated to a template.



Directives

- AngularJS directives are extended HTML attributes with the prefix **ng-**.
- The **ng-app** directive initializes an AngularJS application.
- The **ng-init** directive initialize application data.
- The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

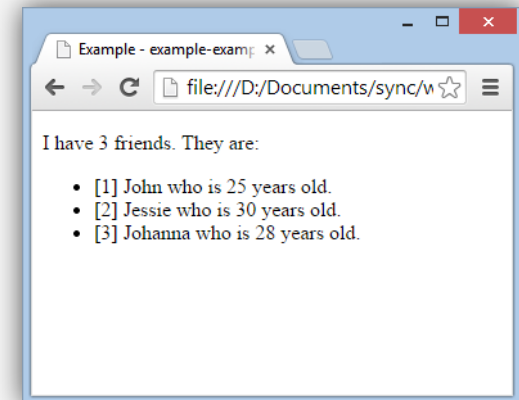
```
1. <div ng-app="" ng-init="firstName='John'">
2.   <p>
3.     Name:
4.     <input type="text" ng-model="firstName">
5.   </p>
6.   <p>
7.     You wrote: {{ firstName }}
8.   </p>
9. </div>
```

“We typically refer to directives by their case-sensitive camelCase normalized name (e.g. ngModel). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using dash-delimited attributes on DOM elements (e.g. ng-model).”

– AngularJS Developer Guide

Directives

```
1. <!doctype html>
2. <html lang="en">
3.   <head>
4.     <title>Example - example-example42-production</title>
5.     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-beta.19/angular.min.js"></script>
6.   </head>
7.   <body>
8.     <div ng-app="" ng-init="friends = [
9.       {name:'John', age:25, gender:'boy'},
10.      {name:'Jessie', age:30, gender:'girl'},
11.      {name:'Johanna', age:28, gender:'girl'}
12.    ]">
13.      <p>
14.        I have {{friends.length}} friends. They are:
15.      </p>
16.      <ul>
17.        <li ng-repeat="friend in friends">
18.          [{{$index + 1}}] {{friend.name}} who is {{friend.age}} years old.
19.        </li>
20.      </ul>
21.    </div>
22.  </body>
23. </html>
```



Directives

Problem	Solution
Create a one-way binding.	Define properties on the controller <code>\$scope</code> and use the <code>ng-bind</code> or <code>ng-bind-template</code> directive or inline expressions (denoted by the <code>{{</code> and <code>}}</code> characters).
Prevent AngularJS from processing inline binding expressions.	Use the <code>ng-non-bindable</code> directive.
Create two-way data bindings.	Use the <code>ng-model</code> directive.
Generate repeated elements.	Use the <code>ng-repeat</code> directive.
Get context information about the current object in an <code>ng-repeat</code> directive.	Use the built-in variables provided by the <code>ng-repeat</code> directive, such as <code>\$first</code> or <code>\$last</code> .
Repeat multiple top-level attributes.	Use the <code>ng-repeat-start</code> and <code>ng-repeat-end</code> directives.
Load a partial view.	Use the <code>ng-include</code> directive.
Conditionally display elements.	Use the <code>ng-switch</code> directive.
Hide inline template expressions while AngularJS is processing content.	Use the <code>ng-cloak</code> directive.

Modules

Module as a container for the different parts of your app – controllers, services, filters, directives, etc.

```
1. <!doctype html>
2. <html lang="en">
3.   <head>
4.     <meta charset="UTF-8">
5.     <title>Example - example-example105-production</title>
6.     <script src="//ajax.googleapis.com/ajax/libs/
angularjs/1.3.0-rc.0/angular.min.js"></script>
7.     <script src="script.js"></script>
8.   </head>
9.   <body >
10.    <div ng-app="myApp">
11.      <div>
12.        {{ 'World' | greet }}
13.      </div>
14.    </div>
15.  </body>
16.</html>
```

```
1. // declare a module
2. var myAppModule = angular.module('myApp', []);
3. // configure the module.
4. // in this example we will create a greeting filter
5. myAppModule.filter('greet', function() {
6.   return function(name) {
7.     return 'Hello, ' + name + '!';
8.   };
9. });
```

Modules

- Keeping our global namespace clean.
- Making tests easier to write and keeping them clean so as to more easily target isolated functionality.
- Making it easy to share code between applications.
- Allowing our app to load different parts of the code in any order.

Name	Description
animation(name, factory)	Supports the animation feature, which I describe in Chapter 23.
config(callback)	Registers a function that can be used to configure a module when it is loaded. See Chapter 9.
constant(key, value)	Defines a service that returns a constant value. See Chapter 9.
controller(name, constructor)	Creates a controller. See Chapter 13 for details.
directive(name, factory)	Creates a directive. See Chapters 15–17 for details.
factory(name, provider)	Creates a service. See the “Using the Factory Method” section later in this chapter for details.
filter(name, factory)	Creates a filter that formats data for display to the user. See Chapter 14 for details.
provider(name, type)	Creates a service, as described in the “Using the Provider Method” section of this chapter.
name	Returns the name of the module.
run(callback)	Registers a function that is invoked after AngularJS has loaded and configured all of the modules. See Chapter 9.
service(name, constructor)	Creates a service, as described in the “Using the Service Method” section of this chapter.
value(name, value)	Defines a service that returns a constant value. See Chapter 9.

Controllers

1. Set up the initial state of the `$scope` object.

```
1. function MyCtrl($scope) {  
2.   $scope.value = "some value";  
3. }
```

```
1. <html>  
2.   <head>  
3.     <script src="js/angular.js"></script>  
4.     <script src="js/app.js"></script>  
5.     <link rel="stylesheet" href="css/bootstrap.css">  
6.   </head>  
7.   <body ng-app>  
8.     <div ng-controller="MyCtrl">  
9.       <p>  
10.        {{value}}  
11.      </p>  
12.    </div>  
13.  </body>  
14.</html>
```

Controllers

2. Add behaviour to the `$scope` object:

We add behaviour to the scope by attaching methods to the `$scope` object. These methods are then available to be called from the template/view.

```
1. <div ng-controller="DoubleController">
2.   Two times
3.   <input ng-model="num">
4.   equals {{ double(num) }}
5. </div>
```

```
1. var myApp = angular.module('myApp', []);
2.
3. myApp.controller('DoubleController', ['$scope',
4. function($scope) {
5.   $scope.double = function(value) {
6.     return value * 2;
7.   };
8. }]);
```

Controllers

3. Watch other parts of the model for changes and take action.

```
1. <html>
2.   <head>
3.     <script src="js/angular.js"></script>
4.     <script src="js/app.js"></script>
5.     <link rel="stylesheet" href="css/bootstrap.css">
6.   </head>
7.   <body ng-app>
8.     <div ng-controller="MyCtrl">
9.       <input type="text" ng-model="name"
placeholder="Enter your name">
10.     <p>
11.       {{greeting}}
12.     </p>
13.   </div>
14. </body>
15.</html>
```

```
1. function MyCtrl($scope) {
2.   $scope.name = "";
3.   $scope.$watch("name", function(newValue, oldValue) {
4.     if (newValue.length > 0) {
5.       $scope.greeting = "Greetings " + newValue;
6.     }
7.   });
8. }
```

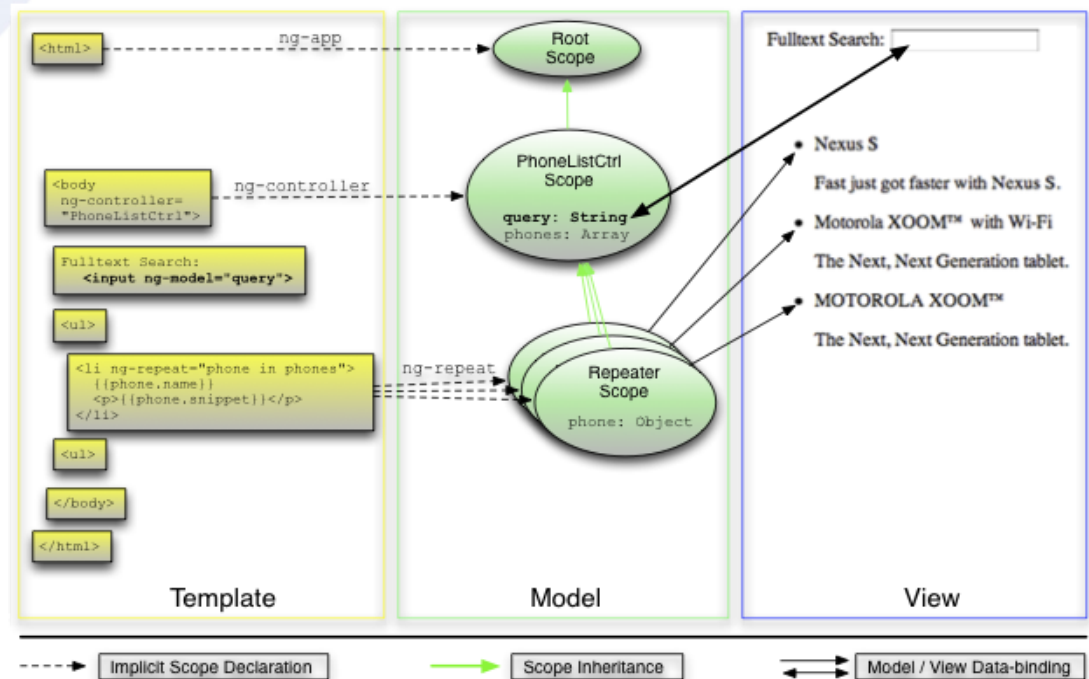
Controllers

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. Angular has **databinding** for most cases and **directives** to encapsulate manual DOM manipulation.
- Format input — Use **angular form** controls instead.
- Filter output — Use **angular filters** instead.
- Share code or state across controllers — Use **angular services** instead.
- Manage the life-cycle of other components (for example, to create service instances).

Model View Controller (MVC)

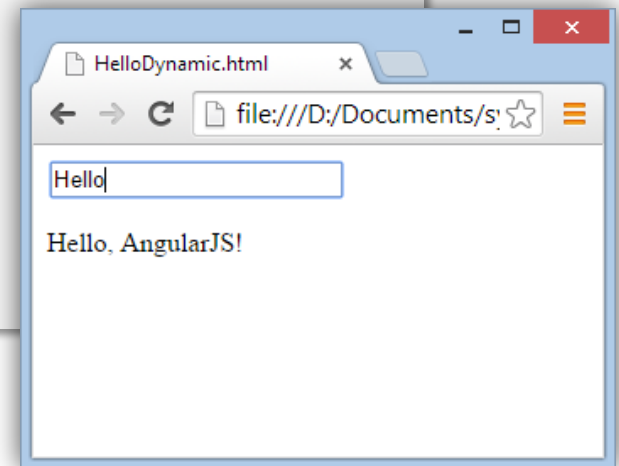
- Models keep track of your app's data.
- Views display your user interface and make up the content of an app.
- Controllers manage your views.



Model View Controller (MVC)

```
1. <html ng-app>
2.   <head>
3.     <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.0.8/angular.min.js"></script>
4.     <script src="controllers.js"></script>
5.   </head>
6.   <body>
7.     <div ng-controller='HelloController'>
8.       <input ng-model='greeting.text'>
9.       <p>
10.        {{greeting.text}}, AngularJS!
11.      </p>
12.    </div>
13.  </body>
14. </html>
```

```
1. function HelloController($scope) {
2.   $scope.greeting = {
3.     text : 'Hello'
4.   };
5. }
```



Template

Model/Controller

View

XMLHttpRequest

1. Create the desired event handler functions.
2. Create a new instance of the `XMLHttpRequest` object and configure it for use (including providing the event handlers).
3. Instruct the `XMLHttpRequest` object to make the request. As it does its work, it will dispatch events at the appropriate times.

```
1. var xmlhttp = new XMLHttpRequest();
2. xmlhttp.onreadystatechange = function() {
3.   if (xmlhttp.readyState == 4 &&
        xmlhttp.status == 200) {
4.     var response = xmlhttp.responseText;
5.   } else if (xmlhttp.status == 400) {
6.     // or really anything in the 4 series
7.     // Handle error gracefully
8.   }
9. };
10. // Setup connection
11. xmlhttp.open("GET", "http://myserver/api",
        true);
12. // Make the request
13. xmlhttp.send();
```

\$http service

- The `$http` service is simply a wrapper around the browser's raw `XMLHttpRequest` object.
- The `$http` service is a function that takes a single argument: a configuration object that is used to generate a HTTP request. The function returns a promise that has two helper methods: success and error.
- The `$http` API is based on the deferred/promise APIs exposed by the `$q` service.

\$http service

```
1. <body ng-controller="ShoppingController">
2.   <h1>Shop!</h1>
3.   <table>
4.     <tr ng-repeat="item in items">
5.       <td>{{item.title}}</td>
6.       <td>{{item.description}}</td>
7.       <td>{{item.price | currency}}</td>
8.     </tr>
9.   </table>
10. </div>
11.</body>
```

```
1. function ShoppingController($scope, $http) {
2.   $http.get('/products').success(function(data,
3.     status, headers, config) {
4.     $scope.items = data;
5.   });
6. }
```

```
1. [
2.   {
3.     "id": 0,
4.     "title": "Paint pots",
5.     "description": "Pots full of paint",
6.     "price": 3.95
7.   }, {
8.     "id": 1,
9.     "title": "Polka dots",
10.    "description": "Dots with that polka groove",
11.    "price": 12.95
12.  }, {
13.    "id": 2,
14.    "title": "Pebbles",
15.    "description": "Just little rocks, really",
16.    "price": 6.95
17.  }
18.]
```

\$http service

General usage

```
1. $http({
2.   method : 'GET',
3.   url : '/someUrl'
4. }).success(function(data, status, headers,
5.   config) {
6.     // this callback will be called
7.     // asynchronously
8.     // when the response is available
9.   }).error(function(data, status, headers,
10.   config) {
11.     // called asynchronously if an error occurs
12.     // or server returns response with an error
13.     status.
14.   });
```

Shortcut methods (Lecture8)

```
1. $http.get('/
   someUrl').success( successCallba
   ck);
2. $http.post('/someUrl',
   data).success(successCallback);
```

[\\$http.get](#)

[\\$http.post](#)

[\\$http.delete](#)

[\\$http.patch](#)

[\\$http.head](#)

[\\$http.put](#)

[\\$http.jsonp](#)