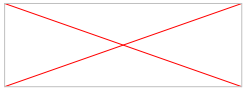
AngularJS: Lesson 4

Custom Services



ANGULARJS by Google

Services

Features that you want to work across components & controllers

What goes in them?

- Business logic
- Data persistence
- Server communication
- Shared state
- Caching
- Factories
- Third-party JS libraries (wrapped in order to be injectable and testable)

Services

Other interesting features

- Not dependent on views
- Each is created as a singleton - one instance is shared across the module

Services

5 methods to create your own service

- Value
- Constant
- Factory
- Service
- Provider

Services: Value

When all you need is to store a value (any type) and retrieve it at run time

```
// Defining the value
angular.module('angularjsTutorial')
     .value('googleApiKey', '12345');

// Retrieve the value
angular.module('angularjsTutorial')
     .controller('MainCtrl', function($scope, googleApiKey){
          console.log(googleApiKey); // '12345'
     });
```

Services: Constant

Just like Value, but also available at config time

```
// Defining the value
angular.module('angularjsTutorial')
     .constant('googleApiKey', '12345');

// Retrieve the value at module config time
angular.module('angularjsTutorial').config(function(googleApiKey) {

});
```

Services: Factory

Simplest way to create more advanced services

Service API is the return object
Equivalent to Service, just a different style

```
angular.module('angularjsTutorial').factory('MyMessageService', function(){
     var privateMessage = 'hello';
     return {
          getMessage : function(){
               return privateMessage;
          },
          setMessage : function(value){
               privateMessage = value;
          }
     }
});
```

Services: Service

Equivalent to Factory, just a different style

Uses a constructor
Service API is the “this” from the constructor

```
angular.module('angularjsTutorial').service('MyMessageService', function MyMessageService(){
     var privateMessage = 'hello';

     this.getMessage = function(){
          return privateMessage;
     };
     this.setMessage = function(value){
          privateMessage = value;
     };
});
```

Services: Provider

The grandaddy.

More complex, but lets you customize the service at configuration time.
The rest of the Angular services use this underneath.

```
angular.module('angularjsTutorial').provider('MyMessageService',function MyMessageService(){
     this.name = 'Default';
     this.setName = function(name) {
          this.name = name;
     };

     this.$get = function() {
          var name = this.name;
          return {
               sayHello: function() {
                    return "Hello, " + name + "!";
               }
          }
     };
});

myApp.config(function(MyMessageServiceProvider){
    MyMessageServiceProvider.setName('World');
});
```

Questions?

To Form or not to Form

nested forms use directive: ng-form

Forms get you:

- one place to attach your submit handler
- submit on “enter” keypress

Syntax

```
<div ng-form name=”myForm” ng-submit=”onMySubmitAction() novalidate>
     <input name=”myInput” type=”text” ng-model=”myModelValue>
     <input type=”submit”>Submit
</div>


<form name=”ctrl.myForm” novalidate>
     <input name=”myInput” type=”text” ng-model=”ctrl.myModelValue>
     <input name=”subInput” type=”submit” value=”submit”>
</form>
```

Built-In Validators

```
<form name=”myForm” novalidate>
     <input name=”textInput” type=”text” required ng-model=”someText”>
     <input name=”myInput” type=”number” min=”0” max=”99” ng-model=”myModelValue”>
     <input name=”emailInput” type=”email” ng-model=”userEmail”>
     <input name=”subInput” type=”submit” value=”submit”>
</form>
```

Others: https://docs.angularjs.org/api/ng/input