specialist ICT learning

GURUTEAM

H&R BLOCK

AngularJS: Services and DI

GuruTeam Instructor: Sang Shin

Topics

- What is and why service?
- Built-in (Angular provided) services
  - $log, $location, $timeout, $interval
- Dependency injection (DI)
- Creating a custom service

What is and Why Service?

What is and Why Service?

- Angular services are reusable objects
  - You can use services to organize and share code across your app
- A service can be injected into controller, another service, filter, directive that depends on that service
  - Angular's dependency injection subsystem ($injector) handles dependency injection
- Angular services are:
  - Lazily instantiated – Angular only instantiates a service when an application component depends on it
  - Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory

Angular has many built-in services

- Angular offers several useful services (like $http), but for most applications you'll also want to create your own
  - Like other core Angular identifiers, built-in services always start with $ (e.g. $http)

- https://docs.angularjs.org/api/ng/service
  - $animate, $controller, $document, $exceptionHandler
  - $filter, $http, $interval, $locale, $location
  - $log, $parse, $q, $rootElement, $rootScope
  - $timeout, $window

Built-in Services:
$log service,
$location service,
$Timeout service,
$Interval service

$log service

- Simple service for logging
- Default implementation writes the message into the browser's console

```
angular.module('logExample', [])
.controller('LogController', ['$scope', '$log', function($scope, $log) {
  $scope.$log = $log;
  $scope.message = 'Hello World!';
}]);
```

$location service (getter & setter)

- Is used to both get location data and set location data
  - Getter: The $location service parses the URL in the browser address bar (based on the window.location) and makes the URL available to your application
  - Setter: Changes to the URL in the address bar are reflected into $location service and changes to $location are reflected into the browser address bar

```
myApp.controller('MyController', ['$scope', '$location', function ($scope, $location) {
   // given url http://example.com/#/some/path?foo=bar&baz=xoxo
   $scope.absUrl = $location.absUrl();  // getter example
   // => "http://example.com/#/some/path?foo=bar&baz=xoxo"

   $scope.setpath = function (path) {
       $location.path(path);                    // setter example
   }
}]);
```

$timeout service

- $timeout service is Angular's wrapper for window.setTimeout

```
myApp.controller('MyController', ['$scope', '$timeout', function ($scope, $timeout) {
        $scope.timeout = function (duration) {
            $scope.duration = duration;
            $timeout(callTimeout, duration);
        }

        function callTimeout() {
            $scope.message = "Timeout occurred in " + $scope.duration / 1000 + "
seconds";
        }
}]);
```

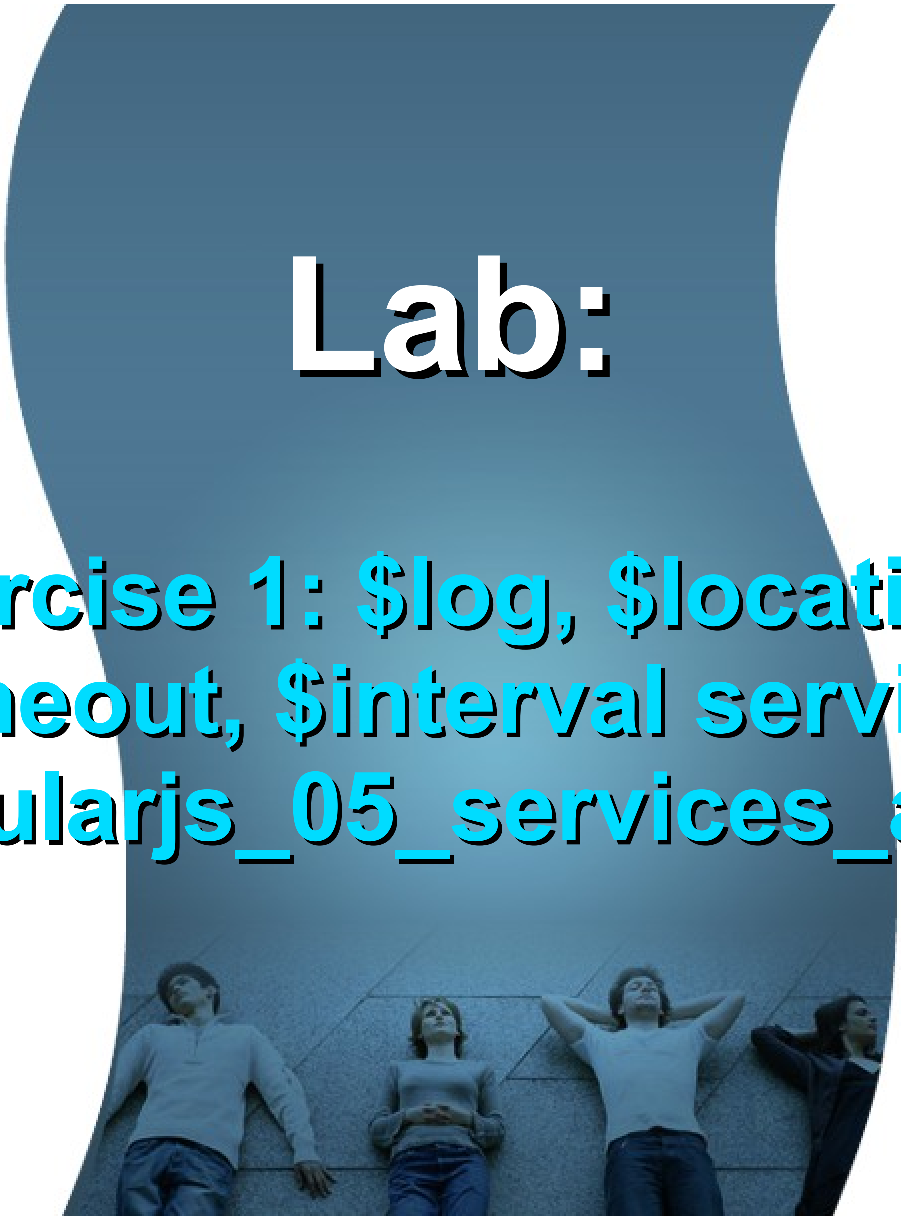
$interval service

- Used to rigger any functions scheduled to run in that time.

```
myApp.controller('MyController', ['$scope', '$interval', function ($scope, $interval) {
        …

        $scope.callAtInterval = function (duration) {
            $interval(callTimeout, duration);
            $scope.duration = duration;
        }

        function callTimeout() {
            $scope.totalDurationInSeconds += $scope.duration / 1000;
            $scope.messages.push("Timeout occurred in " +
$scope.totalDurationInSeconds + " seconds");
        }
}]);
```



Lab:

Exercise 1: $log, $location, $timeout, $interval services

3305_angularjs_05_services_and_di.zip

Dependency Injection

What is and Why DI?

- Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies
  - The component can have the dependency passed to it where it is needed - it removes the responsibility of locating the dependency from the component. The dependency is simply handed to the component
  - Simpler to code and test
- Dependency injection helps to make your web applications both well-structured (e.g. separate entities for presentation, data, and control) and loosely coupled
  - Dependencies between entities are not resolved by the entities themselves, but by the DI subsystem

Injection Syntax

- Implicit annotation
  - > The simplest way to get hold of the dependencies is to assume that the function parameter names are the names of the dependencies
  - > Downside of this approach - If you plan to minify your code, your service names will get renamed and break your app

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

- Inline array annotation (preferred)
  - > In order to avoid the minification problem of implicit annotation scheme, you can specify the dependencies explicitly

```
someModule.controller('MyController', ['$scope', 'greeter', function($scope, greeter) {
  // ...
}]);
```

Using Strict Dependency Injection

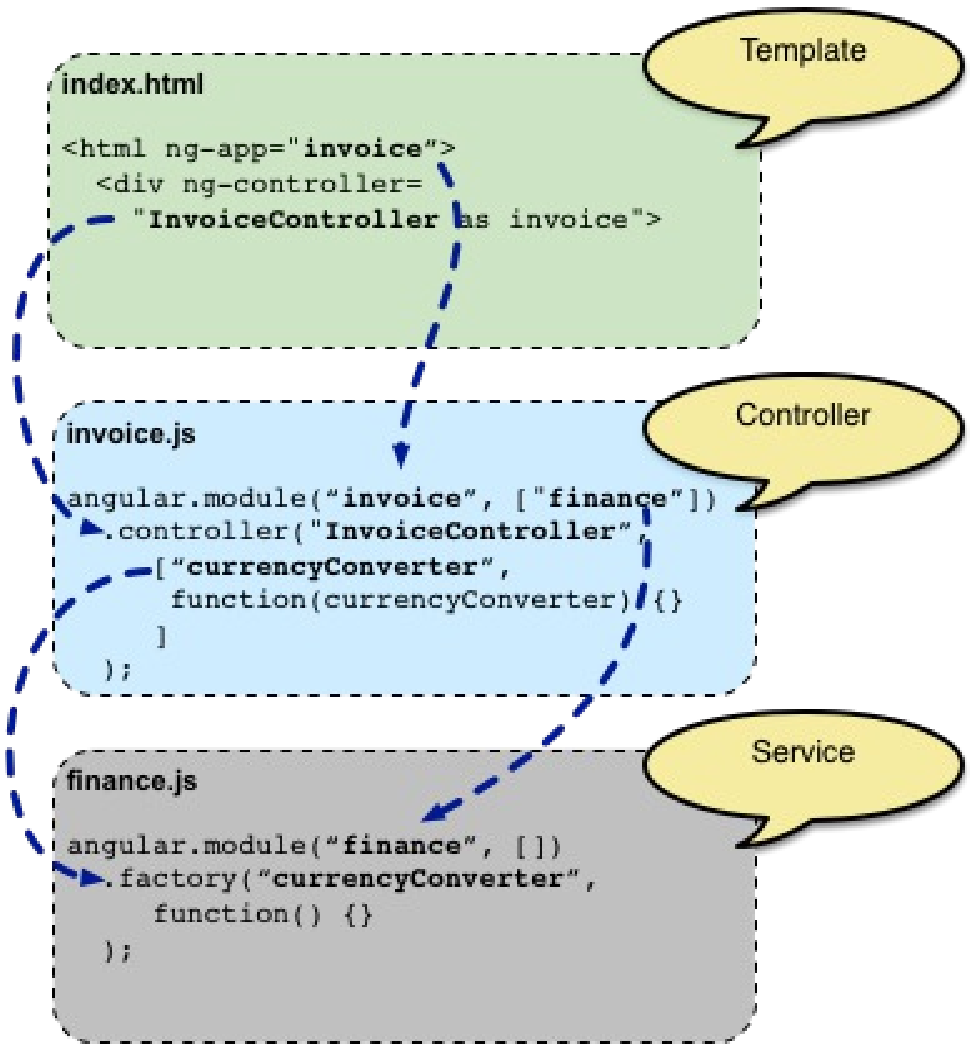
- Add an *ng-strict-di* directive on the same element as ng-app to opt into strict DI mode - Strict mode throws an error whenever a service tries to use implicit annotations

```
<body ng-app="MyApp" ng-strict-di>
    <div ng-controller="MyController">
      <p>absUrl: {{absUrl}} </p>
      <p>protocol: {{protocol}}</p>
    </div>

    <script>
    var myApp = angular.module('MyApp', []);

     myApp.controller('MyController', ['$scope', '$location', function ($scope, $location) {
        $scope.absUrl = $location.absUrl();
        $scope.protocol = $location.protocol();
     }]);
    </script>
</body>
```

Service Injection Example



```
<html ng-app="invoice">
  <div ng-controller=
    "InvoiceController as invoice">
```

```
angular.module("invoice", ["finance"])
 .controller("InvoiceController",
   ["currencyConverter",
     function(currencyConverter) {}
   ]
 );
```

```
angular.module("finance", [])
 .factory("currencyConverter",
   function() {}
 );
```

"currencyConverter" custom service is injected into "InvoiceController"

"currencyConverter" custom service is created

Lab:

Exercise 2: Dependency Injection
3305_angularjs_05_services_and_di.zip

Creating a Custom Service

Why Create a Custom Service?

- If you have business logic that could be used in multiple places, you should create a service
  - > Increases encapsulation
  - > Increases reusability
- If you need to manipulate data, you should create a service
  - > Controller should not directly manipulate data

How to Create a Custom Service

- By registering the service's name and service factory function by using *module.factory(..)* method
- The service factory function generates either a JavaScript object or a function object that represents the service to the rest of the application
  - In other words, a service can be a JavaScript object or a function object (well.. technically a function object is also a JavaScript object)

```
myModule.factory('myService1', function() {
    // return JavaScript object or function object
});
```

- The service can then be injected into any component (controller, service, filter or directive) that specifies a dependency on the service

Register Service Name & Factory

- Note that you are not registering a service instance, but rather a factory function that will create the instance when called - this enables lazy instantiation of service object - if it is not needed, it will not be created

```
var myModule = angular.module('myModule', []);

// The returned service object is JavaScript object
myModule.factory('myService1', function() {
    var shinyNewServiceInstance;
    // factory function body that constructs shinyNewServiceInstance
    return shinyNewServiceInstance;
});

// The returned service object is a function object
myModule.factory('myService1', function() {
    return function () {
    };
});
```

Usage of a Service

- A service is then injected into Controller, another Service, etc and then used

```
// Define “MyCompute” service
app.factory("MyCompute", function () {
    return {
       add: function (x, y) {
                  return x + y;
               }
    }
});

// "MyCompute" service is injected into "MyController"
app.controller("MyController", function ($scope, MyCompute) {
    $scope.addResult = MyCompute.add(50, 30);
});
```

Lab:

Exercise 3: Create custom services
3305_angularjs_05_services_and_di.zip

Who are GuruTeam?

- Specialist onsite training in Linux, Cloud, Database, Architecture, Software and Web Development Technologies
- Accredited by the LPI, CompTIA, Hortonworks and the Cloud Credential Council to deliver training, examinations and certifications.
- Over 230 courses available
- All GuruTeam instructors have extensive real-world experience in their technologies
- Clients are indigenous Irish Companies and Multinationals
- We can bring high spec preconfigured equipment for deliveries in Ireland, the UK and Europe.

Linux Professional Institute Approved Training Partner

CompTIA AUTHORIZED PARTNER

Hortonworks Partner TRAINING DELIVERY

CLOUD CREDENTIAL COUNCIL ACCREDITED PARTNER

Exceed your expectations...
GURUTEAM