CitiusTech


# Services

May 2015

# Agenda

- **Services Introduction**
- Different Ways of registering Service
  - Using Value function
  - Using Service function
  - Using Factory function
  - Using  Provider function
- Using a Service
  - Implicit DI
  - Explicit DI
- AngularJS Internal Services
  - $timeout,$interval
  - $watch, $digest, $apply

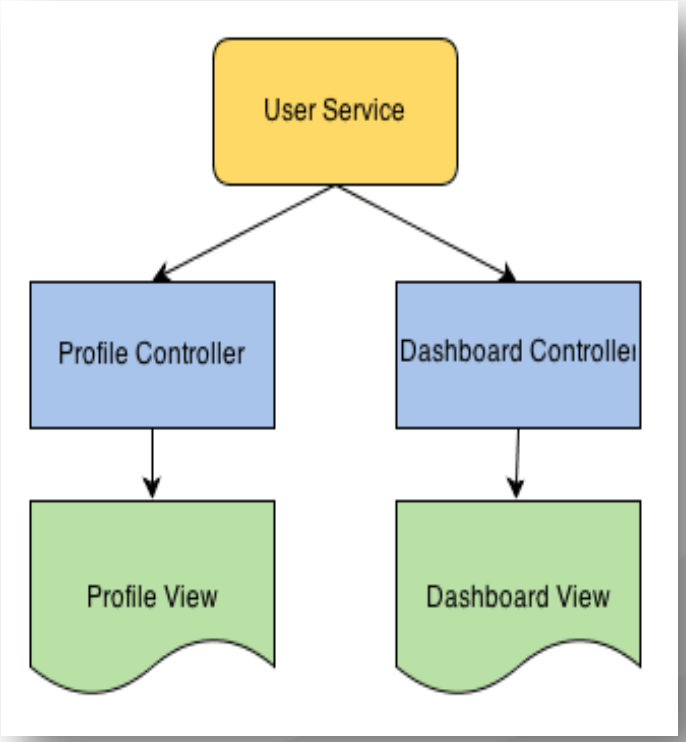
# Services Introduction

- What is Service in AngularJS?
- AngularJS services are singleton components. Only one instance is created by the framework and then supplied for every DI request.
- Services are Lazily instantiated - Angular only instantiates a service when an application component depends on it.
- Example

**Uses :**

Service uses are typically:

- Persist and share data between components
- Provide an interface for loading and accessing that data
- Containers for reusable chunks of business logic and functionality




Source:
http://viralpatel.net/blogs/angularjs-service-factory-tutorial/

# Agenda

- Services Introduction
- **Different Ways of Registering Service**
  - **Using Value function**
  - **Using Service function**
  - **Using Factory function**
  - **Using  Provider function**
- Using a Service
  - Implicit DI
  - Explicit DI
- AngularJS Internal Services
  - $timeout,$interval
  - $watch, $digest, $apply

# Different Ways of Registering Service (1/6)

**Using Value Function**

- Register a ‘UserType’ service using value function.

```
var app = angular.module('myModule', []);
app.value(‘UserType', 2);
```

- To use it, inject ‘UserType’ into a controller as follows:

```
app.controller(‘TestCntrl’, function($scope,UserType){ //some code });
```

- Register a ‘User’ service using value function:

```
app.value(‘User’, { staffid:101,
                    firstname:’namrata’,
                     lastname:’marathe’,
                     locale: ‘de:CH’ });
```

- Injecting ‘User’ into a controller:

```
app.controller(‘LoginCntrl’, function($scope, User){ //some code });
```

# Different Ways of Registering Service (2/6)

**Using Service Function**

- module.service(name,fn) - This is simplest pattern
- It allows registration of a service via a constructor function:

```
myApp.service('helloWorldFromService', function() {
        this.sayHello = function() {
           return "Hello, World!"
        };      });
```

- To use it, inject it into a controller as follows:

```
app.controller('TestCntrl', function($scope, helloWorldFromService)
                                        {  //some code  });
```

- Basic demo on angular service:
  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day3/day3_Demos/1_service_vs_factory.html

# Different Ways of Registering Service (3/6)

**Using Factory Function**

- module.factory(name,fn) - This is a slightly more flexible pattern.
- Register an arbitrary object as a service. Object creation logic can be more complex and private fields can be simulated.

```
myApp.factory('helloWorldFromFactory', function() {
   var privateVal = "baz";
     return {    sayHello: function() {
                                  return "Hello, World!"
                               }
            };     });
```

- Demo on factory, provider:
  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day3/day3_Demos/2_service_factory_provider.html

# Different Ways of Registering Service (4/6)

**Using Factory Function**

- To use it, inject it into a controller as follows:

```
app.controller('TestCntrl', function($scope, helloWorldFromFactory)
                                    //some code })
```

- Creating a service using factory function is more flexible  than creating it using a service function as:
  - It allows to register and return an arbitrary object as a service.
  - Object creation logic can be more complex and customized as needed
  - Private fields can be simulated (as shown above).

# Different Ways of Registering Service (5/6)

**Using Provider Function**

- module.provider(name,fn) - The most complex pattern.
- Allows an arbitrary object to be registered just like the factory pattern.
- Allows  object to be configured during the configuration phase before it's used for DI.
- Usually overkill for most services, but most useful when a service needs to be re-used across applications with configurable changes to behaviour.

- Demo on provider:
  http://ctgit/silviap/angularjsdemosandlabs/blob/master/Day3/day3_Demos/2_service_factory_provider.html

# Different Ways of Registering Service (6/6)

## Using Provider Function

```
//Register a provider service
myApp.provider('helloWorld', function()     {
this.name = 'Default';
      this.$get = function() {
          var name = this.name;
          return {
            sayHello: function() {
               return "Hello, " + name + "!" }
               }       }; //end of $get
      this.setName = function(name) {
               this.name = name;       };     });
 //Configure a provider!
myApp.config(function(helloWorldProvider)
{  helloWorldProvider.setName('Angular');
 });
```

```
//Use a provider in controller
myApp.controller('MyCtrl',function($scope,
helloWorld) {
      $scope.hellos =  helloWorld.sayHello();
});
```

Note: 'Provider' appended to 'helloWorld' in module.config function

# Agenda

- Services Introduction
- Different Ways of registering Service
  - Using Value function
  - Using Service function
  - Using Factory function
  - Using  Provider function
- **Using a Service**
  - **Implicit DI**
  - **Explicit DI**
- AngularJS Internal Services
  - $timeout,$interval
  - $watch, $digest, $apply

# Using a Service

- Each Angular application has an injector, to manage the responsibility of dependency creation and injection.
- Injector is a service locator that is responsible for creating components, resolving their dependencies, and providing them to other components as requested.

```
$injector = angular.injector();
```

- Angular creates a single $injector when it bootstraps an application and uses the single $injector to invoke controller and service functions

```
// inferred (only works if code not minified/obfuscated)
$injector.invoke(function(serviceA){});
```

```
// inline array annotation (works with minified code)
    $injector.invoke(['serviceA', function(serviceA){}]);
```

# Using a Service - Implicit DI (1/2)

- Implicit DI is when you don't explicitly ask $injector to instantiate and get/inject a particular dependency.
- When Angular compiles the HTML, it processes the ng-controller directive.
- It in turn asks the injector to create an instance of the controller and instantiate its dependencies.

```
injector.instantiate(DemoController);
```

- After instantiating dependencies, it injects them into the controller function as follows:

```
var app = angular.module("DemoApp", []);
  // Controller is injected with $scope and 'greeter' service as dependencies
  app.controller('DemoController', ['$scope', 'greeter',
     function (s, g) {
         $scope.sayHello = function() { greeter.greet('Hello World'); };
           }   ]);

<div ng-controller="DemoController">
        <button ng-click="sayHello()"> Hello </button>
     </div>
```

- This is all done behind the scenes, the controller doesn't know about the injector.

# Using a Service - Implicit DI (2/2)

- Service/factory/provider function - all will lazily instantiate service once at the time of injection, cache and reuse the same instance (singleton)for further injections
- Service vs factory vs provider and their injection usage:
  - Service function - When declaring serviceName as an injectable argument you will be provided with an instance of the function. In other words new FunctionYouPassedToService()
  - Factory function - When declaring factoryName as an injectable argument you will be provided with the value that is returned by invoking the function reference passed to module.factory.
  - Provider function - When declaring providerName as an injectable argument you will be provided with ProviderFunction().$get().
    - The constructor function is instantiated before the $get method is called . ProviderFunction is the function reference passed to module.provider.

# Using a Service - Explicit DI

- Creates an injector object that can be used for retrieving services as well as for dependency injection

```
var myInjector = angular.injector(['mymodule' , 'ng']);
```

- Call get() on returned injector instance to resolve a manually specified dependency

```
var $http = myInjector.get("$http");  // After this, you can use $http just like before.
```

- The injector is also responsible for injecting services into functions, services can be injected into any function using the injector's invoke method;

```
var myFunction = function(greeting) { greeting('Ford Prefect'); };
$injector.invoke(myFunction);
```

- Injector will only create an instance of a service once and cache it by service's name; the next time service is injected using it's name, it returns the cached service object.

# Agenda

- Services Introduction
- Different Ways of registering Service
  - Using Value function
  - Using Service function
  - Using Factory function
  - Using  Provider function
- Using a Service
  - Implicit DI
  - Explicit DI
- **AngularJS Internal Services**
  - **$timeout,$interval**
  - **$watch, $digest, $apply**

# AngularJS Internal Services - $timeout, $interval

- AngularJS provide internal services to user which can be used in application.
- All internal service start with '$' e.g. $http, $window, $location
- Lets have a look at some of internal services and its use:
  - **$timeout** is Angular's wrapper for window.setTimeout and **$interval** is for scheduling a repeated function Call

```
var myapp = angular.module("myapp", []);
    myapp.controller("MyController", function($scope, $timeout)
                                          {  $timeout(callAtTimeout, 3000);
                                             $interval(callAtInterval, 5000); });
    function callAtTimeout() { console.log("Timeout occurred"); }
    function callAtInterval() { console.log("Interval occurred"); }
```

# AngularJS Internal Services - $watch, $digest (1/2)

- Angular achieves two way binding internally using $digest cycle.
- When you write an expression ({{aModel}}), behind the scenes Angular sets up a watcher on the scope model using $watch, which in turn updates the view whenever the model changes.

```
$scope.$watch('aModel', function(newValue, oldValue) {
  //update the DOM with newValue
});
```

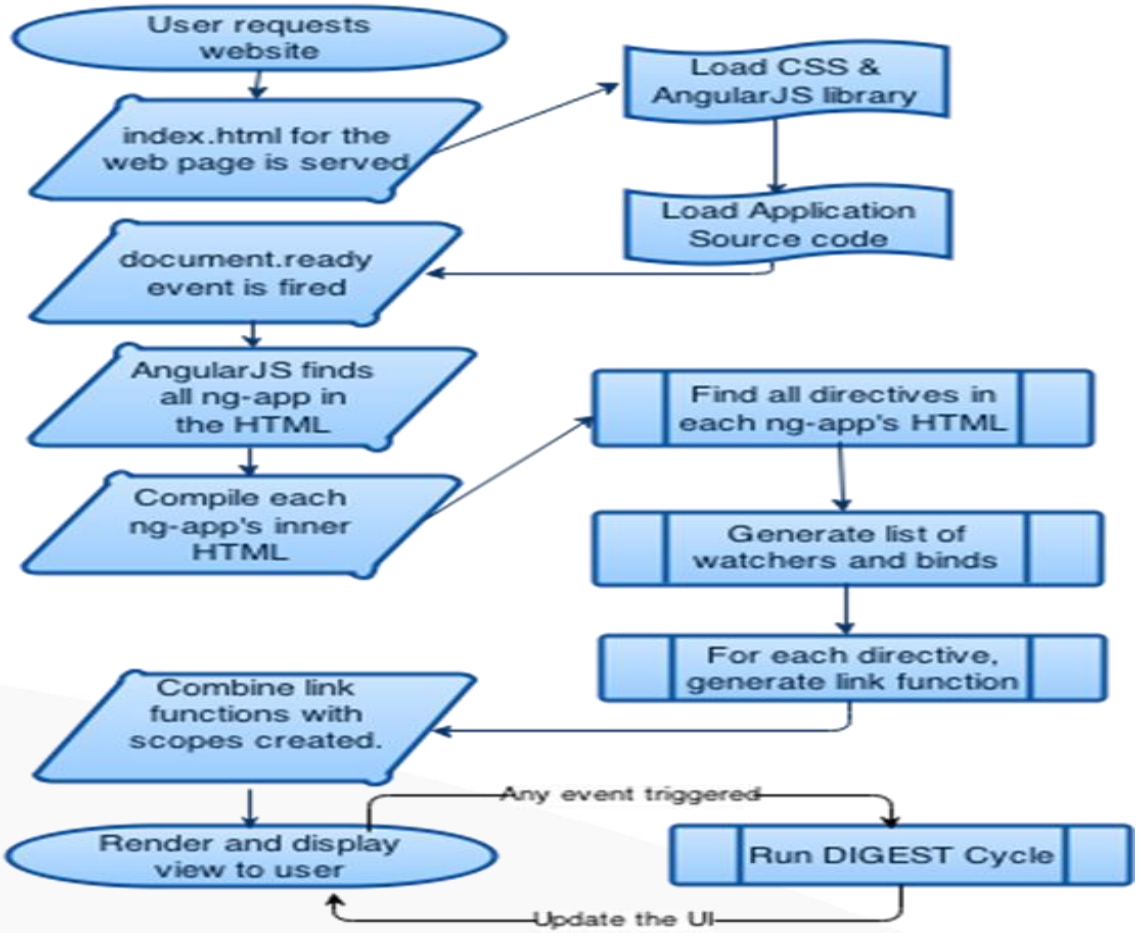
  - $watch listener function does not get executed on its own.

    $digest triggers $watch.

    Several built-in directives and services that change models (e.g. ng-model, ng-click, $timeout, etc.) automatically triggers a $digest cycle.
- A digest cycle starts at the $rootScope, and subsequently visits all the child scopes calling the watchers along the way.

# AngularJS Internal Services - $watch, $digest (2/2)

AngularJS life cycle



# AngularJS Internal Services - $digest, $apply

- Angular doesn't directly call $digest(), it calls $scope.$apply() which in turn calls $rootScope.$digest().
- As a result of this, a digest cycle starts at the $rootScope, and subsequently visits all the child scopes calling the watchers along the way.
- When you change model outside the Angular context (like DOM event listeners, setTimeout, XHR or third party libraries) then use $apply()
  - $scope.$apply();
- Manually calling $apply() informs Angular that model is changed outside the context and it should fire the watchers so that changes propagate properly.

# Services - Finding your way (1/2)

Technical Questions:

1. In which scenarios will you use $apply?
2. In which scenarios will you use $provider?

**As you start to work with AngularJS Services, you will frequently encounter technical issues which are not covered by this training.**

**How will you resolve these technical issues on AngularJS Services?**

# Services - Finding your way (2/2)

| Resources | Remarks |
|---|---|
| | |
| http://viralpatel.net/blogs/angularjs-service-factory-tutorial/ | Good blog post on Angularjs services |
| http://tutorials.jenkov.com/angularjs/dependency-injection.html | Good blog on AngularJS services |
| https://docs.angularjs.org/guide/providers | AngularJS official doc site |
| https://github.com/jedrichards/angularjs-handbook#components | Good handbook on AngularJS that covers Angularjs services |
| | |
| CurioCT - https://interct/SitePages/CurioCT.aspx | |
| CTCourses – Services | In addition to updated course material, CTCourse contains reference sites (Library) and list of project teams with expertise on Services |

# Thank You