AngularJS

Nagaraju Bende

Blog

http://nbende.wordpress.com

Twitter

@nbende

FaceBook

nbende

http://angularjs.org

Agenda

- Introduction to AngularJS
- Pre-Requisites
- Why AngularJS Only
- Getting Started
- MV* pattern of AngularJS
- Directives, Filters and Data Binding
- Views, Controllers and Scope
- Modules, Routes and Factories

AngularJS

- Front-end JS framework for creating web applications
- Open source maintained by Google
- MVC pattern/ MV* pattern
- Handles common (and often trying tasks) such as DOM manipulation, updating UI based on data or input, registering callbacks.
- Declarative programming

Pre-Requisites

HTML
JS
CSS
AJAX
SPA
Api's

SPA
TDD
BDD
DevOps
CDN/Cloud

Why AngularJS Only ?

- AngularJS redefines how to build front-end applications
- Application frameworks, like Backbone, EmberJS, and others, require developers to extend from JavaScript objects that are specific to their frameworks
- In Angular instead of manipulating the DOM “directly,” you annotate your DOM with metadata (directives), and Angular manipulates the DOM for you

Contd…

Why AngularJS Only ?

- AngularJS redefines how to build front-end applications
- Application frameworks, like Backbone, EmberJS, and others, require developers to extend from JavaScript objects that are specific to their frameworks
- In Angular instead of manipulating the DOM “directly,” you annotate your DOM with metadata (directives), and Angular manipulates the DOM for you

Contd…

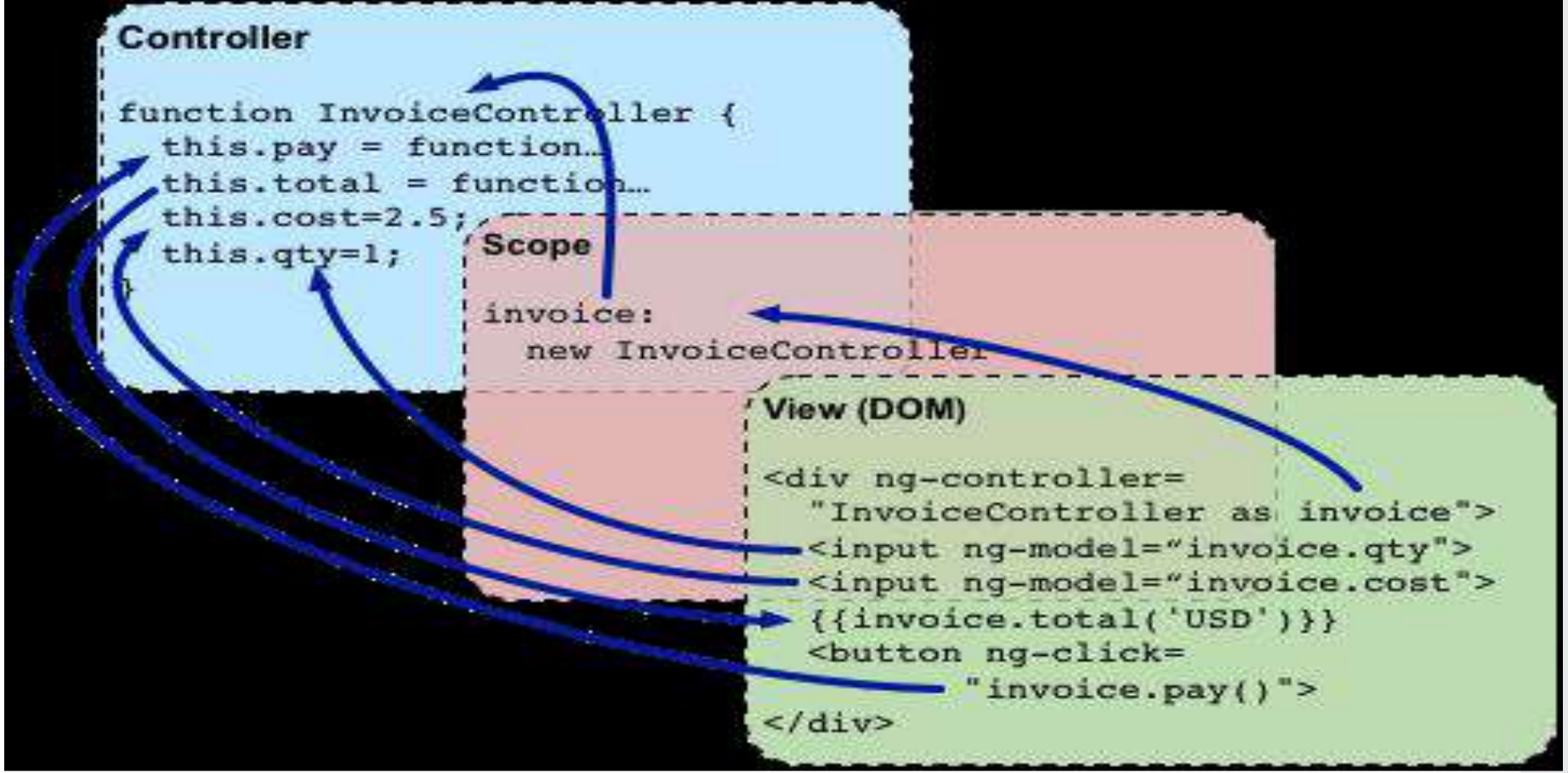
- Good for dynamic web sites/web apps (CRUD based)
- Framework imposes a structure that is good for organization
- Helps create responsive (fast) websites
- Easy to test – to create software that is easily maintained

Agenda

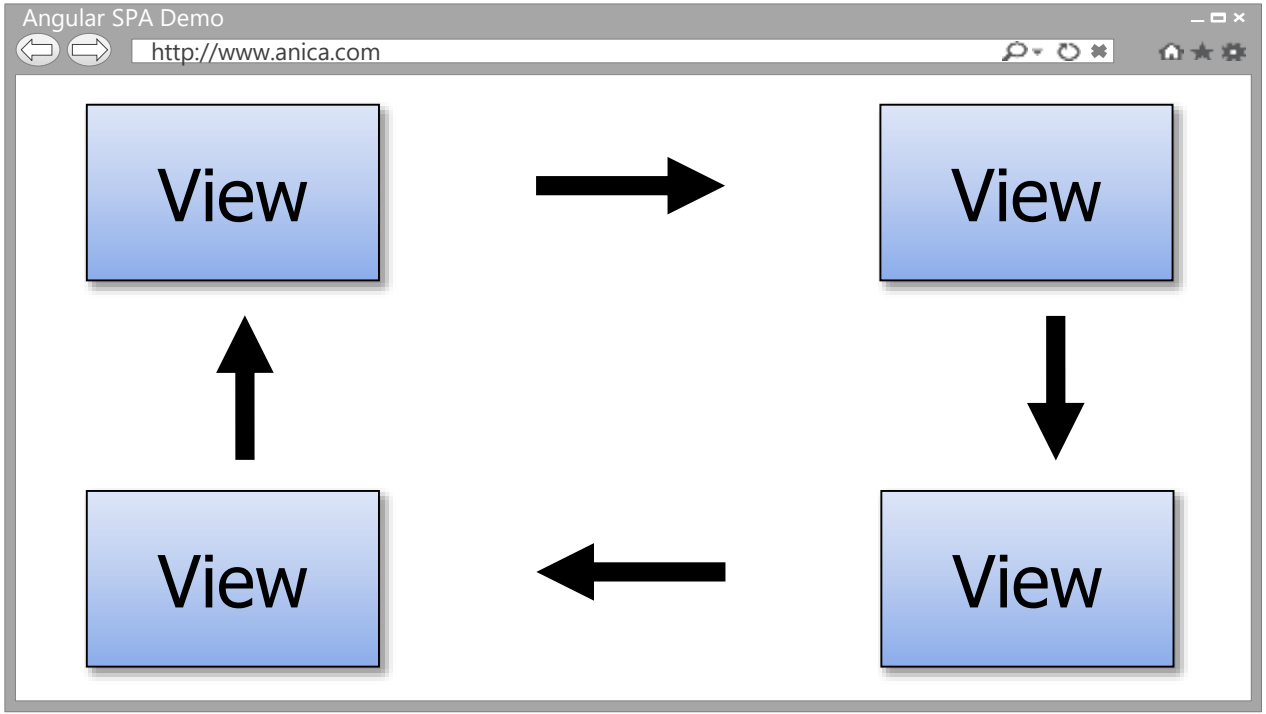
- History
- SPA
- Angular Architecture
- Directives
- Demos

Small History to know

- Miško Hevery and Adam Abrons in 2009
- Creators of Angular
- Used in 100's of projects in Google itself
- Angle braces → Angular JS (Adam Abrons)


Controller
function InvoiceController {
this.pay = function…
this.total = function…
this.cost=2.5;
this.qty=1;
}
Scope
invoice:
new InvoiceController
View (DOM)
<div ng-controller=
"InvoiceController as invoice">
<input ng-model="invoice.qty">
<input ng-model="invoice.cost">
{{invoice.total('USD')}}
<button ng-click=
"invoice.pay()">

Single Page Application (SPA)



The Challenge with SPAs

DOM Manipulation

History

Module Loading

Routing

Caching

Object Modeling

Data Binding

Ajax/Promises

View Loading

Data Binding | MVC | Routing | Testing

jqLite | Templates | History | Factories

AngularJS is a full-featured SPA framework

ViewModel | Controllers | Views | Directives

Controllers | Dependency Injection | Validation

Downloading the libraries

- Download AngularJS

  http://angularjs.org/

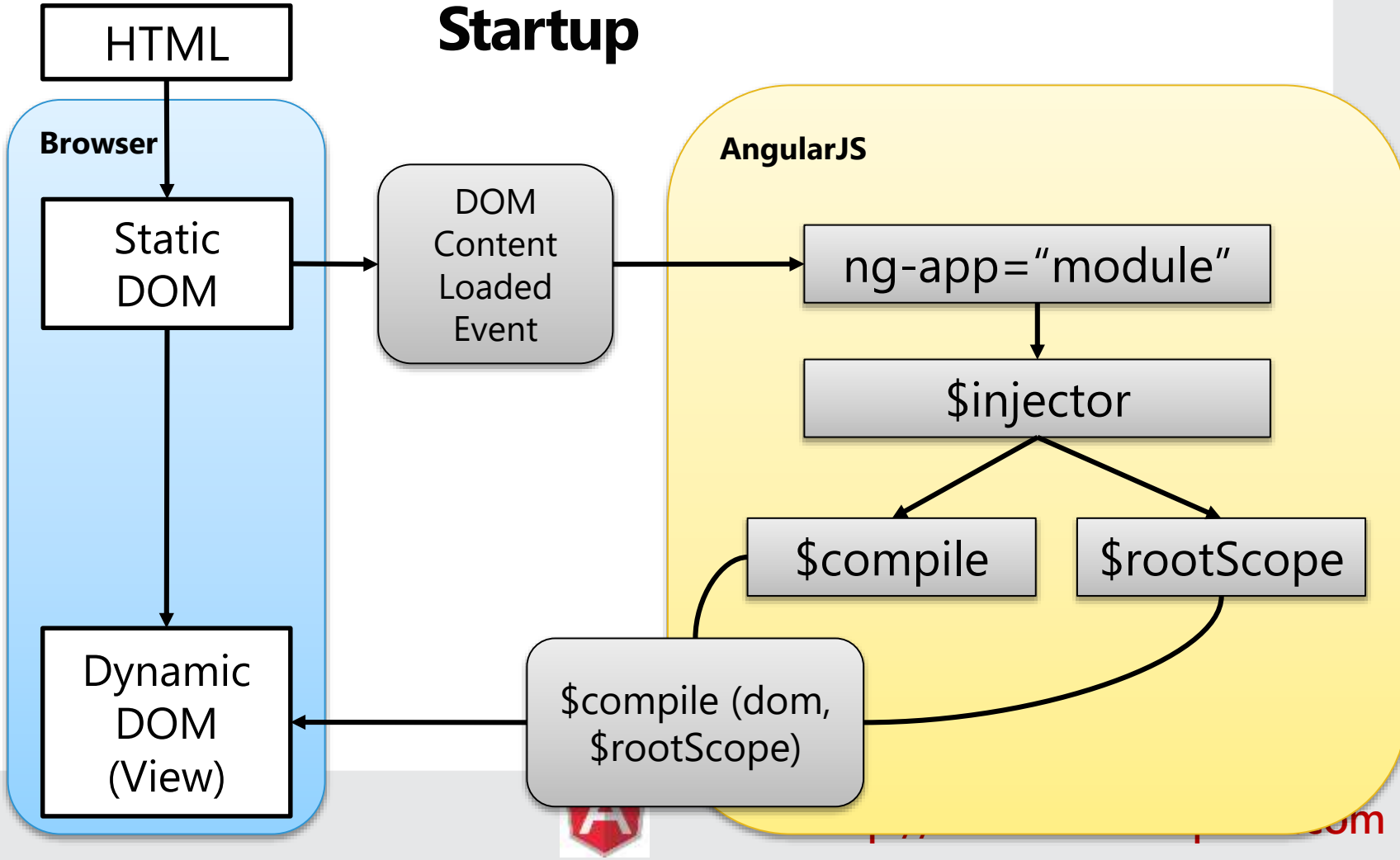
  We'll need angular.min.js

- Download Twitter Bootstrap

  http://getbootstrap.com/

  We'll need bootstrap.min.css

Startup


HTML
Browser
Static DOM
Dynamic DOM (View)
DOM Content Loaded Event
AngularJS
ng-app="module"
$injector
$compile
$rootScope
$compile (dom, $rootScope)

How AngularJS Works

- AngularJS will initialize when the DOM content is loaded
- Looks for the ng-app directive – if its found, that is the root of the app
- Directives can be declared a variety of ways: typically with the ng- prefix, but you can use data-ng
- It will load the module associated with the directive if specified

Directives

Directives

- Directives are markers (such as attributes, tags, and class names) that tell AngularJS to attach a given behavior to a DOM element (or transform it, replace it, etc.)
- A directive is an extension of the HTML vocabulary that allows us to create new behaviours. This technology lets the developers create reusable components that can be used within the whole application and even provide their own custom components

Using Directives and Data Binding

```
<!DOCTYPE html>
<html ng-app>
<head>
    <title></title>
</head>
<body>
    <div class="container">
        Name: <input type="text" ng-model="name" /> {{ name }}
    </div>

    <script src="Scripts/angular.js"></script>
</body>
</html>
```

Directive

Directive

Data Binding Expression

http://nbende.wordpress.com

Angular JS Built-in Directives

- The **ng-app** - Bootstrapping your app and defining its scope.
- The **ng-controller** - defines which controller will be in charge of your view.
- The **ng-repeat** - Allows for looping through collections
- **ng-bind**
- **ng-bindHtml**
- **ng-repeat**
- **ng-model**
- **ng-click**
- **ng-disable**
- **ng-class**
- **ng-options**
- **ng-style**
- **ng-show**
- **ng-hide**
- **ng-include**

AngularJS Help for Directives

- ngApp
- ngBind
- ngBindHtmlUnsafe
- ngBindTemplate
- ngChange
- ngChecked
- ngClass
- ngClassEven
- ngClassOdd
- ngClick
- ngCloak
- ngController
- ngCsp
- ngDblclick
- ngDisabled
- ngForm
- ngHide
- ngHref
- ngInclude
- ngInit
- ngList
- ngModel

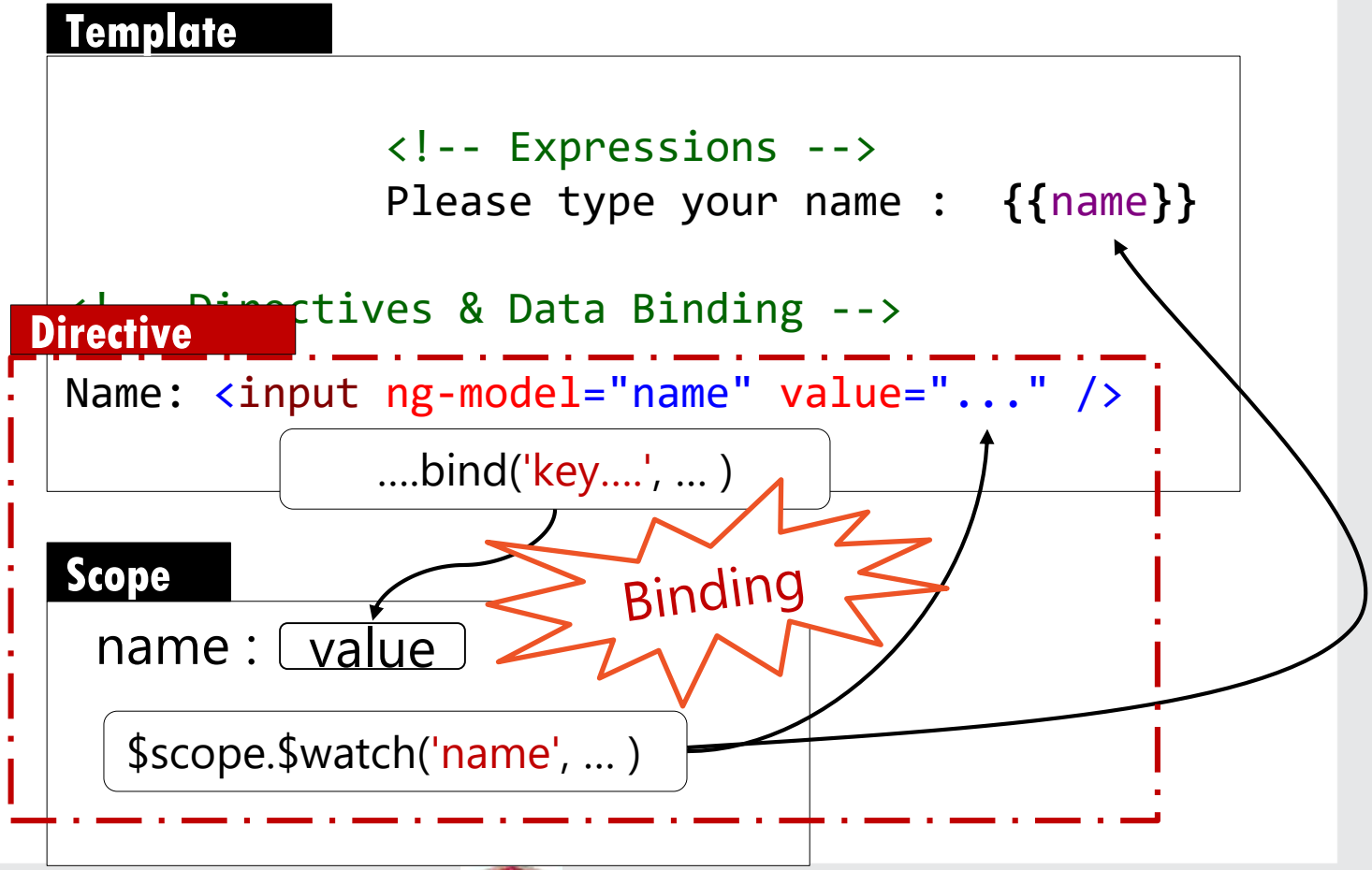
http://nbende.wordpress.com

Iterating with the ng-repeat Directive

```
<html ng-app="">
...

   <div class="container"
      ng-init="names=['Kiran', 'SCOTT', 'Smith','Shiva']">

       <h3>Looping with the ng-repeat Directive</h3>
       <ul>
           <li ng-repeat="name in names">{{ name }}</li>
       </ul>
   </div>

...
</html>
```

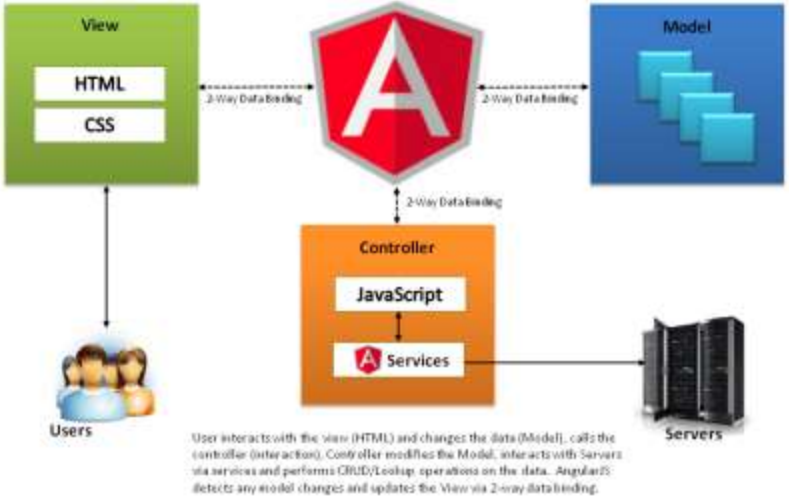
Iterate through names

Live DOM

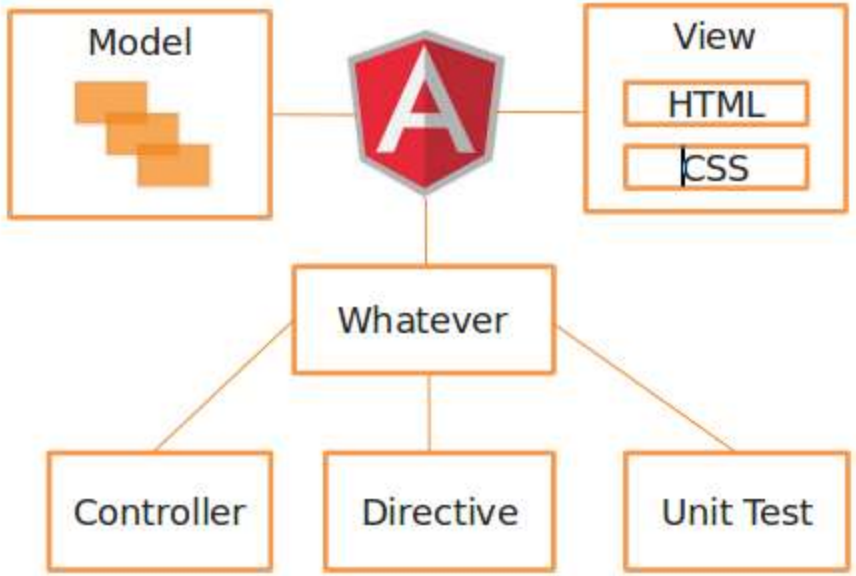

Template
<!-- Expressions -->
Please type your name : {{name}}
<!-- Directives & Data Binding -->
Directive
Name: <input ng-model="name" value="..." />
....bind('key....', ...)
Scope
name : value
Binding
$scope.$watch('name', ...)

Angular MVC

MVC


View
HTML
CSS
2-Way Data Binding
Model
2-Way Data Binding
2-Way Data Binding
Controller
JavaScript
Services
Users
Servers
User interacts with the view (HTML) and changes the data (Model), calls the controller (interaction). Controller modifies the Model, interacts with Servers via services and performs CRUD/Lookup operations on the data. AngularJS detects any model changes and updates the View via 2-way data binding.

MVW

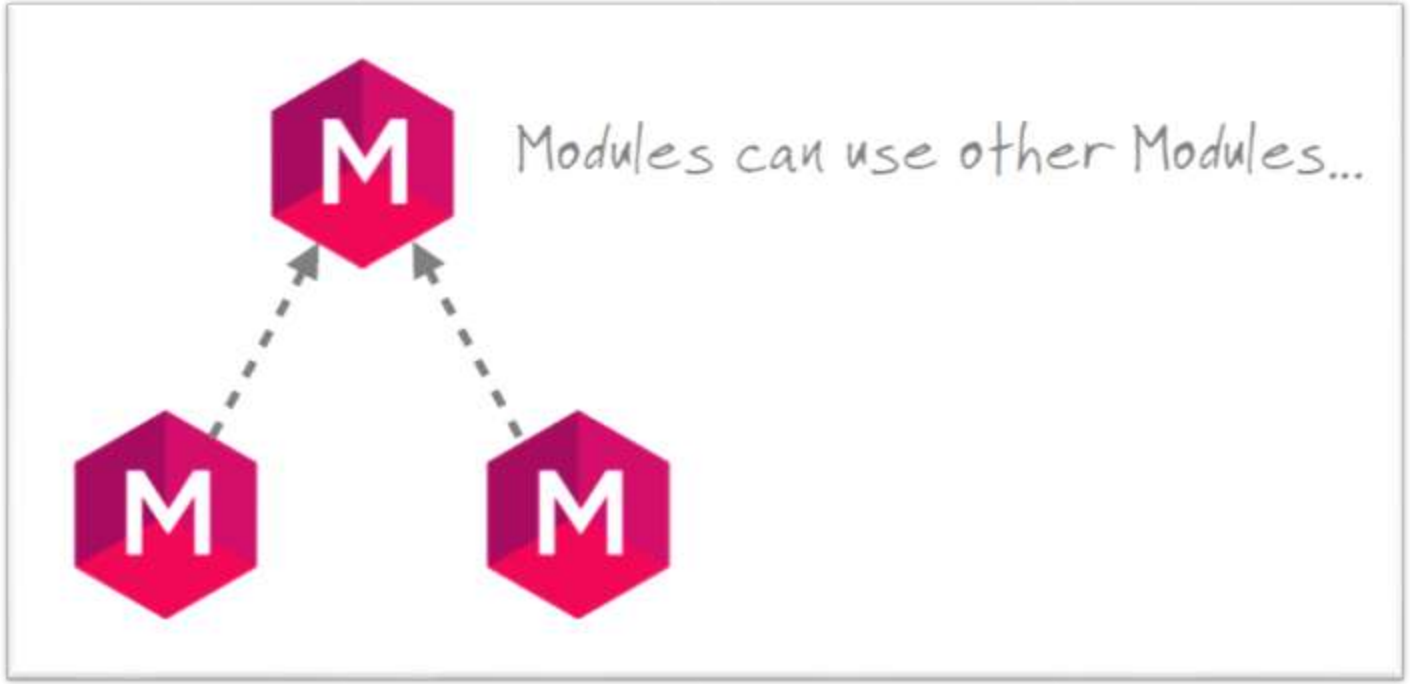

Model
View
HTML
CSS
Whatever
Controller
Directive
Unit Test

Modules

Angular - Modules

- Where we write pieces of our Angular application.
- Makes our code more maintainable, testable, and readable.
- Where we define dependencies for our app.
- Modules can use other Modules…

Angular - Modules



Views, Controllers and Scope

View, Controllers and Scope

$scope is the "glue" (ViewModel)
between a controller and a view

Creating a View and Controller

```
<div class="container" ng-controller="SimpleController">
    <h3>Adding a Simple Controller</h3>
    <ul>
        <li ng-repeat="cust in customers">
            {{ cust.name }} - {{ cust.city }}
        </li>
    </ul>
</div>

<script>
    function SimpleController($scope) {
        $scope.customers = [
            { name: 'Dave Jones', city: 'Phoenix' },
            { name: 'Jamie Riley', city: 'Atlanta' },
            { name: 'Heedy Wahlin', city: 'Chandler' },
            { name: 'Thomas Winter', city: 'Seattle' }
        ];
    }
</script>
```

Define the controller to use

Access $scope

$scope injected dynamically

Basic controller

Demos - Angular

Filters
and Data Binding

Using Filters

```
<ul>
    <li data-ng-repeat="cust in customers | orderBy:'name'">
        {{ cust.name | uppercase }}
    </li>
</ul>
```

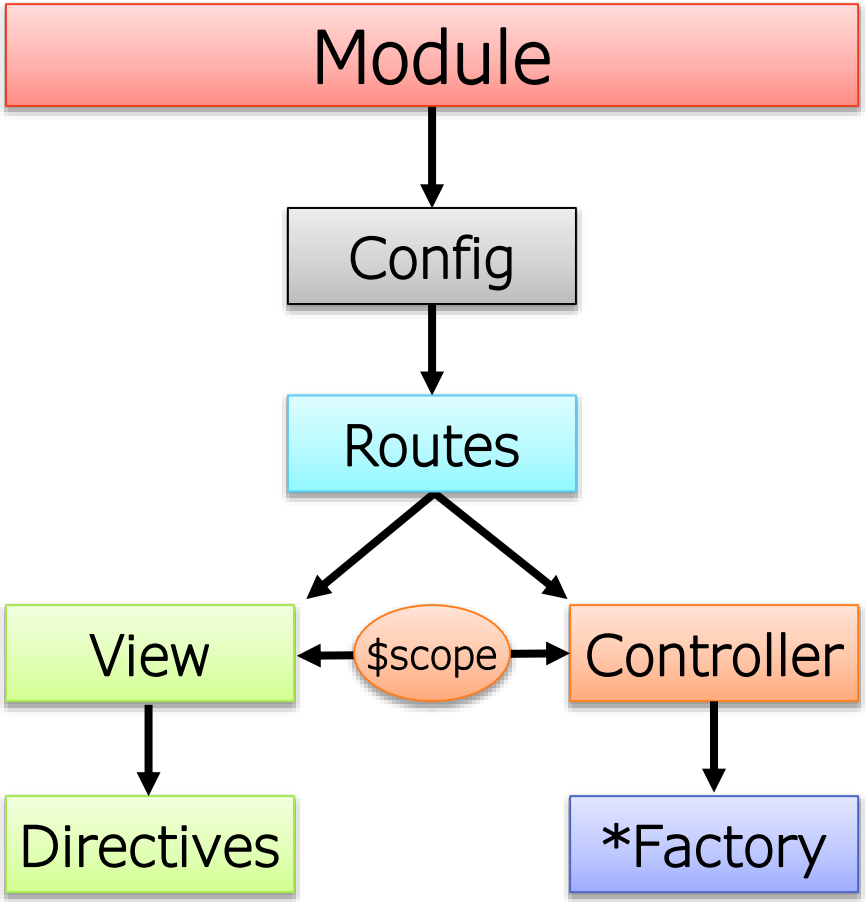
Order customers by name property

Filter customers by model value

```
<input type="text" data-ng-model="nameText" />
<ul>
    <li data-ng-repeat="cust in customers | filter:nameText | orderBy:'name'">
      {{ cust.name }} - {{ cust.city }}</li>
</ul>
```

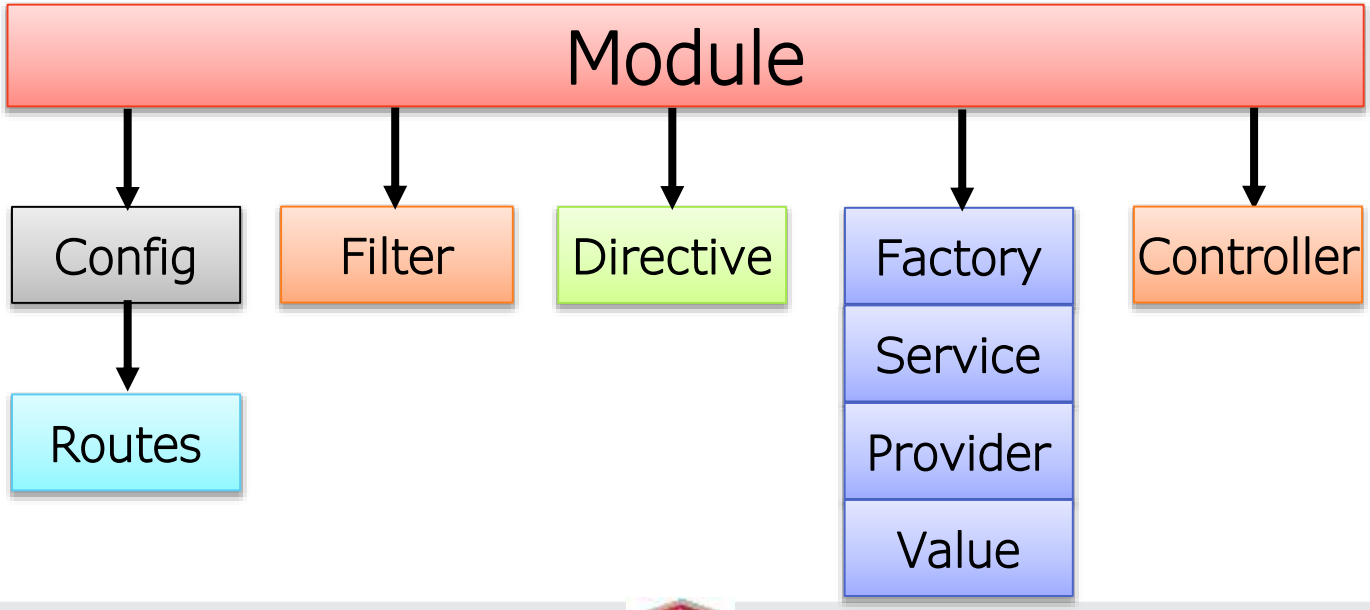
AngularJS Help for Filters

Modules, Routes and Factories


Module
Config
Routes
View
$scope
Controller
Directives
*Factory

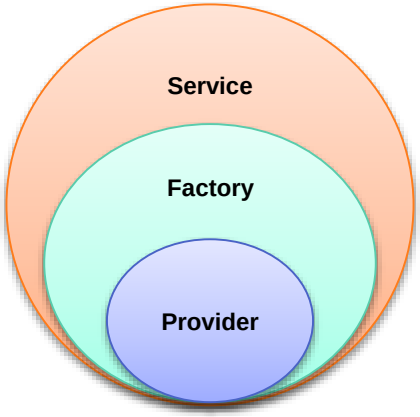
Modules are Containers

```
<html ng-app="moduleName">
```



Angular Services

- Angular services are **singletons** that carry out specific tasks.
- All services in Angular are **instantiated lazily**.
- There are three functions for creating generic services, with different levels of complexity and ability:



Provider

```
function provider(name, provider_) {

    if (isFunction(provider_) || isArray(provider_)) {
        provider_ = providerInjector.instantiate(provider_);
    }

    if (!provider_.$get) {
        thr                                                    efine $get
                                                               ctory method.');
    }

    return                                                     provider_;
}
```

```
// Register an object provider
myApp.provider('awesome', {
    $get: function () {
        return 'awesome data';
    }
});
```

Factory

```
function factory(name, factoryFn) {
    return provider(name, { $get: factoryFn });
}
```

```
// Register factory
myApp.factory('awesome',
function() {
    return 'awesome data';
});
```

Service

Everything that uses the service will get the **same instance**!

```
function se
    return
{
     // inst
         retu
    }]);
}
```

```
    this.color = 'grey';
}

Gandalf.prototype.comeBack = function() {
    this.color = 'white';
};

gandalf.service('gandalfService', Gandalf);
```

Creating a Module

What's the Array for?

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

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

Module that demoApp depends on

Creating a Controller in a Module

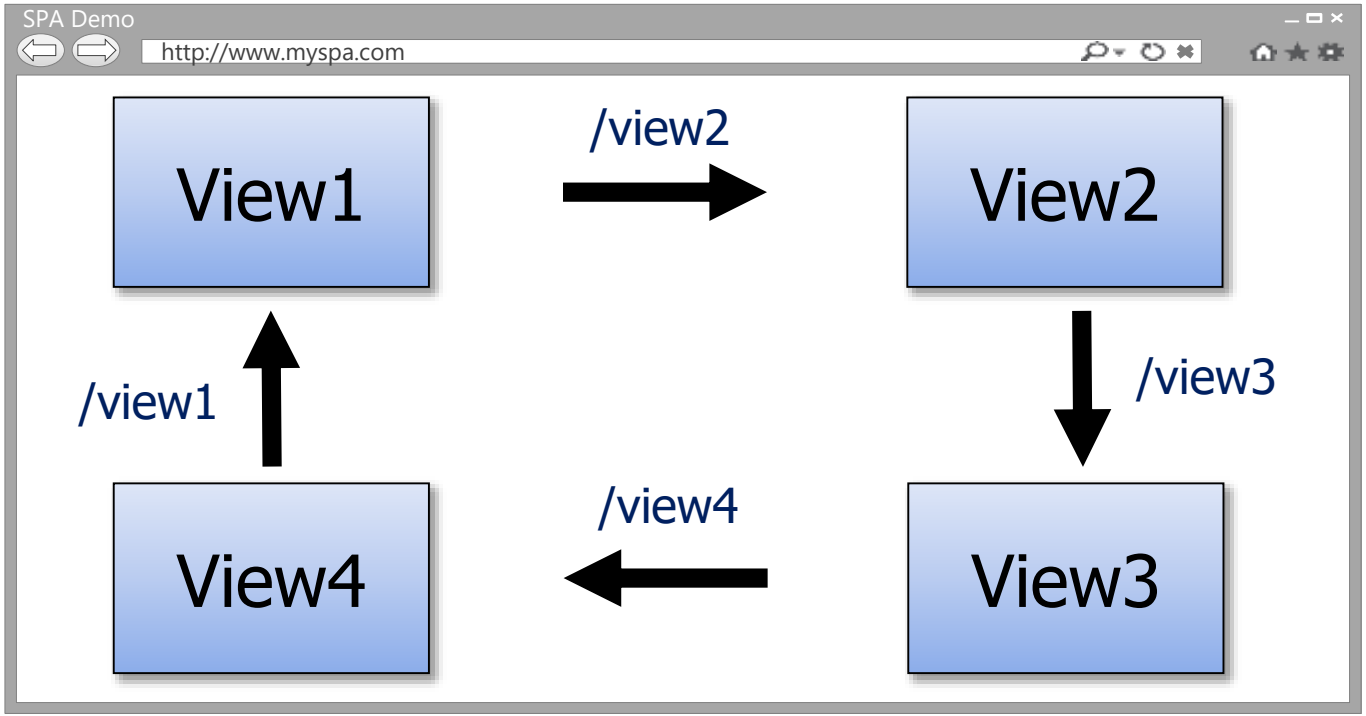
```
var demoApp = angular.module('demoApp', []);
```

Define a Module

Define a Controller

```
demoApp.controller('SimpleController', function ($scope) {
    $scope.customers = [
       { name: 'Dave Jones', city: 'Phoenix' },
       { name: 'Jamie Riley', city: 'Atlanta' },
       { name: 'Heedy Wahlin', city: 'Chandler' },
       { name: 'Thomas Winter', city: 'Seattle' }
    ];

});
```

The Role of Routes



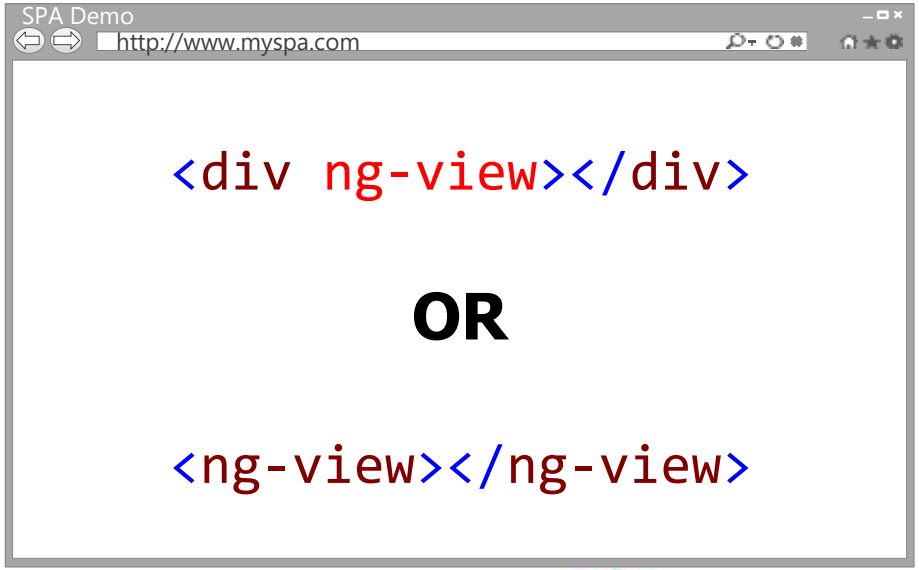
Defining Routes

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

demoApp.config(function ($routeProvider) {
    $routeProvider
        .when('/',
            {
                controller: 'SimpleController',
                templateUrl:'View1.html'
            })
        .when('/view2',
            {
                controller: 'SimpleController',
                templateUrl:'View2.html'
            })
        .otherwise({ redirectTo: '/' });
});
```

Define Module Routes

Where do Views Go in a Page?

Dynamically loaded views are injected into the shell page as a module loads:

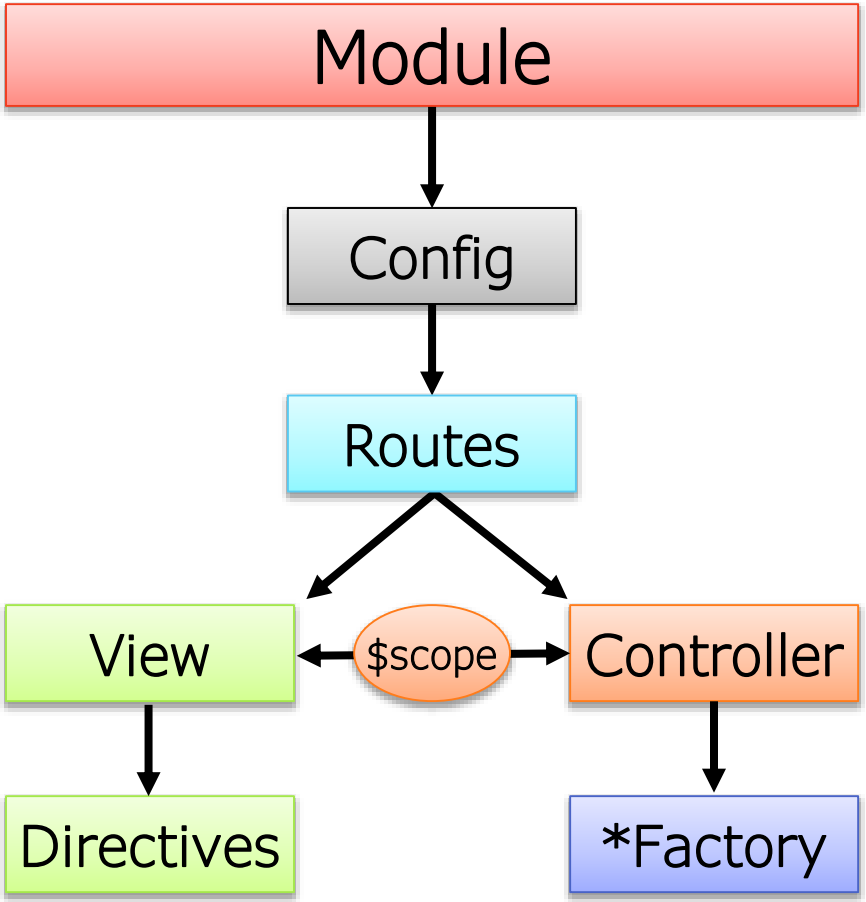


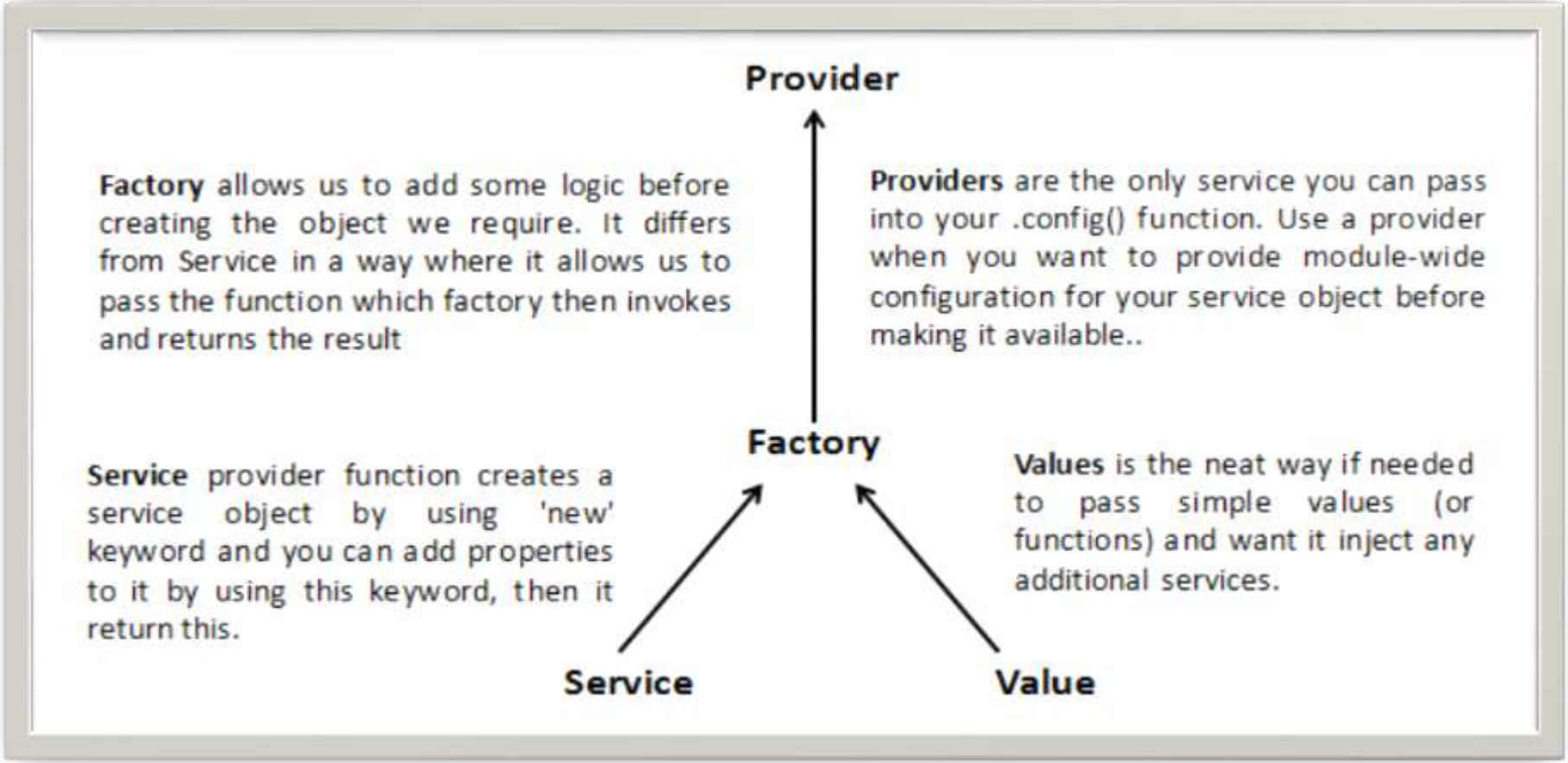
View1

The Role of Factories

```
var demoApp = angular.module('demoApp', [])
 .factory('simpleFactory', function () {
    var factory = {};
    var customers = [ ... ];
    factory.getCustomers = function () {
        return customers;
    };
    return factory;
 })
 .controller('SimpleController', function ($scope,
    simpleFactory) {
     $scope.customers = simpleFactory.getCustomers();
});
```

Factory injected into controller at runtime


Module
Config
Routes
View
$scope
Controller
Directives
*Factory



Features / Recipe type	Factory	Service	Value	Constant	Provider
can have dependencies	yes	yes	no	no	yes
uses type friendly injection	no	yes	yes*	yes*	no
object available in config phase	no	no	no	yes	yes**
can create functions	yes	yes	yes	yes	yes
can create primitives	yes	no	yes	yes	yes

AngularJS .service

```
module.service('MyService', function() {

    this.method1 = function() {
        //..method1 logic
    }

    this.method2 = function() {
        //..method2 logic
    }
});
```

AngularJS .factory

```
module.factory('MyFactory', function() {

    var factory = {};
    factory.method1 = function() {
        //..method1 logic
      }
    factory.method2 = function() {
        //..method2 logic
      }

    return factory;
});
```

- **Factory Provider**
  - Gives us the function's return value ie. You just create an object, add properties to it, then return that same object.
  - When you pass this service into your controller, those properties on the object will now be available in that controller
  - through your factory.
  - Singleton and will only be created once
  - Reusable components
  - Factory are a great way for communicating between controllers like sharing data.
  - Can use other dependencies
  - Usually used when the service instance requires complex creation logic
  - Cannot be injected in .config() function.
  - Used for non configurable services
  - If you're using an object, you could use the factory provider.
  - Syntax: module.factory('factoryName', function);

- **Service Provider**

  - Gives us the instance of a function (object)- You just instantiated with the ‘new’ keyword and you’ll add properties to ‘this’
  - and the service will return ‘this’.When you pass the service into your controller, those properties on ‘this’ will now
  - be available on that controller through your service. (Hypothetical Scenario)
  - Singleton and will only be created once
  - Reusable components
  - Services are used for communication between controllers to share data
  - You can add properties and functions to a service object by using the `this` keyword
  - Dependencies are injected as constructor arguments
  - Used for simple creation logic
  - Cannot be injected in `.config()` function.
  - If you're using a class you could use the service provider
  - Syntax: `module.service('serviceName', function);`

Summary

- AngularJS provides a robust "SPA" framework for building robust client-centric applications

- Key features:
  - Directives and filters
  - Two-way data binding
  - Views, Controllers, Scope
  - Modules and Routes

Thank you!


@nbende


nbende.wordpress.com


www.linkedin.com/nbende