

Instructor Notes:

Add instructor notes here.

AngularJS

Introduction to AngularJS

February 15, 2015

Proprietary and Confidential

- 1 -



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- JavaScript fundamentals
- MV* Frameworks
- AngularJS Fundamentals



Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

JavaScript fundamentals

- Browser gets the HTML text of the page, parses it into DOM structure, lays out the content of the page, and styles the content before it gets displayed.
- HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications.
- JavaScript has become one of the most popular client side scripting language on the web which is used to create dynamic views in web-applications.
- JavaScript plays a major role in the usage of ajax, user experience and responsive web design.
- DOM manipulation libraries like jQuery simplifies client side scripting, but it is not solving the problem of handling separation of concerns.
- Fortunately there are few libraries and frameworks are available to accomplish this task.

February 15, 2015

Proprietary and Confidential

• 3 •



Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Objects in JavaScript

- **JavaScript is an object oriented language. In JavaScript we can define our own objects and assign methods, properties to it.**
- **In JavaScript objects are also associative arrays (or) hashes (key value pairs).**
 - Assign keys with obj[key] = value or obj.name = value
 - Remove keys with delete obj.name
 - Iterate over keys with for(key in obj), iteration order for string keys is always in definition order, for numeric keys it may change.
- **Properties, which are functions, can be called as obj.method(). They can refer to the object as this. Properties can be assigned and removed any time.**
- **A function can create new objects when run in constructor mode as new Func(params). Names of such functions are usually capitalized**

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Creating objects

➤ An empty object can be created using

— obj = new Object(); (or) obj = {};

— It stores values by key, with that we can assign or delete it using "dot notation" or "Square Brackets" (associative arrays).

using dot notation

```
> var employee = {};
undefined
> employee.Id = 714709;
714709
> employee.Name = "Karthik"
"Karthik"
> employee.Name
"Karthik"
> delete employee.Name
true
> employee
Object {Id: 714709} [employee.Name deleted]
```

using square brackets

```
> var employee = {};
undefined
> employee["Id"] = 714709;
714709
> employee["Name"] = "Karthik"
"Karthik"
> employee
Object {Id: 714709, Name: "Karthik"}
> delete employee["Name"]
true
> employee
Object {Id: 714709}
```

February 15, 2015

Proprietary and Confidential

• 5 •

IGATE
Speed.Agency.Imagination

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Checking for non existing property in object

- If the property does not exist in the object , then `undefined` is returned
- To check whether key existence we can use `in` operator

```
> var employee = {}  
undefined  
> employee.Id      //Checking non existing Property  
undefined  
> employee.Id === undefined // strict comparison  
true  
> "Id" in employee    //"in" operator to check for keys existence  
false  
> employee.Id = 714709  
714709  
> "Id" in employee  
true
```

February 15, 2015

Proprietary and Confidential

- 6 -



Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Iterating over object keys

- We can iterate over keys using `for .. In`

```
> var employee = {}  
undefined  
> employee.Id = 714709  
714709  
> employee.Name = "Karthik"  
"Karthik"  
> for(key in employee) { console.log("Key : " + key + " Value : " + employee[key]) }  
Key : Id Value : 714709  
Key : Name Value : Karthik
```

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Object reference

- A variable which is assigned to object actually keeps reference to it.
- It acts like a pointer which points to the real data. Using reference variable we can change the properties of object.
- Variable is actually a reference, not a value when we pass an object to a function.

```
> var employee = {};
undefined
> employee.Id = 714709;
714709
> var obj = employee; // now obj points to same object
undefined
> obj.Id = 707224;
707224
> employee.Id
707224
```

February 15, 2015 | Proprietary and Confidential | - 8 -



Pass by value

```
> function incrementValue(x){
    x++;
    console.log("Inside function x = "+x);
}
```

```
var x = 5;
incrementValue(x); //Passing the Value
console.log("x = "+x);
```

```
Inside function x = 6
```

```
x = 5
```

Pass by reference

```
> function incrementValue(obj){
    obj.x++;
    console.log("Inside function x = "+obj.x);
}
```

```
var obj = {x:5};
incrementValue(obj); //Passing the reference
console.log("x = "+obj.x);
```

```
Inside function x = 6
```

```
x = 6
```

Instructor Notes:

Add instructor notes here.

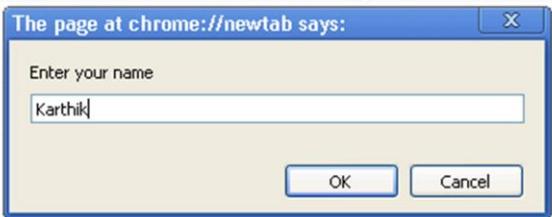
1.1: JavaScript Fundamentals

this keyword

- When a function is called from the object, *this* becomes a reference to this object.

```
> var foo = {  
    name : "Guest",  
    setName : function(){  
        this.name = prompt('Enter your name'); //this acts as a reference to foo object  
    },  
    getName : function(){  
        console.log("Your name is : "+this.name);  
    }  
};  
undefined  
> foo.getName();  
Your name is : Guest  
< undefined  
> foo.setName()  
undefined  
> foo.getName();  
Your name is : Karthik  
< undefined
```

prompts for name when foo.setName() is called



February 15, 2015

Proprietary and Confidential

- 9 -

IGATE

Speed.Agency.Imagination

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Constructor Function

- We can create an object using `obj = {....}`
- Another way of creating an object in JavaScript is to construct it by calling a function with `new` directive (Constructor function). Constructor functions should be in Pascal case.
- It takes `this`, which is initially an empty object, and assigns properties to it. The result is returned (unless the function has explicit return).

February 15, 2015

Proprietary and Confidential

+ 10 +



Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Constructor Function

```
> function Calculator(firstVar,secondVar){  
    this.firstVar = firstVar;  
    this.secondVar = secondVar;  
    this.sum = function(){  
        return this.firstVar + this.secondVar;  
    }  
}  
undefined  
> new Calculator(5,5); // returns this  
► Calculator {firstVar: 5, secondVar: 5, sum: function}  
> var calcObj1 = new Calculator(5,5);  
undefined  
> var calcObj2 = new Calculator(15,15);  
undefined  
> calcObj1.sum();  
10  
> calcObj2.sum();  
30
```

February 15, 2015

Proprietary and Confidential

• 11 •

IGATE
Speed.Agency.Imagination

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Prototypal inheritance

- In JavaScript, the inheritance is prototype-based. Instead of class inherits from other class, an object inherits from another object.
- object inherits from another object using the following syntax.
- `childObject.__proto__ = baseObject`
 - Above mentioned syntax provided by Chrome / FireFox. In other browsers the property still exists internally, but it is hidden
- `childObject = Object.create(baseObject)`
- `ConstructorFunction.prototype = baseObject`
 - Above mentioned syntax works with all modern browsers.

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Prototypal inheritance using `__proto__`

```
> var foo = {  
    fooVar : "Foo Variable",  
    fooMethod : function(){  
        console.log(this.fooVar);  
    }  
}  
  
var bar = {  
    barVar : "Bar Variable"  
}  
< undefined  
> bar.__proto__ = foo;      //bar object inherits from foo  
< ► Object {fooVar: "Foo Variable", fooMethod: function}  
> bar  
< ► Object {barVar: "Bar Variable", fooVar: "Foo Variable", fooMethod: function}
```

February 15, 2015

Proprietary and Confidential

+ 13 +

IGATE

Speed. Agility. Imagination

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Prototypal inheritance using Object.create()

```
> var foo = {  
    fooVar : "Foo Variable",  
    fooMethod : function(){  
        console.log(this.fooVar);  
    }  
}  
  
< undefined  
> var bar = Object.create(foo) //bar object inherits from foo object  
< undefined  
> bar  
< ► Object {fooVar: "Foo Variable", fooMethod: function}  
> bar.barVar = "Bar Variable";  
< "Bar Variable"  
> bar  
< ► Object {barVar: "Bar Variable", fooVar: "Foo Variable", fooMethod: function}
```

February 15, 2015

Proprietary and Confidential

• 14 •

IGATE
Speed. Agility. Imagination

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Prototypal inheritance using prototype

```
> function Employee(){
    this.Id = 0;
    this.Name = "";
}

function Manager(){}
//Manager Inherits Employee object
> Manager.prototype = new Employee();
< Employee {Id: 0, Name: ""}
> var anil = new Manager();
< undefined
> anil
< Manager {Id: 0, Name: ""} // All objects created by new Manager will have
> anil.Id = 5085;           // Id and Name
< 5085
> anil.Name = "Anil Patil";
< "Anil Patil"
> anil
< Manager {Id: 5085, Name: "Anil Patil"}
```

February 15, 2015

Proprietary and Confidential

+ 15 +

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Prototypal inheritance

- `Object.getPrototypeOf(obj)` returns the value of `obj.__proto__`.

```
> var foo = {fooVar : "Foo Variable"};
  var bar = Object.create(foo);
< undefined
> Object.getPrototypeOf(bar)
< Object {fooVar: "Foo Variable"}
> Object.getPrototypeOf(bar) === foo
< true
```

- `for..in` loop lists properties in the object and its prototype chain.

`obj.hasOwnProperty(prop)` returns true if property belongs to that object.

```
> var foo = {fooVar : "Foo Variable"};
  var bar = {barVar : "Bar Variable"};
  bar.__proto__ = foo;
  for(property in bar){
    if(bar.hasOwnProperty(property))
      console.log("Own Property : "+property);
    else
      console.log("Inherited Property : "+property);
  }
Own Property : barVar
Inherited Property : fooVar
```



Speed.Agency.Imagination

February 15, 2015

Proprietary and Confidential

< 16 >

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Static variables and methods

- In JavaScript we can directly put data into function object which acts like **Static member**.
- Static Members need to be accessed directly by Object name, cannot be accessed by reference variable. Static members gets created when the first object gets created.

```
> var Employee = function(){}
  Employee.CompanyName = "IGATE";
  Employee.doWork = function(){
    console.log('Work Implementation');
  }
<
< undefined
> Employee.CompanyName
< undefined
> new Employee();
< Employee {}
> Employee.CompanyName
< "IGATE"
> Employee.doWork()
  Work Implementation
```

February 15, 2015

Proprietary and Confidential

+ 17 +



In JavaScript :

private variables are declared with the 'var' keyword inside the object, and can only be accessed by private functions and privileged methods.

private functions are declared inline inside the object's constructor (or alternatively may be defined via var functionName=function(){...}) and may only be called by privileged methods (including the object's constructor).

privileged methods are declared with this.methodName=function(){...} and may be invoked by code external to the object.

public properties are declared with this.variableName and may be read/written from outside the object.

public methods are defined by Classname.prototype.methodName = function(){...} and may be called from outside the object.

prototype properties are defined by Classname.prototype.propertyName = someValue

static properties are defined by Classname.propertyName = someValue

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

JavaScript Functions

- JavaScript treats functions as objects(first-class functions).
- In JavaScript functions can be instantiated, returned by other functions, stored as elements of arrays and assigned to variables.
- A function with no name is called an anonymous function.
- Closure is a function to which the variables of the surrounding context are bound by reference.
- JavaScript function acts as a constructor when we use it together with the new operator

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Working with JavaScript Functions

- Declaring the function anonymously

```
function(){  
    console.log('IGATE');  
}
```

- Invoking the anonymous function. Function executes immediately after declaration.

```
(function(){  
    console.log('IGATE');  
})();
```

February 15, 2015

Proprietary and Confidential

+ 19 +



Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Working with JavaScript Functions

- Declaring a named function. function doSomething will be available inside the scope in which it's declared.

```
function doSomething(){  
    console.log('IGATE');  
}  
  
/* Inner Scope */  
(function(){  
    doSomething();  
})();
```

- Assigning function to a variable.

```
var doSomething = function(){  
    console.log('IGATE');  
}
```



February 15, 2015 | Proprietary and Confidential | * 20 *

Instructor Notes:

Add instructor notes here.

1.1: JavaScript Fundamentals

Working with JavaScript Functions

```
/*Anonymous Closures*/
(function(){
    var data = "Closing the variables inside the function from the rest of
    the world"
    console.log('Closure Invoked');
})();

var employee = function(){
    this.employeeId = 0;
    this.name = "";
};

/* JavaScript function acts as a constructor */
var emp = new employee();
```

February 15, 2015

Proprietary and Confidential

+ 21 +



Instructor Notes:

Add instructor notes here.

Demo

- Working-with-Javascript-Functions
- Closure-Demo



IGATE
Speed. Agility. Imagination

February 15, 2015

Proprietary and Confidential

• 22 •

Instructor Notes:

Add instructor notes here.

1.2: MV* Frameworks

MV* Frameworks

- MV* Frameworks are designed to make our code easier to maintain and to improve the user experience
- MV* framework is nothing but the popular patterns like
 - Model-View-Controller(MVC)
 - Model-View-ViewModel(MVVM)
 - Model-View-Presenter(MVP)
 - MVW(hatever works for you)
- Idea of all the patterns is to separate Model, View and the Controller (the logic that hooks up model and view)
- AngularJS, Backbone.JS, Knockout, EmberJS, Meteor, ExtJS are some of the famous MV* framework libraries.

February 15, 2015

Proprietary and Confidential

23



Instructor Notes:

Add instructor notes here.

1.2: MV* Frameworks

Model, View and Controllers

Model

- Contains the data which we are using in our application

View

- Displays the data to the user and read the user input.

Controller

- Format the data for views and handle application state.

February 15, 2015

Proprietary and Confidential

~ 24 ~



Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

Introduction to AngularJS

- AngularJS is an open source JavaScript library that is sponsored and maintained by Google.
- Developed in 2009 by Misko Hevery. Publicly released as version 0.9.0 in Oct 2010.
- AngularJS makes it easy to build interactive, modern web applications by increasing the level of abstraction between the developer and common web app development tasks by following Model–View–Controller (MVC) pattern.
- AngularJS lets you to extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.
- AngularJS helps us to create single page applications easily.
 - No page refresh on page change and different data on each page

February 15, 2015

Proprietary and Confidential

* 25 *



AngularJS is a client-side technology, written entirely in JavaScript.

It works with the long-established technologies of the web (HTML, CSS, and JavaScript) to make the development of web apps easier and faster than ever before.

It is a framework that is primarily used to build single-page web applications. AngularJS makes it easy to build interactive, modern web applications by increasing the level of abstraction between the developer and common web app development tasks.

The AngularJS team describes it as a “structural framework for dynamic web apps.”

Single page applications can be done with just JavaScript and AJAX calls, Angular will make this process easier

Instructor Notes:

Use the demos mentioned below to demonstrate AngularJS features

- AngularJs-PhoneBookApp
- AngularJs-Service
- AngularJs-Filters
- AngularJs-Routing

1.3: AngularJS Introduction

AngularJS Features

- Extending HTML to add dynamic nature so that we can build modern web applications with separation of application logic, data models, and views templates
- Two way binding
 - It synchronize the data between model and view, view component gets updated when the model get change and vice versa, no need for events to accomplish this
- Templates can be created using HTML itself
- Testability is the primary consideration in AngularJS. It supports both isolated unit tests and Integrated end to end test
- It also supports Routing, Filtering, Ajax calls, data binding, caching, history, and DOM manipulation.

February 15, 2015 | Proprietary and Confidential | * 26 *



HTML is designed for static webpages, but developing a dynamic site with HTML we need to do many tricks to achieve what we want. With AngularJS extending the HTML it is really simple to make a dynamic site in a proper MVC structure.

With AngularJS we can achieve

Create a template and reuse it in application multiple times.

- Can bind data to any element in two ways
- Can directly call the code-behind code in your html.
- Easily validate forms and input fields before submitting it.
- Can control complete DOM structure show/hide, changing everything with AngularJS properties.

In other JavaScript frameworks, we are forced to extend from custom JavaScript objects and manipulate the DOM from the outside. For instance, using jQuery, to add a button in the DOM, we'll have to know where we're putting the element and insert it in the appropriate place.

```
var $btn = $("<button>jQuery Button</button>");  
$("#target").append($btn);
```

AngularJS, on the other hand, augments HTML to give it native Model-View-Controller (MVC) capabilities. It enables the developer, to encapsulate a portion of entire page as one application, rather than forcing the entire page to be an AngularJS application.

This distinction is particularly beneficial, if the web application already includes another framework or if there is a need to make a portion of the page dynamic

Instrumented while the rest operates as a static page or is controlled by another JavaScript framework.

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJS Controller and Scope

- Controllers primary responsibility is to create scope object (\$scope), It also constructs the model on \$scope and provides commands for the view to act upon \$scope.
- Scope communicate with view in two way communication
- Scope exposes model to view, but scope is not a model. Model is nothing but the data present in the scope.
- View can be binded to the functions on the scope.
- We can modify the model using the methods available on the scope.

\$scope is the glue between Controller and Model

February 15, 2015 | Proprietary and Confidential | + 27 +

IGATE
Speed.Agency.Imagination

Scopes are a core fundamental of any Angular app. They are used all over the framework. \$scope object is where we define the business functionality of the application, the methods in our controllers, and properties in the views.

Scopes are the source of truth for the application state. Because of this live binding, we can rely on the \$scope to update immediately when the view modifies it, and we can rely on the view to update when the \$scope changes.

\$scopes in AngularJS are arranged in a hierarchical structure so that we can reference properties on parent \$scopes.

It is ideal to contain the application logic in a controller and the working data on the scope of the controller.

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJS Model

- The model is simply a plain old JavaScript object, does not use getter/setter methods or have any special framework-specific needs.
- Changes are immediately reflected in the view via the two-way binding feature.
- All model objects stem from scope object.
- Typically model objects are initialized in controller code with syntax like:
 - `$scope.companyName = "IGATE";`
- In the HTML template, that model variable would be referenced in curly braces such as: `{{companyName}}` without the `$scope` prefix.

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJS View and Templates

- The View in AngularJS is the compiled DOM.
- View is the product of \$compile merging the HTML template with \$scope.
- In Angular, templates are written with HTML that contains Angular-specific elements and attributes. Angular combines the template with information from the model and controller to render the dynamic view that a user sees in the browser

Classic One-Way Data Binding

```
graph TD; Template[Template] --> View[View]; Model[Model] --> View;
```

The diagram shows a central green box labeled "View". Two arrows point upwards from a red box labeled "Template" and a blue box labeled "Model" towards the "View" box. A dashed arrow labeled "one-time merge" points downwards from the "View" box back to the "Template" and "Model" boxes.

Source : AngularJS.org

AngularJs Two-way Data Binding

```
graph TD; Template[Template] -- Compile --> View[View]; View[View] <-- Continuous Updates --> Model[Model]; Model[Model] -- Change to Model updates View --> View[View];
```

The diagram shows a red box labeled "Template" with an arrow pointing down to a green box labeled "View". The "View" box has a curved arrow pointing back up to the "Template" box, labeled "Compile". Below the "View" box is a blue box labeled "Model". There are two curved arrows between the "View" and "Model" boxes: one pointing left labeled "Change to View updates Model" and one pointing right labeled "Change to Model updates View". A dashed arrow labeled "Continuous Updates Model is Single-Source-of-Truth" points from the "View" box back to the "Model" box.

February 15, 2015 | Proprietary and Confidential | 29 |

IGATE
Speed. Agility. Imagination

Most other templating systems consume a static string template and combine it with data, resulting in a new string. The resulting text is then innerHTMLed into an element. This means that any changes to the data need to be re-merged with the template and then innerHTMLed into the DOM.

Angular is different. The Angular compiler consumes the DOM, not string templates. The result is a linking function, which when combined with a scope model results in a live view. The view and scope model bindings are transparent. The developer does not need to make any special calls to update the view. And because innerHTML is not used, you won't accidentally clobber user input. Furthermore, Angular directives can contain not just text bindings, but behavioral constructs as well.

The Angular approach produces a stable DOM. The DOM element instance bound to a model item instance does not change for the lifetime of the binding. This means that the code can get hold of the elements and register event handlers and know that the reference will not be destroyed by template data merge.

Instructor Notes:

Add instructor notes here.

Demo

- [AngularJs-MVC](#)



Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJS Modules

- A module is the overall container used to group AngularJS code. It consists of compiled services, directives, views controllers, etc.
- Module is like a main method that instantiates and wires together the different parts of the application.
- Modules declaratively specify how an application should be bootstrapped.
- The Angular module API allows us to declare a module using the angular.module() API method.
- When declaring a module, we need to pass two parameters to the method. The first is the name of the module we are creating. The second is the list of dependencies, otherwise known as injectables.
 - angular.module('myApp', []); // setter method for defining Angular Module.
 - angular.module('myApp'); // getter method for referencing Angular Module.

February 15, 2015 | Proprietary and Confidential | + 31 +



Advantages of 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

When writing large applications, we can create several different modules to contain our logic. Creating a module for each piece of functionality gives us the advantage of isolation to write and test.

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJs Expressions

- Expressions {{expression}} are JavaScript like code snippets.
- In Angular, expressions are evaluated against a scope object.
- AngularJS let us to execute expressions directly within our HTML pages.
- Expressions are generally placed inside a binding and typically it has variable names set in the scope object.
- Expression can also hold computational codes like {{3 * 3}}, but we cannot directly use JavaScript syntax like {{Math.random()}}, conditionals, loops or exceptions inside it.

```
<div>{{3 * 3}}</div> returns 9
```

```
<div>{{'Karthik'+' '+'Muthukrishnan'}}</div> returns Karthik Muthukrishnan
```

```
<div>{{['Ganesh','Abishek','Karthik','Anil'][2]}}</div> returns Karthik
```

February 15, 2015

Proprietary and Confidential

+ 32 -

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

\$rootScope

- When Angular starts to run and generate the view, it will create a binding from the root ng-app element to the \$rootScope.
- \$rootScope is the eventual parent of all \$scope objects and it is set when the module initializes via run method.
- The \$rootScope object is the closest object we have to the global context in an Angular app. It's a bad idea to attach too much logic to this global context.

```
<div ng-app="myApp">    <h1>{{companyName}}</h1>  </div>
<script>
  var app= angular.module("myApp",[]);
  app.run(function($rootScope){
    $rootScope.companyName = "IGATE";
    $rootScope.printCompanyName = function(){
      console.log($rootScope.companyName);
    }
  });
</script>
```

February 15, 2015 | Proprietary and Confidential | + 33 +



Speed. Agility. Imagination

Run blocks - get executed after the injector is created and are used to kickstart the application. Only instances and constants can be injected into run blocks. This is to prevent further system configuration during application run time.

Run blocks are the closest thing in Angular to the main method. It is executed after all of the service have been configured and the injector has been created.

Variables set at the root-scope are available to the controller scope via prototypical inheritance.

```
<div ng-app="myApp" ng-controller="myCntrl">
  {{company}}
</div>

var app= angular.module("myApp",[]);
app.run(function($rootScope){
  $rootScope.companyName = "Microsoft";
  $rootScope.printCompanyName = function() {
    console.log($rootScope.companyName);
  }
});

app.controller("myCntrl",function($scope,$rootScope){
  console.debug($rootScope.companyName);
  $rootScope.companyName= "IGATE"; //Changing Value
  $scope.company = $rootScope.companyName;
});
```

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

Steps for Coding Hello World in AngularJs

- Step 1: Declare the module
- Step 2: Declare the controller and set the properties (or) function to the scope.
- Step 3: Bootstrap angularjs using ng-app and define the controller, so that the properties which we have set in the controller can be consumed in the view(HTML).

```
<html ng-app="sampleApp">  Step - 3
<head>
<script type="text/javascript" src="angular.js"></script>
<script type="text/javascript">
  angular.module('sampleApp',[])
    .controller('SampleController',function($scope){
      $scope.greet = "Hello World";  Step - 2
    });
</script>
<head>
<body>
<div ng-controller="SampleController">  Step - 3
  <h2>{{greet}}</h2>
</div>
</body>
</html>
```

February 15, 2015

Proprietary and Confidential

* 34 *

IGATE
Speed.Agency.Imagination

Instructor Notes:

Add instructor notes here.

Demo

➤ AngularJs-Modules



IGATE
Speed. Agility. Imagination

February 15, 2015

Proprietary and Confidential

• 35 •

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

Dependency Injection

- Dependency injection is a design pattern that allows for the removal of hardcoded dependencies, thus making it possible to remove or change them at run time.

```
function Foo(object){  
    this.object = object;  
}  
Foo.prototype.showDetails = function(data){  
    this.object.display(data);  
}  
  
var greeter = {  
    display : function(msg){  
        alert(msg);  
    }  
}  
  
var foo = new Foo(greeter);  
foo.showDetails("IGATE");
```

February 15, 2015

Proprietary and Confidential

- 36 -

IGATE

Speed. Agility. Imagination

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

Dependency Injection

- At runtime, the Foo doesn't care how it gets the dependency, so long as it gets it. In order to get that dependency instance into Foo, the creator of Foo is responsible for passing in the Foo dependencies when it's created.
- This ability to modify dependencies at run time allows us to create isolated environments that are ideal for testing. We can replace real objects in production environments with mocked ones for testing environments.
- In AngularJS at run time, an injector will create instances of the dependencies and pass them along to the dependent consumer.

Instructor Notes:

Add instructor notes here.

Demo

➤ Dependency Demo



IGATE
Speed. Agility. Imagination

February 15, 2015

Proprietary and Confidential

- 38 -

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

AngularJS Services

- Services provide a method for us to keep data around for the lifetime of the app and communicate across controllers in a consistent manner.
- Services are singleton objects that are instantiated only once per app (by the \$injector) and lazyloaded (created only when necessary).
- We need to put our business logic in Services.

We will discuss in detail about Services, later in this course.

February 15, 2015

Proprietary and Confidential

• 39 •



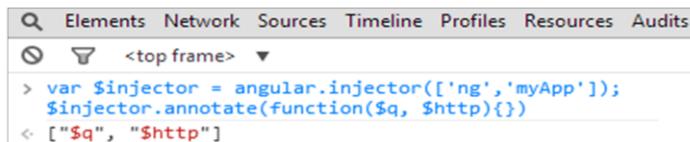
Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction
injector Service

- Angular uses the injector for managing lookups and instantiation of dependencies. We will very rarely work directly with injector service
- injector is responsible for handling all instantiations of our Angular components, including app modules, directives, controllers, etc.
- injector is responsible for instantiating the instance of the object and passing in any of its required dependencies. Injector API has following methods.
- **annotate()**

- The annotate() function returns an array of service names that are to be injected into the function when instantiated.



The screenshot shows the Chrome DevTools Elements tab. It displays a code snippet within the browser's developer console. The code is as follows:

```
<top frame>
var $injector = angular.injector(['ng','myApp']);
$injector.annotate(function($q, $http){});
$injector.annotate(function($q, $http){});
```

February 15, 2015 | Proprietary and Confidential | - 40 -

Instructor Notes:

Add instructor notes here.

**1.3: AngularJS Introduction
injector Service****➤ get()**

- The get() method returns an instance of the service which takes the name argument. (the name of the instance we want to get).

```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
> var $injector = angular.injector(['ng', 'myApp']);
  var http = $injector.get('$http');
```

➤ has()

- The has() method returns true if the injector knows that a service exists in its registry and false if it does not.

```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
> var $injector = angular.injector(['ng', 'myApp']);
  var http = $injector.get('$http');
  $injector.has('$http');
<- true
```

Instructor Notes:

Add instructor notes here.

**1.3: AngularJS Introduction
injector Service****➤ instantiate()**

- The instantiate() method creates a new instance of the JavaScript type. It takes a constructor and invokes the new operator with all of the arguments specified.

A screenshot of a browser's developer tools console tab labeled "Console". The console shows the following code execution:
> var \$injector = angular.injector(['ng','myApp']);
> \$injector.instantiate(function(\$http){});
< Constructor {}

➤ invoke()

- The invoke() method invokes the method and adds the method arguments from the \$injector. The invoke() method returns the value that the fn function returns

A screenshot of a browser's developer tools console tab labeled "Console". The console shows the following code execution:
> var \$injector = angular.injector(['ng','myApp']);
> \$injector.invoke(function(){console.log('Invoked')});
Invoked

February 15, 2015 | Proprietary and Confidential | - 42 -

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

How Angular uses injector Service

The screenshot shows a browser window titled "injector Demo". The address bar shows "file:///D:/Karthik/AngularJS-Firstlook/Lesson01/Angular-Using-Inj...". The main content area displays the following JavaScript code:

```
// Load the app with the injector
var $injector = angular.injector(['ng','myApp'])

// Loading the $controller service with the injector
var $controller = $injector.get('$controller')

// Loading the controller, passing in a scope this is how angular does it at runtime

var $rootScope = $injector.get('$rootScope')
$rootScope.globalVariable = "Root Scope variable value from Chrome Dev Tools";
var scope = $injector.get('$rootScope').$new()
scope.scopeVariable = "Scope variable value from Dev tools";
var MyController = $controller('MyController', {$scope :scope});
```

Below the code, there are tabs for "Console", "Search", "Emulation", and "Rendering".

February 15, 2015 | Proprietary and Confidential | - 43 -

IGATE
Speed.Agency.Imagination

```
// Load the app with the injector
var $injector = angular.injector(['ng','myApp'])

// Loading the $controller service with the injector
var $controller = $injector.get('$controller')

// Loading the controller, passing in a scope this is how angular does
it at runtime

var $rootScope = $injector.get('$rootScope')
$rootScope.globalVariable = "Root Scope variable value from
Chrome Dev Tools";
var scope = $injector.get('$rootScope').$new()
scope.scopeVariable = "Scope variable value from Dev tools";
var MyController = $controller('MyController', {$scope :scope});
```

Instructor Notes:

Run this demo by typing the contents shown in Previous slide using Chrome Developer tools

Demo

- [Angular-Using-Injector](#)
- [Injector-Demo](#)



Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

Config and Run Method

- **angular.Module type has config() and run() method**
- **config(configFn)**
 - We can use this method to register the work which needs to be performed on module loading.
 - It will be very useful for configuring the service.
- **run(initializationFn)**
 - We can use this method to register the work which needs to be performed when the injector is done loading all modules.

Instructor Notes:

Run this demo by typing the contents shown in Previous slide using Chrome Developer tools

Demo

➤ Config-Demo



February 15, 2015 | Proprietary and Confidential | - 46 -

IGATE
Speed. Agility. Imagination

Instructor Notes:

Add instructor notes here.

1.3: AngularJS Introduction

jqLite

- Angular.js comes with a simple compatible implementation of jQuery called jqLite
- Angular doesn't depend on jQuery. In order to keep Angular small, Angular implements only a subset of the selectors in jqLite, so error will occurs when a jqLite instance is invoked with a selector other than this subset.
- We can include a full version of jQuery, which Angular will automatically use. So that all the selectors will be available.
- If jQuery is available, angular.element is an alias for the jQuery function. If jQuery is not available, angular.element delegates to Angular's built-in subset of jQuery, called "jQuery lite" or "jqLite."
- All element references in Angular are always wrapped with jQuery or jqLite; they are never raw DOM references.

February 15, 2015 | Proprietary and Confidential | - 47 -

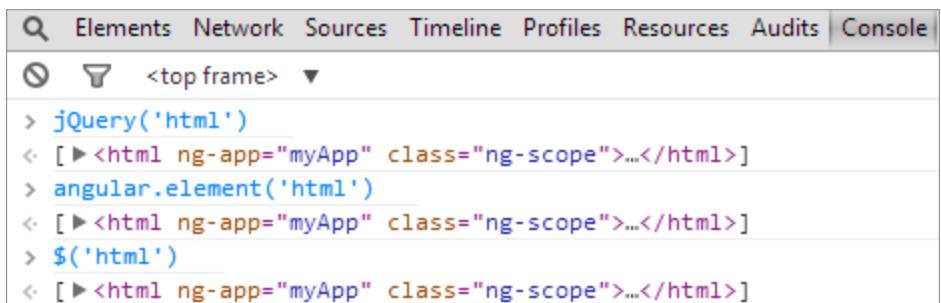


Angular's jqLite

jqLite provides only the following jQuery methods:

addClass(), after(), append(), attrclone(), contents(),
css(), data(), detach(), empty(), eq(), hasClass(),
html(), prepend(), prop(), ready(), remove(),
removeAttr(), removeClass(), removeData(), replaceWith(),
text(), toggleClass(), val() & wrap()

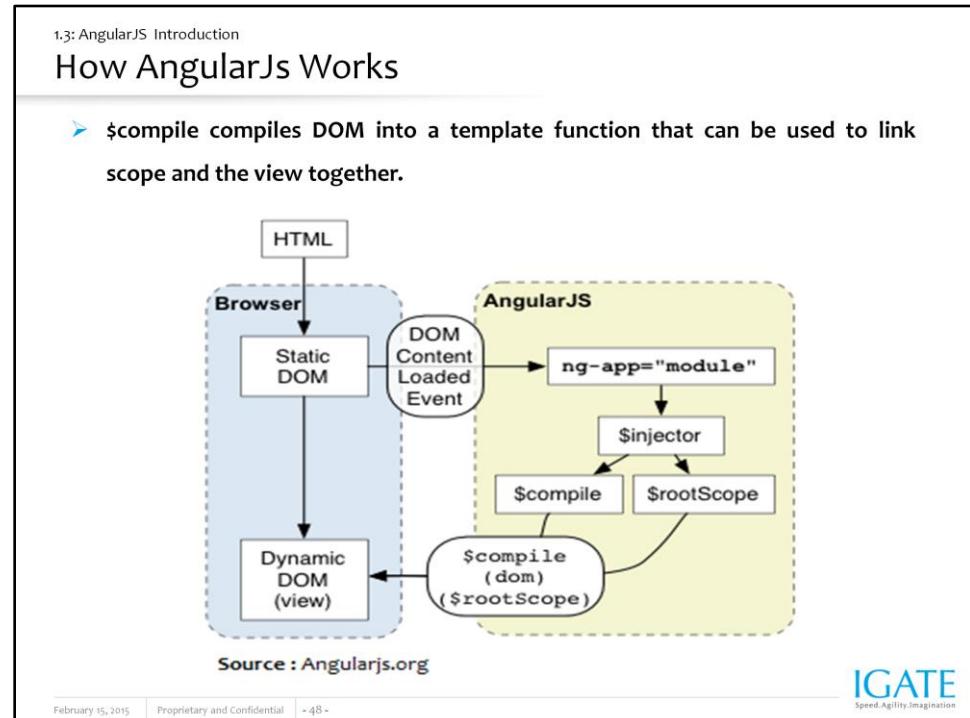
children(), parent(), next() - Does not support selectors,
find() - Limited to lookups by tag name,
bind() - Does not support namespaces, selectors or eventData,
on() - Does not support namespaces, selectors or eventData,
off() - Does not support namespaces or selectors,
one() - Does not support namespaces or selectors
unbind() - Does not support namespaces
triggerHandler() - Passes a dummy event object to handlers.



```
Elements Network Sources Timeline Profiles Resources Audits Console
< top frame >
> jQuery('html')
< [▶ <html ng-app="myApp" class="ng-scope">...</html>]
> angular.element('html')
< [▶ <html ng-app="myApp" class="ng-scope">...</html>]
> $('html')
< [▶ <html ng-app="myApp" class="ng-scope">...</html>]
```

Instructor Notes:

Add instructor notes here.



Angular initializes automatically upon DOMContentLoaded event or when the angular.js script is evaluated if at that time document.readyState is set to 'complete'.

At this point Angular looks for the ng-app directive which designates your application root.

If the ng-app directive is found then Angular will:

- Load the module associated with the directive.
- Create the application injector
- Compile the DOM treating the ng-app directive as the root of the compilation.

This allows you to tell it to treat only a portion of the DOM as an Angular application.

Instructor Notes:

Add instructor notes here.

Summary

- JavaScript objects are also associative arrays
- In JavaScript, the inheritance is prototype-based.
- JavaScript treats functions as objects(first-class functions).
- JavaScript function acts as a constructor when we use it together with the new operator.
- Angular thinks of HTML as if it had been designed to build applications instead of documents.
- Angular supports unit tests and end to end tests.
- Controller is the central component in an angular application.



IGATE

Speed.Agency.Imagination

February 15, 2015

Proprietary and Confidential

- 49 -

Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

- Creating the scope is the primary responsibility of the controller.
- \$scope is the glue between Controller and Model
- View can bind to the properties as well as functions on the scope.
- Expressions only supports a subset of JavaScript. We can create an array in expression.
- Using Dependency Injection we can replace real objects in production environments with mocked ones for testing environments



February 15, 2015 | Proprietary and Confidential | • 50 •

IGATE
Speed.Agency.Imagination

Add the notes here.