Getting Started with AngularJS

Akshay Mathur

What is Angular

- Browser-side MVC toolkit
  - For creating Single Page Web Apps
  - With less custom code
- All in one JavaScript framework
  - Encapsulates many concepts within
  - jQuery may not be required
- Resembles more with server side Frameworks

Prerequisites

- Advance JS objects and objects instances
- Model, View, Controller and App objects
- Data Bindings (one-way and two way)
- Client-side templates
- URL routing
- Module definition
- Module dependency
- Asynchronous programing

We already Know

Custom HTML Attributes

- To any HTML node, any arbitrary attribute can be added

```
<div howdy="I am fine"></div>
```

- Browsers simply ignore such attributes
- These attributes can be read by JavaScript

Directives

- Angular uses custom HTML attributes for its use
  - They call it directives
- Angular script reads these directives and acts accordingly
- HTML tags are also used as directives
  - Standard HTML tags with changed behavior
  - Custom HTML tags

ng-app

- The ng-app directive serves two purposes
  - Tells Angular about the root node for the application
  - Assigns witch app object (module) to use

```
<html ng-app="phonecatApp">
<head>
  <script src=”angular.js"></script>
</head>
<body> … …</body>
</html>
```

Creating Angular Module

- All modules (angular core or 3rd party) that should be available to an application must be registered
- The angular.module is a global place for creating, registering and retrieving Angular modules

```
angular.module(name,
                 [requires],
                 configFn);
```

App Object

- App may be used as the top level Angular Module
  - All other objects are defined as member of this object

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

The Phone Catalogue App

- Nexus S
  Fast just got faster with Nexus S.
- Motorola XOOM with Wi-Fi
  The Next, Next Generation tablet.

```
<ul>
    <li>
    <span>Nexus S</span>
    <p> Fast just got
faster with Nexus S. </p>
    </li>
    <li>
    <span>Motorola XOOM
with Wi-Fi</span>
    <p> The Next, Next
Generation tablet. </p>
    </li>
</ul>
```

We already Know

HTML Templates

- When similar HTML needs to be written for different content, templates are used
- Template system allows to fill up data into templates
  - This is done via data binding expressions
- The template system also allows for basic program constructs e.g. loops

We already Know

JavaScript Template System

- Dynamically creates HTML code in JS
  - Data driven HTML
  - Allows variable substitution, looping and conditional statements
- Generated HTML is inserted into the DOM for changing part of the page on-the-fly
- Many libraries are available e.g. Handlebars, DustJS, Mustache, UnderscoreJS etc.
  - Angular has its own template system

We already Know

View

- Object that holds visual representation of the data
- Provides methods for
  - Rendering the user interface
  - Handling the user interaction within the view
- Angular uses template as view
- The view gets its data from its Model
  - Each view has its own model

We already Know

Model

- Object that holds data in a structural form
- Makes data available to view
- Acts as a unit of data
- By default Angular uses $scope object as model
  - Each view has its own model so the scope of $scope is limited to the view

Value Substitution

- Values are passed to template using $scope object
  - Members of $scope can be used directly in template
- Items in {{ }} are treated as variables and replaced by its values

```
<li>
  <span>{{phone_name}}</span>
  <p>{{phone_desc}}</p>
</li>
```

Looping

- Looping is done using ng-repeat directive
- Data passed to ng-repeat should be an Array

```
<li ng-repeat="phone in phones">
  <span>{{phone.name}}</span>
  <p>{{phone.desc}}</p>
</li>
```

Controller

- Object that acts as a glue to connects view and model
- The ng-controller directive attaches controller to the view

```
<ul ng-controller=
"PhoneListCtrl">
   <li> … …</li>
</ul>
```

Passing Data to View

- Controller method of the App Object creates a controller
- Value of $scope is set in the controller

```
phonecatApp.controller('PhoneListCtrl',
   function ($scope) {
      $scope.phones = [
         {'name': 'Nexus S',
         ’desc': 'Fast just got faster'},
         {'name': 'Motorola XOOM',        ’desc': ’Next
Generation tablet.’}
      ];
   }
);
```

Everything Together: HTML

```
<html ng-app="phonecatApp">
<head>
   <script src=”angular.js"></script>
</head>
<body>
   <ul ng-controller="PhoneListCtrl">
      <li ng-repeat="phone in phones">
         <span>{{phone.name}}</span>
         <p>{{phone.desc}}</p>
      </li>
   </ul>
</body>
</html>
```

Everything Together: JS

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

phonecatApp.controller('PhoneListCtrl',
  function ($scope) {
     $scope.phones = [
        {'name': 'Nexus S',
        ’desc': 'Fast just got faster'},
        {'name': 'Motorola XOOM',
        ’desc': ’Next Generation tablet.’}
     ];
  }
);
```

Reading from Form Elements

- A model can be attached to form elements
  - AngularJS updates the attached Model as value in form element changes

```
<input ng-model="query">
```

  - AngularJS also updates the value of form element changes when the attached model is changed

Data Bindings

- Corresponding changes trigger as soon as data changes at one place
- One way data binding
  - Template re-renders as data in $scope changes
- Two way data binding
  - Value of form element and attached model always remain in sync

Modifying Data

- AngularJS provides a few readymade filter functions for achieving certain effect
  - Can be included within expressions in the template

```
var val = 54.2
{{ val | number : 3}} // 54.200
```

- Option to write custom filters is available

Formatting Filters

- Number: Formats number
  ```
  {{ val | number : 3}}
  ```
- Currency: Puts a currency identifier
  ```
  {{amount | currency: "USD"}}
  ```
- Date: Formats date
  ```
  {{today | date: 'medium'}}
  ```
- Lowercase/Uppercase
  ```
  {{’somestr’| uppercase}}
  ```
- JSON: Formats JS object as string
  ```
  {{ {'name':'value'} | json }}
  ```

Filtering Arrays

- Limit: Picks limited number of items

```
{{[1,2,3,4,5,6,7]| limitTo: 3 }}
```

- Filter: Picks items based on condition

```
{{['Bob', 'Mike'] | filter:
'm'}}
```

- Order: Orders the array of objects in a field

```
{{[o1, o2] | orderBy: o1.f1}}
```

Filtering Repeater

```
<body>
   Search: <input ng-model="query">
   Sort by:
      <select ng-model="orderProp">
         <option value="name">Alphabetical</option>
         <option value="age">Newest</option>
      </select>

   <ul class="phones">
      <li ng-repeat="phone in phones | filter: query |
                                orderBy: orderProp">
         {{phone.name}}
         <p>{{phone.snippet}}</p>
      </li>
   </ul>
</body>
```

Creating Filter

- Custom filter cam be created using filter method of Angular module

```
angular.module(
             'phonecatFilters', []
             ).filter('status',
                       filter_func);
```

We already Know

Function Facts

- A function can be assigned to a variable
- A function can be defined within another function
- A function can return a function

```
function sqr(){
  sq = function (x){
      return x * x * x;
  };
  return sq;
}
My_sqr = sqr(); // function
My_sqr(3);      // 27
```

Custom Filter

- Filter returns a function
  - This function takes the value to be transformed as input
  - Optionally other arguments can be taken
  - Returns the transformed value as output

```
filter_func = function(){
   return function(input){
         return input ? ‘smart’: ‘dumb’
   }
}
```

Complete Filter

Definition:

```
angular.module(
      'phonecatFilters',
      []).filter('status', filter_func);

filter_func = function(){
         return function(input){
            return input ? 'smart': 'dumb';
      }
});
```

Usage:

```
{{ phone_type | status }}
```

Question

- The filter is defined as a member of top level Angular module named phonecatFilters

```
angular.module(
    'phonecatFilters',
    []).filter('status', filter_func);
```

- How will this be available to the HTML template connected to phonecatApp?

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

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

Dependency Injection

We already Know

Object as Argument

- Objects can be passed to a function as an argument
- They proxy for named arguments

```
Say_hello = function (options){
  if (typeof options === ‘Object’){
    options.msg = (options.msg)?
        options.msg : ’Hello!’;
  }
  alert(options.msg + ‘ ‘ + options.name);
}

Say_hello({name: ‘Akshay’});
```

We already Know

Using Functions as Objects

- Functions are actually First Class objects

So we can change

```
User= {}
User.name = 'Akshay'
User.greet = function(){
   alert('Hello ' + User.name)
}
```

to

```
User = function(){
   this.name = 'Akshay'
   this.greet = function(){
      alert('Hello ' + this.name)
   }
}
```

We already Know

Creating Instances

- Now the object instance can be created using new keyword

```
 user1 = new User;  user1.name
//Akshay user1.greet() //Hello
Akshay
```

We already Know

Class Constructor

- The main object function can take arguments for initializing properties

```
User = function(name){
  this.name = name;
  this.greet = function(){
    alert('Hello ' + this.name)
  }
}
user1 = new User('Cerri');
user1.greet()   //Hello Cerri
```

We already Know

Extending a Class

- More functions and properties can be defined extending the prototype of the class

```
User.prototype.setGender =
   function(gender){
      this.gender = gender;
   };

User.prototype.sayGender =
   function(){
      alert(this.name + ‘ is ’ +
               this.gender);
   };
```

What is Dependency Injection

- A software design pattern that deals with how code gets hold of its dependencies
- The best option is that the dependency is passed in to the function or the object where it is needed

Passing Dependency

- If the dependency is simply handed to the component, it removes the responsibility of locating the dependency

```
function SomeClass(greeter) {
  this.greeter = greeter;
}

SomeClass.prototype.doSomething =
  function(name) {
    this.greeter.greet(name);
  }
```

Injector

- To manage the responsibility of dependency creation, each Angular application has an injector.
- The injector is a ‘service locator’ that is responsible for construction and lookup of dependencies.
- How does the injector know what service needs to be injected?

How injector knows..

- Infer the name of dependencies from the name of function arguments

```
function MyCtlr($scope,
                        greeter){}
```

- Pass to injector

```
MyCtlr.$inject = ['$scope',
                  'greeter']
```

Making the Filter Available

- While creating a module, other required modules can be passed as dependency list

```
var phonecatApp =
    angular.module('phonecatApp',
        [‘phonecatFilters’]);
```

```
{{ phone_type | status }}
```

Services

Angular Services

- What is a service?
  - A system doing work for us
- Angular services are substitutable objects
  - Wired together using dependency injection
- Angular services are:
  - Lazily instantiated
  - Singletons

Registering Custom Service

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

myMod.factory('serviceId’,  function() {
     var shinyNewServiceInstance;
     //factory function body that constructs 
shinyNewServiceInstance
     return shinyNewServiceInstance;
});
```

Built-in Services

- Angular provides built-in services for various needs
  - $filter: for formatting the data
  - $window: for accessing window object
  - $location: for parsing URL
  - $timeout: a wrapper on setTimeout
  - $http: for communicating with server using XHR or JSONP
  - …

Talking to Server

We already Know

Asynchronous JavaScript & XML

- A way in web browser to communicate with server without reloading the page
- XmlHttpRequest (XHR) object can also create HTTP request and receive response
  - The request happens asynchronously
    - Other operations on page are allowed during the request
  - Received data does not render automatically
    - Data needs to be received in a callback function and used programmatically

We already Know

AJAX Call in jQuery

```
$.ajax({
   url: '/my_ajax_target',
   type: 'POST',
   data: {id: 2},
   success: function(response){
         alert('Hello! ' + response.name);
      },
   error: function(){
      alert('Please try later');
      }
});
```

AJAX Call in Angular

```
$http({
    method: 'GET',
    url: '/someUrl’,
    params: {id: 2}
    }
).success(function(data, status, headers, config) {
        // this callback will be called asynchronously
        // when the response is available
    }
).error(function(data, status, headers, config) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    }
);
```

Shortcut Methods

- For HTTP methods:
  - GET: `$http.get('/someUrl')`
  - POST: `$http.post('/someUrl', data)`
  - PUT: `$http.put`
  - DELETE: `$http.delete`
- For getting only headers
  - `$http.head`
- For cross-domain JSON (JSONP) call
  - `$http.jsonp('http://domain/Url')`