Essentialist Software Design

Web Development + Product Management | Glenn Stovall

AngularJS: An Overview

Jun 27th, 2013

AngularJS is a JavaScript framework made by Google for building complex client-side applications. Angular's killer feature is 'directives' which allow you to extend HTML by creating your own tags and attributes. Angular projects have a somewhat different structure than other JavaScript MVC frameworks, but it can be highly modular and easy to maintain once you understand the structure. Let's take a look at the main components of AngularJS and how they work, and why you should strongly consider Angular for your next project.

The Philosophy of AngularJS

Data First

If you are familiar with JavaScript libraries like jQuery, using AngularJS will require a bit of a paradigm shift. AngularJS is like jQuery, but backwards. what I mean by that, is that jQuery is mainly focused on DOM manipulation, and then you update data based on that. When writing Angular, you will mostly be updating data, and the DOM will be updated for you, with very little work on your end.

Highly Testable

AngularJS was designed to be testable end-to-end. It uses Dependency Injection In many of objects so that parts of it can easily be mocked. It is also designed to encourage you to break the functionality of your application into several smaller, more testable parts.

Declarative HTML

Angular is designed to extend HTML to make it the language you need for building complex web applications. Adding your own tags and attributes allows you to write simple HTML tags that do very complex things.

Data Binding

Let’ s take a look at a very simple AngularJS app, that uses two-way data binding.

index.html

```
<body ng-app>
  <div>
    <input type='text' ng-model='name' />
    <h2>{{name}}</h2>
  </div>
</body>
```

Let’ s break this down line by line to explain what is happening:

- `<body ng-app>` : all Angular code must be wrapper in this directive. This declares that everything within this tag will be treated as an angular application.
- `<input type='text' ng-model='name' />` : This is one part of the data binding. Here we are using another directive, ng-model, to bind the input to a string. Note that in this example we also see that attribute directives can have arguments.
- `<h2>{{name}}</h2>` : Whenever you type into the input box, this h2 tag updates automatically. This is the automatic DOM manipulation I mentioned earlier. We were able to do this without writing a single line of JavaScript.

Modules

Modules are used to organize the objects in an AngularJS application. Modules can either be used as as the ‘core’ of an application, and contain all of the classes used for it, or they can be used to group several objects that have similar functionality. Let’ s create a module for our application:

app.js

```
app = angular.module('myApp', []);
```

index.html

```
<body ng-app='myApp'>
```

The first argument passed to `angular.module` is the name of the module. We have passed that name to the `ng-app` to bind everything contained in that directive to our module. The second argument is an array of other modules your module depends on. This is empty since we currently don’ t have any. Much like directives, AngularJS contains several modules already built that you can include in your projects.

Now that we have a module, we will write our first type of AngularJS object, a controller.

Controllers

Controllers are tied to particular html elements. They contain data and functions that the html can interact with, and can interact with other service objects, which can handle things such as communicating with the server. Let’ s create a controller and bind it to a div in our application.

app.js

```
app.controller('mainCtrl', function($scope){
  $scope.name = 'Default Name';
});
```

index.html

```
<body ng-app='myApp'>
  <div ng-controller='mainCtrl'>
    <input type='text' ng-model='name' />
    <h2>{{name}}</h2>
  </div>
</body>
```

Here we’ ve used the `ng-controller` directive to bind our controller function to a div. Controllers get one argument by default which is called `$scope`. `$scope` contains all of the data that the html can interact with. When you reload the page, you will notice that the input now starts with the field filled in with ‘Default Name’ . This is because we set that variable in the controller, which is called when the page is loaded.

Functions can also applied to scope and then called inside the HTML. Angular contains several directives for handling various events. Let’ s change our example to include a function that we will call with an `ng-click` directive. We will add a button to save the name we create and later display the list.

app.js

```
app.controller('mainCtrl', function($scope){
  $scope.name = 'Default Name';
  $scope.people = [];
  $scope.savePerson = function() {
    $scope.people.push($scope.name);
    $scope.name = '';
  };
});
```

index.html

```
<body ng-app='myApp'>
  <div ng-controller='mainCtrl'>
    <input type='text' ng-model='name' />
    <button ng-click='savePerson()'>Save Person</button>
    <h2>{{name}}</h2>
  </div>
</body>
```

Now we can save a list of these people. We can use another directive called `ng-repeat` To output a list of the people we have saved:

index.html

```
<ul>
  <li ng-repeat='person in people'>{{person}}</li>
</ul>
```

Now that we have seen how we can manipulate data inside the JavaScript, let's look at how we can manipulate the DOM, by creating our own directives

Directives

You have already seen directives in use in several places inside the application. You can also define your own directives as part of your application. Let's create an alertable directive that will allow you to set a message to be alerted whenever you click on the element.

app.js

```
app.directive('alertable', function(){
  return {
    restrict : 'A',
    link: function(scope, element, attrs) {
      element.bind('click', function() {
        alert(attrs.alertable);
      });
     }
  };
});
```

And now we an add this to our person list:

index.html

```
<li ng-repeat='person in people'>
  <span alertable='{{person}}'>{{person}}</span>
</li>
```

As you can see, directives return an object that will define the directive. There are several optional arguments that can be passed here, but let' s look at the two we used here:

- restrict : This tells your directive what kind of directive it will be. restrict is required and there are four possible arguments that can be passed to it:
- E : Element. example usage: `<my-directive></my-directive>`
- A : Attribute. example usage: `<div my-directive></div>`
- C : Class. example usage: `<div class='my-directive'></div>`
- M : Comment. example usage: `<!-- directive:my-directive -->`
- link : The link function is responsible for adding event listeners and updating the dom.

You can read about some of the other possible options in the Angular directive documentation.

Services

Services are classes that can either contain business logic or handle data. Now we are going to refactor our previous example and create a service that can handle the data in our list of people.

app.js

```
app.factory('PersonService', function() {
  var PersonService = {};
  PersonService.people = [];
  PersonService.addPerson = function(person) {
    PersonService.people.push(person);
  };
  return PersonService;
});
```

You' ll notice that services are created slightly differently than controllers. Controllers are just a function, where services are a function that returns an object. This allows you to add private methods to your service if you would like.

Now we need a way for our controller to have access to the service we just created. Since they are in the same module, this is very easy to do. This also involves one of the more "magical" features of Angular. To give the controller access to our service, we just have to add it as an argument to our controller, like so:

```
app.controller('mainCtrl', function($scope, PersonService){});
```

What' s really surprising is that if you change the order of the arguments to the controller function, it will not change how the controller function operates. So, if we did this instead:

```
app.controller('mainCtrl', function(PersonService, $scope){});
```

Our function would behave in exactly the same way. This is because AngularJS looks at the names of the arguments of a function to decide what to pass to them. Now, you may be concerned that this could break if you were to minify your JavaScript, and you would be right. Luckily, Angular provides an alternate way of declaring functions to prevent issues when minifying your code. It involves passing an array of strings to tell angular what the arguments should be, with the actual function as the last element in the array. So now your controller would look like this:

```
app.controller('mainCtrl', ['$scope', 'PersonService', function($scope, PersonService){}]);
```

When writing services, Angular provides some helpful modules for communicating with the server. Some of the most useful are ngHttp and ngResource. ngHttp is used to make HTTP requests. ngResource is an extension of ngHttp designed to work exclusively with REST APIs. This is another reason why when you are declaring a service, you are asked for a function that returns an object; We are able to use this function to extend an existing object like ngResource and return it.

Routing

Angular also provides support for routing with URLs. Routing is performed by using the `config` function of the module. Let' s use this to structure separate pages for our main page, and give each person in our person list a profile page. Here is what the route provider will look like:

app.js

```
app.config(function($routeProvider){
  $routeProvider.when('/', {
    templateUrl: 'templates/home.html',
    controller: 'homePageCtrl'
  });
  $routeProvider.when('/person/:id', {
    templateUrl: 'templates/profile.html',
    controller: 'profileCtrl'
  });
});
```

Filters

Filters are a smaller, but often useful part of AngularJS. Filters are used to transform data when it is displayed to the user. Like directives, you can build them yourself, and Angular also includes several useful ones built-in. Let' s look at the following HTML:

```
<span>Price:</span> 300
```

Which will look like this in the browser:

```
Price: 300
```

We can apply a couple filters to this to make it look different, by using the | operator:

```
<span>{{ "Price:" | uppercase }}</span> {{ "300" | currency }}
```

And our output will now look like this:

```
PRICE: $300.00
```

Additional Reading

I hope you found this helpful, but this is just scratching the surface of what Angular.js is capable of. If you would like to learn more about Angular, here are some resources for further reading:

- AngularJS Documentation - official documentation, lots of good examples
- EggHead.io - several short (usually 4 or 5 minute) videos explaining several particular parts of angular.
- BuiltWith AngularJS - examples of applications built with Angular
- A Comparison of Angular, Backbone, CanJS and Ember - a great article comparing AngularJS to some

Posted by Glenn Stovall Jun 27th, 2013 angularjs, javascript, programming