Dependency Injection & Filters

What's Hot?

Who's excited about what right now?
Async fails: https://twitter.com/jonathansampson/status/676487374495342592/photo/1

Review Yesterday

Angular Routing (ngRoute, ui-router, parameters)

Today

DI & Filters

Dependency Injection

We know that keeping code modular is good (encapsulation). But if we're going to separate code out into various small objects, those components are going to depend on other objects to perform their various functions. How do we make those dependencies available to use in our code? The primary method AngularJS uses to solve this problem is **dependency injection**: **we pass in dependencies where they are needed**.

```
angular.module('navController', [])
   .controller('navController', navController);
navController.$inject = ['$location'];

function navController($location)
{
   var nav = this;
   nav.isActive = isActive;

   function isActive(viewLocation) {
       return viewLocation === $location.path();
   }
}
```

Note that our controller is not concerned with the implementation or location of the **$location** object. It is simply provided with the **$location** object when it is created, and is free to use it.

DI is useful because:

1. The dependency injector ensures that all dependencies are instantiated before we use them.
2. It helps us simplify our code by keeping each piece modular. We can put various tasks into service modules, then inject those services into controllers and directives where they are needed.

From the official documentation, we have three ways of using a dependency in our module:

1. We can create the dependency, typically using the "new" operator.
2. We can look up the dependency, by referring to a global variable.
3. We can have the dependency passed in where it is needed.

Dependency injection represents the third choice in this list.

Option 1 is not preferred because now the dependency is directly coupled to the component via code, whereas with dependency injection we can instantiate a component with variable dependencies (which is critical for testing, where dependencies are often mocked).

Option 2 is highly problematic--global state should be avoided. It becomes difficult to track down which piece of the application modified global state as your app grows.

DI Syntax

Dependencies can be injected into services, directives, filters, controllers, providers, etc. Dependencies that are injectable include services, filters, providers, etc. Angular Doc.

```
angular.module('myModule', [])
.factory('serviceId', ['depService', function(depService) {
   // ...
}])
.directive('directiveName', ['depService', function(depService) {
   // ...
}])
.filter('filterName', ['depService', function(depService) {
   // ...
}])
```

```
.config(['depProvider', function(depProvider) {
   // ...
}])
.run(['depService', function(depService) {
   // ...
}])
.controller('MyController', ['dep1', 'dep2',
   function(dep1, dep2) {
      // ...
}]);
```

Dependency Annotation

In the above example, you'll notice that all dependencies are declared twice--once as a string, and once as a variable reference. **The string version is used to preserve functionality when the code is minified.** Minification will change dependency variable names to shorter strings, such as "a", "b", "c", etc. Because dependency injection happens at runtime, and because the dependencies don't exist in the scope of the component, the real name needs to be available. String literals are not minified (it would make no sense to change a string literal that might be displayed to the user, for example), so they preserve this ability.

Note that it is preferred per the John Papa Style Guide to use the $inject property to annotate components rather than using lists of strings as shown above. The official docs recommend using lists of strings, but it creates long, difficult-to-read lists. The recommended method looks like this:

```
angular.module('navController', [])
   .controller('navController', navController);
navController.$inject = ['$location', '$http'];
function navController($location, $http)
{
   // ...
}
```

Strict Mode

Add **ng-strict-di** to the ng-app element if you know your code will be minified. This helps you make sure you either use the list of strings in the component declaration or in the $inject property. If you use the ng-strict-di attribute and forget to inject the string version of your dependencies, Angular will throw an error when you try to load the app.

```
<html ng-app="myApp" ng-strict-di>
```

Built-in Filters

currency - Formats a number as local currency

```
{{ 999.99 | currency }}
$999.99
```

date - Formats a date object using local format

```
{{ dateVariable | date : format : timezone}}

Examples:
     {{1288323623006 | date:'medium'}}
     Oct 28, 2010 9:40:23 PM

     {{1288323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}
     2010-10-28 21:40:23 -0600

     {{'1288323623006' | date:'MM/dd/yyyy @ h:mma'}}
     10/28/2010 @ 9:40PM

     {{'1288323623006' | date:"MM/dd/yyyy 'at' h:mma"}}
     10/28/2010 at 9:40PM
```

filter - Selects a subset of items from an array of items and returns a new array. Example from official docs:

```
<input ng-model="searchText">
<table>
<tr ng-repeat="friend in friends | filter:searchText">
    <td>{{friend.name}}</td>
    <td>{{friend.phone}}</td>
</tr>
</table>

friends = [{name:'John', phone:'555-1276'},
           {name:'Mike', phone:'555-4321'},
            {name:'Adam', phone:'555-5678'},
           {name:'Julie', phone:'555-8765'},
           {name:'Juliette', phone:'555-5678'}];
```

The table will contain those friends which contain text that matches the text typed into the input. By default, all properties are searched. You can change the filter to "filter:searchText.name" to only search the name property, for example.

json - Takes JSON or an object and outputs a JSON string.

```
{{ {'Author': 'Tolkien', 'Works': ['The Hobbit', 'The
Silmarillion']} | json }}
```

limitTo - Creates a new array or string containing only the specified number of elements. Useful with ng-repeats that repeat over data sets too large to display.

```
{{ "This string is far too long" | limitTo:7 }}
"This st"

{{ [1,2,3,4,5,6,7,8,9] | limitTo:7 }}
[1,2,3,4,5,6,7]
```

lowercase and **uppercase** - Converts string to lowercase or uppercase.

```
{{ "This stRing is FAr tOo InconSisTent" | lowercase }}
"this string is far too inconsistent"

{{ "This stRing is FAr tOo InconSisTent" | uppercase }}
"THIS STRING IS FAR TOO INCONSISTENT"
```

number - Formats a number as text.

```
{{ 123456789 | number }}
1,234,567,890

{{ 1.234567 | number:2 }}
1.23
```

orderBy - Orders an array. Consider the same list of friends used above. The resultant table will order the list in descending age (using the friend.age property). Example from official docs:

```
<table>
   <tr ng-repeat="friend in friends | orderBy:'-age'">
     <td>{{friend.name}}</td>
     <td>{{friend.phone}}</td>
     <td>{{friend.age}}</td>
   </tr>
</table>
```

Using Filters in JavaScript

Most commonly, you'll use filters in HTML. All filters can also be used in JavaScript, however. An example from the official docs using orderBy:

```
homeController.$inject = ['$filter'];
function homeController($filter) {
   var hc = this;
   hc.someDate = new Date(1990, 11, 25);
   hc.formattedDate = $filter('date')(hc.someDate, 'yyyy-MM-dd');
   hc.someNumber = 22.5;
   hc.formattedNumber = $filter('number')(hc.someNumber, 2);
}
```

Review

Dependency Injection & Filters

Project

Work on the ToDo app!

Resources

Angular Docs for Dependency Injection
    https://docs.angularjs.org/guide/di
Angular Docs for Filters
    https://docs.angularjs.org/api/ng/filter

Tomorrow

Custom Filters