The Anatomy of an Angular JS app

Pro Angular JS: Chapter 9

Module

- Top level component
- *JS* - angular.module
- *HTML* - ng-app

3 module roles

1. Associate an AngularJS application with a region of an HTML document.

2. To act as a gateway to key AngularJS framework features.

3. To help organize the code and components in an AngularJS application.

Creating a Module

angular.module(‘exampleApp’, []);

- module arguments:
  - name
  - dependencies
  - config

3 categories of module features

1. Define components for an AngularJS application.
2. Make it easier to create components.
3. Manage the AngularJS life cycle.

Module features

- animation(name, factory)
- config(callback)
- constant(key, value)
- controller(name, constructor)
- directive(name, factory)
- factory(name, provider)
- filter(name, factory)
- provider(name, type)
- name
- run(callback)
- service(name, constructor)
- value(name, value)

Controller

- Module component
- *Conduit between the model and the view*
- *HTML* - ng-controller
- *JS* - angular.module(‘exampleApp’).controller(‘controller Name’, function($scope) { });

Fluent API

- Rather than assign variables to the module declaration, you can chain methods to it.

- angular.module(‘exampleApp’, []) is a module constructor

- angular.module(‘exampleApp’) returns the module itself.

Dependency Injection

- *Example* - passing $scope in as a controller param.
- Declare the other features that need to be included inside your component.
- Prevents a lot of global variables / objects.
- Order of dependencies:
  - *params only* — not important
  - *array of dependencies && params* — must match

Controller example

```
angular.module(‘exampleApp’)
    .controller(‘appController’, [‘$rootScope’,
        ‘$scope’, ‘revit.logging.LoggerFactory’,
        ‘dataFactory’,  function($rootScope, $scope,
        loggerFactory, dataFactory) {



}]);
```

Organize Dependencies

Keep your controller's dependencies organized by ordering them by type:

1. AngularJS dependencies ($)
2. Third-party dependencies (revit)
3. Your created dependencies (dataFactory)

Apply Controller to View

<body>

</body>

ControllerAs

- Assigns controller values to object notation
  - controller.value
- Better semblance of a Class based syntax
- Easy to know where values are coming from
- Reduces $scope clutter

Controller example

- http://jsfiddle.net/U3pVM/11306/

Directive

- Most powerful AngularJS feature
  - extend and enhance HTML
  - improves site performance by limiting DOM traversal
  - Implemented using the module.directive method.

Directive example, fixed controller

- http://jsfiddle.net/U3pVM/11332/

- Fixed controller, controllerAs syntax

- Isolate scope on highlight param

Directive Link method

- factory method, runs after the DOM has been compiled
- handles DOM manipulation, two way binding, and reusable methods
- params
  - **scope** - the scope of implementation
  - **element** - jqLite wrapped element from where this directive is invoked
    - remove the need for DOM traversal here
  - **attrs** - returns key/value of the other attributes from the node where this directive is invoked

Directive example, unnamed controller

- http://jsfiddle.net/U3pVM/11333/
- Factory implemented for common data structure
- Isolate scope on highlight AND day params
- Nothing is assumed from the controller's scope

Factory

- A service for worker functions
  - Does not register a service instance
  - Creates a factory function that will instantiate itself when called by other parts of your application.
  - Ideal for doing the work for your controllers, or sharing data across your application

Filters

- Used in views to format the data displayed to the user.
- Ensures consistency in data presentation
- Applied in template expressions
- Use conservatively, as filters are applied every time there's a $scope change, multiple times.

Filter example

- http://jsfiddle.net/U3pVM/11393/

Services

- Services are singleton objects that can be used throughout your application.

- A good example is for making HTTP requests.

- Services are already instantiated, but can still be modified via prototypal inheritance.

- Example can be found in the sample application:

  - https://cdlprttnconfl01.es.ad.adp.com/display/NCP/Sample+Project

Values

- The module.value method lets you create services that return fixed values and objects.

- You can start your application with some initial variables, that can be extended later using decorators.

Value example

- http://jsfiddle.net/U3pVM/11397/

Use modules to organize code

- Module as top level of a component.
- Keep files organized by what component they belong to and what type of angular feature they are.
- Use dependency injection to include the required parts
- When you start to get a lot of different dependency injections into your module/feature, it may be helpful to abstract it one level further and include it into the component's module definition.