Routes and Events

by Vlad Costel Ungureanu
for “Learn Stuff” and “CGM Romania”

Providers

- ✓Providers are syntactically defined as a custom type for creating angular elements
- ✓Providers are responsible for creating objects
- ✓A provider needs to implement the $get functionality
- ✓When you create an Angular element a provider is created automatically to expose that element
- ✓Providers have the following constructor method:
  - ✓Constant
  - ✓Value
  - ✓Service
  - ✓Factory
  - ✓Decorator
  - ✓Provider

Providers

✓A provider can also be used to inject existing services

```
angular.module('myModule', []).config(['$provide', function($provide) {
  $provide.factory('serviceId', function() {
    var shinyNewServiceInstance;
    // factory function body that constructs shinyNewServiceInstance
    return shinyNewServiceInstance;
  });
}]);
```

Route Provider

```
var app = angular.module('myApp', ['ngRoute']);
app.config(function($routeProvider, $locationProvider, $httpProvider) {
        $routeProvider.when('/', {
                template : ‘<div> … </div>',
                controller : 'mainController',
                controllerAs : 'main'
        }).when('/forgot-password/:userEmail', {
              templateUrl : 'forgot-password.html',
              controller : 'forgotPasswordController',
              controllerAs : 'main‘,
              resolve: {  async: ['someService', function(someService) {
                            return someService.fetchAllItems(‘criteria'); }] }
          }).otherwise({
                redirectTo : '/'
        });
```

Route Provider

- ✓Used to make single page applications
- ✓Requires the usage of ng-view directive in main page
- ✓Adds view to page, bounds view to controller and assigns the controller the specified name
- ✓Configuration of route provides is done in the config block which is run when the app is loading
- ✓ Route Provider also allows the injection of route parameters in the controller responsible for a specific view
- ✓Resolve can be used for asynchronous operation to be executed before the route is loaded
- ✓Resolve expressions can be injected in the controller assigned to the route

State Provider

```
app.config(function($stateProvider, $urlRouterProvider) {
   $urlRouterProvider.otherwise('/home');
   $stateProvider
      .state('home', {
         name: 'home',
         url: '/home',
         templateUrl: 'home.html',
         controller: function($scope) { $scope.vals = ['Bernese', 'Husky', 'Goldendoodle']; }
      })
      .state('about', {
            url: '/about',
             controller: 'aboutController'
      });
});
```

State vs Route

- ✓ Routes impose a single view per page and a single associated controller
- ✓ To provide flexibility(elements displayed based on various conditions) when using routes angular directives as ng-include, ng-show or ng-if need to be used
- ✓ When using routes, all routes are considered equals
- ✓ States allow for multiple views, each with their own controller to exist in the same page

```
<div ui-view>
  <div ui-view='header'></div>
  <div ui-view='content'></div>
  <div ui-view='footer'></div>
</div>
```

- ✓ States allow for parent child relationship between routes

UI Router nested views

✓User state of the application to load views, decoupling them from the actual URL

```
$stateProvider.state('home', {
      url: '/home',
      templateUrl: 'partial-home.html'
   }).state('home.list', {
      url: '/list',
      templateUrl: 'partial-home-list.html',
      controller: function($scope) {
         $scope.dogs = ['Bernese', 'Husky', 'Goldendoodle'];
      }
   }).state('home.paragraph', {
      url: '/paragraph',
      template: 'I could sure use a drink right now.'
   })
```

...

UI Router nested views

```
<div class="jumbotron text-center">
    <h1>The Homey Page</h1>
    <p>This page demonstrates <span class="text-danger">nested</span> views.</p>

    <a ui-sref=".list" class="btn btn-primary">List</a>
    <a ui-sref=".paragraph" class="btn btn-danger">Paragraph</a>
</div>

<div ui-view></div>
```

UI Router multiple views

```
<div class="row">

   <!-- COLUMN ONE NAMED VIEW -->
   <div class="col-sm-6">
      <div ui-view="columnOne"></div>
   </div>

   <!-- COLUMN TWO NAMED VIEW -->
   <div class="col-sm-6">
      <div ui-view="columnTwo"></div>
   </div>

</div>
```

UI Router multiple views

```
.state('about', {
    url: '/about',
    views: {
      // the main template will be placed here (relatively named)
      '': { templateUrl: 'partial-about.html' },
      // the child views will be defined here (absolutely named)
      'columnOne@about': { template: 'Look I am a column!' },
      // for column two, we'll define a separate controller
      'columnTwo@about': {
        templateUrl: 'table-data.html',
        controller: ‘someController'
      }
    }
  });
```

Events

✓Using $scope.$emit will fire an event UP the $scope.

✓Using $scope.$broadcast will fire an event DOWN the $scope.

✓Using $scope.$on is how we listen for these events.

✓Using $scope de dispatch event will always miss child scopes

✓To ensure an event is dispatched to scopes with no hierarchical link we can use $rootScope

Broadcast

- ✓Dispatches an event to all child scopes and their child scopes
- ✓Notifies all listeners registered to $rootScope
- ✓Listeners in the origin scope are first to triggers
- ✓Broadcasted events cannot be canceled
- ✓Events broadcasted through $rootScope reach all child scopes

Emit

✓Dispatches an event to all parent scopes and their parent scopes

✓It does not go other child scopes of the parent scopes

✓Notifies all listeners registered to $rootScope

✓ Emitted events can be canceled by a listener to stop propagation

✓Using $rootScope to emit events will only trigger $rootScope listeners

LEARN stuff IO

On

✓Will be triggered by the expected event

✓Emitted events can be canceled through event.stopPropagation();

✓Scope listeners are destroyed alongside the scope

✓ Root scoped listeners need to be destroyed manually using the $destroy event

```
var myListener = $rootScope.$on('someEvent', function (event, data) { …. });
$scope$on('$destrioy', myListener);
```

Watchers

- ✓ $watch(watchExpression, listener, objectEquality)
  - ✓watchExp: the expression being watched (if expression is a function it needs to be idempotent or the $digest will detect a change at every iteration); this expression is evaluated at each digest
  - ✓listener: callback fired first when watcher is set, and then in each $digest cycle when a $watchExp change is detected
  - ✓objectequality: when true it will perform deepComparison, otherwise it will compare references
- ✓ $watchGroup(watchExpressions, listener)
  - ✓watchExpressions: array of watched expressions
  - ✓Listener:c allback fired first when watcher is set, and then in each $digest cycle when a $watchExp change is detected
- ✓ $watchCollection: shallow watches the items of a collection

Unwanted Watchers

- ✓ ngRepeat will add a watcher for each value of an array, each watcher being triggered on $digest
  - ✓Can be fixed with bound one notation **: : model**
  - ✓It can also be fixed by using tracked by in ngRepeat which reduces the number of watchers to the number of elements

- ✓ Filtering in ng-repeat also triggers unwanted watchers
  - ✓Pass filtered array to the ngRepeat
  - ✓Avoid providing the array through a non-idempotent function

LEARN stuff IO

Unwanted Watchers

```
(function () {
   var root = $(document.getElementsByTagName('body'));
   var watchers = [];
   var f = function (element) {
      if (element.data().hasOwnProperty('$scope')) {
         angular.forEach(element.data().$scope.$$watchers, function (watcher) {
            watchers.push(watcher);
         }); }
      angular.forEach(element.children(), function (childElement) {
         f($(childElement));
      });
   };
   f(root);
   console.log(watchers.length);
})();
```

Async using $q

✓Like any promise in JS a promise has 3 states: pending, resolved and rejected

✓ Essentially $q is exactly as Promise but can be used in dependency injection

✓ $q provides access to the same resolve/then and reject methods with the same meaning

✓ $q also provides defer function which is sugar syntax over resolve

✓ $q.all() allows for an array of promises to be solved and after all are solved a object will all data is returned

✓ $.race() allows for an array of promises to be solved, but only the result of the first promise will be returned

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com