Buenas Prácticas de AngularJS Services

Singletons

Los Servicios son instanciados con la palabra clave new y usan la palabra clave this para variables y funciones. Al ser estos muy similares a las factories, usar factory en su lugar para una mejor consistencia.

Nota: Todos los servicios en AngularJS son singletons. Esto significa que hay sólo una instancia de un servicio dado por inyector.

```
// service
angular
    .module('app')
    .service('logger', logger);

function logger() {
  this.logError = function(msg) {
    /* */
  };
}
```

```
// factory
angular
    .module('app')
    .factory('logger', logger);

function logger() {
    return {
        logError: function(msg) {
          /* */
        }
    };
}
```

Factories

Single Responsibility

Factories deben tener una responsabilidad y funcionalidad específica, la cual quedará encapsulada en su contexto. Si se empieza a perder o extender este propósito único, se debe pensar en llevar la funcionalidad extra a un nuevo factory.

Singletons

Factories son singletons y devuelven un objeto que contiene todos los miembros del servicio.

Nota: Todos los servicios en AngularJS son singletons.

Accessible Members Up Top

Exponer los miembros que serán invocados arriba de todo en nuestro servicio para lo cual se puede usar el Revealing Module Pattern.

Por qué?: Colocando los miembros expuestos al comienzo del servicio, facilita la lectura del mismo y obtener en un primer paneo del mismo una vista rápida de todos los elementos que estamos exponiendo para luego ser utilizados y testeados.

Por qué?: Esto es muy útil cuando el archivo empieza a ser largo y se tiene que hacer scroll hacia abajo para poder ver qué contiene.

Por qué?: Cuando se van definiendo funciones es más sencillo escribir y definer todo en una sóla línea, dejando en la parte superior todos los componentes que serán expuestos y dejando para el final del archivo una sección especial con la definición de todos ellos, haciendo nuestro código más legible.

```
/* avoid */
function dataService() {
  var someValue = '';
  function save() {
    /* */
  };
  function validate() {
    /* */
  };

  return {
      save: save,
      someValue: someValue,
      validate: validate
  };
}
```

```
/* recommended */
function dataService() {
    var someValue = '';
    var service = {
        save: save,
        someValue: someValue,
        validate: validate
    };
    return service;

    ////////////

    function save() {
        /* */
    };

    function validate() {
        /* */
    };
}
```

```
(function() {
    'use strict';

    angular
        .module('blocks.logger')
        .factory('logger', logger);

    logger.$inject = ['$log', 'toastr'];

    function logger($log, toastr) {
        var service = {
            showToasts: true,

            error   : error,
            info    : info,
            success : success,
            warning : warning,

            // straight to console; bypass toastr
            log     : $log.log
        };

        return service;
        ////////////////////

        function error(message, data, title) {
            toastr.error(message, title);
            $log.error('Error: ' + message, data);
        }

        function info(message, data, title) {
```

Functions Declarations to Hide Implementatio Details

Usar declaración de funciones para ocultar detalles de implementación. Mantener los miembros accesibles de nuestro factory al principio. Crear punteros a funciones que aparecerán luego en nuestro código.

Por qué?: Al ubicar los miembros a los cuales se puede acceder al principio, hacen la lectura más simple y nos ayuda a identificar rápidamente qué funciones del factory puede ser accedidas externamente.

Por qué?: Ubicando la implementación de nuestras funciones al final quita complejidad en la lectura de nuestro código dejando al principio los miembros utilizables que son lo más importante.

Por qué?: Las declaraciones de funciones son hoisted por lo tanto no tendrás que preocuparte por invocar una función que todavía no ha sido definida (como si puede suceder con expresiones de funciones).

Por qué?: Utilizando declaración de funciones no tendrás que preocuparte si mueves la variable a antes de la variable b haciendo que tu código se rompa debido a que a depende de b.

Por qué?: Orden es crítico con expresiones de funciones.

```
/**
* avoid
* Using function expressions
*/
function dataservice($http, $location,
$q, exception, logger) {
  var isPrimed = false;
  var primePromise;

  var getAvengers = function() {
    // implementation details go here
  };

  var getAvengerCount = function() {
     // implementation details go here
  };
```

```
var getAvengersCast = function() {
    // implementation details go here
  };

  var prime = function() {
    // implementation details go here
  };

  var ready = function(nextPromises) {
     // implementation details go here
  };

  var service = {
     getAvengersCast: getAvengersCast,
     getAvengerCount: getAvengerCount,
     getAvengers: getAvengers,
     ready: ready
  };

  return service;
}
```

```
/**
* recommended
* Using function declarations
* and accessible members up top.
*/
function dataservice($http, $location, $q,
exception, logger) {
    var isPrimed = false;
    var primePromise;

    var service = {
        getAvengersCast: getAvengersCast,
        getAvengerCount: getAvengerCount,
        getAvengers: getAvengers,
        ready: ready
    };

    return service;

    ////////////
    function getAvengers() {
       // implementation details go here
    }

    function getAvengerCount() {
        // implementation details go here
    }

    function getAvengersCast() {
       // implementation details go here
    }

    function prime() {
        // implementation details go here
    }

    function ready(nextPromises) {
        // implementation details go here
    }
}
```

Data Services

Separate Data Calls

Llevar lógica de operaciones con datos e interacción con los mismos a factories. Hacer en servicios llamadas XHR, local storage, guardado en memoria, y cualquier otra operación con datos.

Por qué?: La responsabilidad de los controladores es la de presenter información para las vistas. No deben preocuparse por cómo es obtenida la información, sinplemente saber a quién debe llamar para obtenerla. Llevando la lógica a servicios permitimos que el controlador sea más simple y enfocado específicamente en la vista.

Por qué?: Cuando un controlador utiliza servicios para accede a los datos se hace mucho más facil el testeo.

Por qué?: Los servicios para obtener datos tienen código específico para el manejo de la misma. Como por ejemplo, encabezados, cómo hablar con los datos u otros servicios tales como $http. Separando la lógica de manejo de datos en un servicio oculta la complejidad para los consumidores externos, como puede ser un controlador, hacienda más fácil cambiar su implementación.

```
/* recommended */
// dataservice factory
angular
    .module('app.core')
    .factory('dataservice', dataservice);

dataservice.$inject = ['$http', 'logger'];

function dataservice($http, logger) {
    return {
        getAvengers: getAvengers
    };

    function getAvengers() {
        return $http.get('/api/maa')
            .then(getAvengersComplete)
            .catch(getAvengersFailed);

        function getAvengersComplete(response) {
            return response.data.results;
        }

        function getAvengersFailed(error) {
            logger.error('XHR Failed for getAvengers.' + error.data);
        }
    }
}
```

Nota: El servicio de datos es llamado desde los consumidores, como puede ser un controlador, ocultando la implemtación de la lógica necesaria a los mismos.

```
/* recommended */

// controller calling the dataservice factory
angular
    .module('app.avengers')
    .controller('Avengers', Avengers);

Avengers.$inject = ['dataservice', 'logger'];

function Avengers(dataservice, logger) {
    var vm = this;
    vm.avengers = [];

    activate();

    function activate() {
        return getAvengers().then(function() {
            logger.info('Activated Avengers View');
        });
    }

    function getAvengers() {
        return dataservice.getAvengers()
            .then(function(data) {
                vm.avengers = data;
                return vm.avengers;
            });
    }
}
```

Return a Promise from Data Calls

Cuando se llame a un servicio que va a proveer datos, devolver una promesa con la información, tal como lo hace el servicio $http, devolver también una promesa desde la función que implemente la llamada.

Por qué?: Las promesas pueden encadenarse y tomar acciones futuras cuando las llamadas han sido completadas y resolver o rechazar la promesa.

```
/* recommended */

activate();

function activate() {
    /**
     * Step 1
     * Ask the getAvengers function for the
     * avenger data and wait for the promise
     */
    return getAvengers().then(function() {
        /**
         * Step 4
         * Perform an action on resolve of final promise
         */
        logger.info('Activated Avengers View');
    });
}
```

```
function getAvengers() {
    /**
     * Step 2
     * Ask the data service for the data and wait
     * for the promise
     */
    return dataservice.getAvengers()
        .then(function(data) {
            /**
             * Step 3
             * set the data and resolve the promise
             */
            vm.avengers = data;
            return vm.avengers;
    });
}
```