Comunicación con el servidor y promesas

Anxo Fole
afole@plainconcepts.com

promesas

$q

- Servicio que ofrece la api de promesas
- Implementación reducida de la librería **Q** de Kris Kowal

- Una promesa es una interfaz para interactuar con un objeto que representa el resultado de una operación que se ejecuta de manera asíncrona y, que por lo tanto, puede o no terminar

```
function asyncGreet(name) {

return $q(function (resolve, reject) {
    setTimeout(function () {

            if (okToGreet(name)) {
                resolve('Hello, ' + name + '!');
            } else {
                reject('Greeting ' + name + ' is not allowed.');
            }
    }, 1000);
});
}
```

```
var promise = asyncGreet('Robin Hood');
promise.then(function (greeting) {
    alert('Success: ' + greeting);
}, function (reason) {
    alert('Failed: ' + reason);
});
```

Estado

- Una vez creada la promesa
  - resolve(value): finaliza la promesa con éxito
  - reject(reason): finaliza la promesa con un error
- Una promesa que ha sido resuelta o rechazada una vez ya no puede cambiar su estado
- No se puede
  - Resolver una promesa ya rechazada
  - Resolver una promesa ya resuelta con un nuevo resultado
  - Rechazar una promesa ya resuelta
  - Rechazar una promesa ya rechazada con una razón diferentes

Varios callbacks

- Sobre una misma promesa se pueden registrar varios callbacks
  - Todos serán invocados

```
var success1 = function (greeting) {
    alert('Success1: ' + greeting);
};
var success2 = function (greeting) {
    alert('Success2: ' + greeting);
};

var promise = asyncGreet('Robin Hood');
promise.then(successs1);
promise.then(successs2);
```

Encadenado de promesas

- Simula la invocación de métodos síncronos en el mundo asíncrono
- return en el callback de una promesa devuelve otra promesa
- Un único callback de error

```
var success1 = function (greeting) {
    return alert('Success1: ' + greeting);
};
var success2 = function (greeting) {
    alert('Success2: ' + greeting);
};
var error = function (reason) {
    alert('Failed: ' + reason);
};

var promise = asyncGreet('Robin Hood');
promise.then(successs1).then(successs2, error);
```

Agregación de promesas

- Cuando queremos ejecutar una acción cuando más de una promesa ha sido resuelta

```
var promise1 = $q.defer();
var promise2 = $q.defer();

var agregatedPromise = $q.all([promise1, promise2]);
agregatedPromise.then(function (result) {
    var result1 = result[0];
    var result2 = result[1];
})

promise1.resolve('result1');
promise2.resolve('result2');
```

Tratar como promesas valores que no lo son

- **$q.when** nos permite envolver un objeto javascript como si fuera una promesa

```
var promise = $q.defer();
var object = 'Not a promise!';

var agregatedPromise = $q.all([promise1, $q.when(object)]);
agregatedPromise.then(function (result) {
    var result1 = result[0]; // 'Promise result'
    var result2 = result[1]; // 'Not a promise!'
})

promise.resolve('Promise result');
```

$http

$http

- Servicio que encapsula las peticiones mediante XHR o JSONP
- Ofrece métodos para realizar diferentes tipos de peticiones
- Se complementa con el servicio $httpBackend para simular las peticiones en las pruebas unitarias
- Su funcionamiento se basa en las promesas(deferred objects)

$http

- El servicio es una función que acepta un objeto de configuración como único parámetro
- Su resultado es una promesa
  - Se pueden utilizar las funciones ***success*** y ***error***

```
$http({ method: 'GET', url: '/someUrl' }).
  success(function (data, status, headers, config) {
      // this callback will be called asynchronously
      // when the response is available
  }).
  error(function (data, status, headers, config) {
      // called asynchronously if an error occurs
      // or server returns response with an error status.
  });
```

$http

- El servicio es una función que acepta un objeto de configuración como único parámetro
- Su resultado es una promesa
  - Se puede utilizar su función ***then***

```
$http({ method: 'GET', url: '/someUrl' }).
  then(function successCallback(response) {
      // this callback will be called asynchronously
      // when the response is available
  }, function errorCallback(response) {
      // called asynchronously if an error occurs
      // or server returns response with an error status.
  });
```

Response Status Code

- En función del valor del status code de la respuesta se considera válida o no
  - **Validos:** rango de 200 a 299
  - **No válidos:** 400 a 599

- Si la respuesta es un redirect (301 y 302) XHR lo seguirá

Atajos a peticiones habituales

- **$http.get**(url, config)
- **$http.head**(url, config)
- **$http.post**(url, data, config)
- **$http.put**(url, data, config)
- **$http.delete**(url, config)
- **$http.jsonp**(url, config)

Objeto configuración (I)

- **method**: {string} 'GET', 'POST', etc
- **url**: {string} – Url absoluta o relativa del servicio.
- **params**: {Object.<string|Object>} – Map of strings or objects which will be turned to ?key1=value1&key2=value2 after the url. If the value is not a string, it will be JSONified.
- **data**: {string|Object} – Data to be sent as the request message data.
- **headers**: {Object} – Map of strings or functions which return strings representing HTTP headers to send to the server. If the return value of a function is null, the header will not be sent.
- **xsrfHeaderName**: {string} – Name of HTTP header to populate with the XSRF token.
- **xsrfCookieName**: {string} – Name of cookie containing the XSRF token.

Objeto configuración (II)

- **transformRequest**: {function(data, headersGetter)|Array.<function(data, headersGetter)>} – transform function or an array of such functions. The transform function takes the http request body and headers and returns its transformed (typically serialized) version.
- **transformResponse**: {function(data, headersGetter)|Array.<function(data, headersGetter)>} – transform function or an array of such functions. The transform function takes the http response body and headers and returns its transformed (typically deserialized) version.
- **cache**: {boolean|Cache} – If true, a default $http cache will be used to cache the GET request, otherwise if a cache instance built with $cacheFactory, this cache will be used for caching.
- **timeout**: {number|Promise} – timeout in milliseconds, or promise that should abort the request when resolved.
- **withCredentials**: {boolean} - whether to set the withCredentials flag on the XHR object. See [requests with credentials]https://developer.mozilla.org/en/http_access_control#section_5 for more information.
- **responseType**: {string} - see requestType.

Parámetros de las respuestas

- **data**: {string|Object} – The response body transformed with the transform functions.
- **status**: {number} – HTTP status code of the response.
- **headers**: {function([headerName])} – Header getter function.
- **config**: {Object} – The configuration object that was used to generate the request.
- **statusText**: {string} – HTTP status text of the response.

Interceptores

- Permiten realizar operaciones antes de que las peticiones se envíen al servidor y justo antes de que nuestra aplicación procese la respuesta
- Se registran como un servicio que devuelve un objeto con las siguientes propiedades:
  - Request
  - RequestError
  - Response
  - ResponseError
- Posteriormente se añade el interceptor al array de interceptores en $httpProvider

```
$provide.factory('myHttpInterceptor', function ($q, loadingService) {
    return {
        'request': function (config) {
            loadingService.show();
            return config;
        },
        'requestError': function (rejection) {
             loadingService.hide();
            return $q.reject(rejection);
        },
        'response': function (response) {
            loadingService.hide();
            return response;
        },
        'responseError': function (rejection) {
         loadingService.hide();
            return $q.reject(rejection);
        }
    };
});

$httpProvider.interceptors.push('myHttpInterceptor');
```

Same Origin Policy

- Restricción al objeto XHR para que pueda hacer peticiones sólo a recursos con su mismo origen (protocolo, host y puerto)
- Por defecto activado en AngularJS

$httpBackend

- Servicio interno utilizado por $http
- Mockeado por angular mock
  - Utilizado en test unitarios para poder evaluar el comportamiento de objetos que hacen uso del servicio $http

$resource

$resource

- Servicio de más alto nivel que $http para utilizar junto con API’s REST
- Definido en módulo independiente
  - ngResource