El uso de RxJS Ajax para realizar peticiones asíncronas cada día es más común. ¿Cómo podemos realizar peticiones Ajax con RxJS y gestionar invocaciones complejas?. Vamos a verlo , el primer paso va a ser configurar un pequeño servidor de Node.js con Express.js como framework de servidor. El primer paso es usar npm e instalar express.js.

npm install express –save

Una vez instalado el framework express vamos a ver el código que vamos a usar del lado de servidor.

```
var express = require('express');
var app = express();

app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers",
   "Origin, X-Requested-With, Content-Type, Accept");
  next();
});


var persona={nombre:"juan",edad:20};
var persona2={nombre:"gema",edad:30};

app.get('/persona1', function (req, res) {
  setTimeout(function() {

      res.send(persona);
  },2000)
```

```
});

app.get('/persona2', function (req, res) {
  setTimeout(function() {
  res.send(persona2);

},4000)

});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

En este caso simplemente estamos publicando dos urls . La primera url es la de /persona1 que nos envía los datos pasados 2 segundos. La segunda url es /persona2 que nos envía los datos de la segunda persona al pasar 4 segundos.

Vamos a ver como podemos recoger los datos e imprimir el nombre de la persona que tiene una edad mayor que 20 por la consola. El primer paso es realizar es crear una pagina web que tenga dos botones.

```
  function pulsar() {
      // modificamos y ponemos Rx.Observable.ajax
      Rx.Observable.ajax({ url: 'http://localhost:3000/persona1',
responseType: 'json'})
        .subscribe(function (data) {
            //añadimos la propiedad response y saldra por la consola
            console.log(data.response);
        });

      Rx.Observable.ajax({ url: 'http://localhost:3000/persona2',
responseType: 'json'})
        .subscribe(function (data) {
            //añadimos la propiedad response y saldra por la consola
            console.log(data.response);
          });
      }
```

Una vez hemos construido este código la extensiones Rx harán su función y realizaran las peticiones Ajax. Las cuales veremos su resultado impreso por la consola.

El problema es que cada una de las peticiones se ejecuta y se resuelve en tiempos diferentes. A nosotros nos vendría bien ejecutar ambas y cuando hayan terminado obtener un resultado por pantalla. Esto con jQuery clásico se resuelve con promesas. Vamos a ver como lo resolvemos con Rxjs.

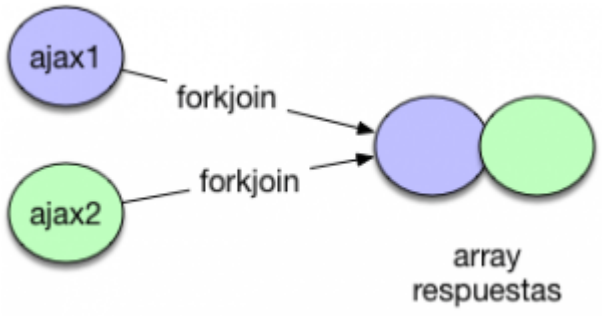
```
  function pulsar3() {

      var observable1=Rx.Observable.ajax({ url:
'http://localhost:3000/persona1', responseType: 'json'})
      var observable2=Rx.Observable.ajax({ url:
'http://localhost:3000/persona2', responseType: 'json'})
      Rx
      .Observable
      .forkJoin(observable1,observable2)
      .flatMap(x=>x)
      .map(x=>x.response)
      .subscribe(function(datos) {
          console.log(datos);
      })
```

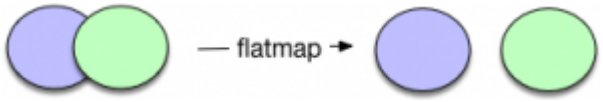
El resultado sera muy parecido con la diferencia que los datos saldrán a la vez:

¿Cómo funciona el código exactamente?

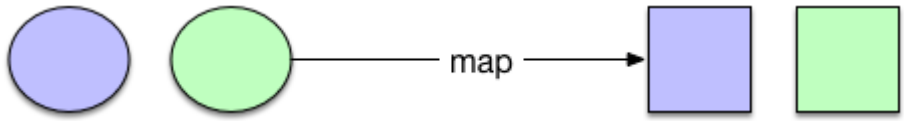
forkJoin: Este método fusiona ambas peticiones Rest y nos devuelve un array con los resultados finales.



flatmap: Una vez tenemos el array de resultado ejecutamos flatmap. Esta instrucción convierte el array de respuestas en dos observables diferentes.



map: Utilizamos la función map para extraer de la respuesta los datos json que vienen en la propiedad de response.



Ya tenemos los datos completamente transformados usamos el método subscribe y los imprimimos por la consola.

¿Como podríamos imprimir los mayores de 30 años? . Sería suficiente con modificar el flujo de datos y añadir una función de filter.

```
Rx
.Observable
.forkJoin(observable1,observable2)
.flatMap(x=>x)
.map(x=>x.response)
.filter(x=>x.edad>=30)
.subscribe(function(datos) {
console.log(datos);
})
```

Saldrá gema por la consola:

Tener un conocimiento fuerte de el manejo de Observables va a ser un tema fundamental en el futuro diseño de arquitecturas. Acabamos de ver un ejemplo de RxJS Ajax.

Otros artículos relacionados:

1. Usando Rx Observables en JavaScript
2. JavaScript Deferred Objects y peticiones asíncronas
3. JavaScript Promise y la programación asíncrona
4. Rx.js documentación