RxJava es una de las librerías que más interés esta generando dentro de la comunidad. Esto es debido a que cada día nos encontramos con más sistemas que funcionan de forma asíncrona y que necesitan poder gestionar flujos de información complejos. Para entender como funciona RxJava necesitamos entender como funcionan los Observables que son los elementos principales de la librería.

RxJava Observables e Iterables

Los observables tienen fuertes similitudes con los Iterables , recordemos que estos últimos son estructuras de datos que se pueden recorrer utilizando el interface Iterator. Vamos a verlos en acción para recordarlos un poco.

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class Principal0 {

 public static void main(String[] args) {

 List<String> lista= new
ArrayList<String>(Arrays.asList("hola","que","tal"));

 Iterator<String> i= lista.iterator();
 while(i.hasNext()) {
```

```
 System.out.println(i.next());


 }
 }

}
```

En este caso recorremos una lista de cadenas y las imprimimos por pantalla.



Vamos a ver el mismo ejemplo a través del uso de RxJava y Observables . El primer paso que tendremos que realizar es instalar las dependencias de Maven necesarias:

```
<dependency>
<groupId>io.reactivex</groupId>
<artifactId>rxjava</artifactId>
<version>1.0.8</version>
</dependency>
```

Una vez instaladas las dependencias vamos a crear un Observable y subscribirnos a él.

```
package com.arquitecturajava;

import java.util.Arrays;

import rx.Observable;
import rx.functions.Action1;

public class Principal {

 public static void main(String[] args) {

 Observable<String> o =
Observable.from(Arrays.asList("hola","que","tal"));

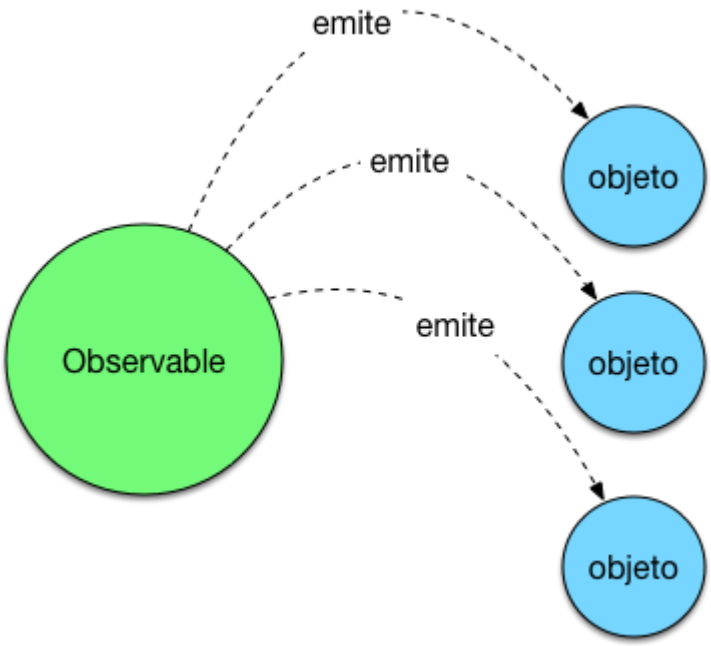
 o.subscribe(new Action1<String>() {

 public void call(String arg0) {
 System.out.println(arg0);

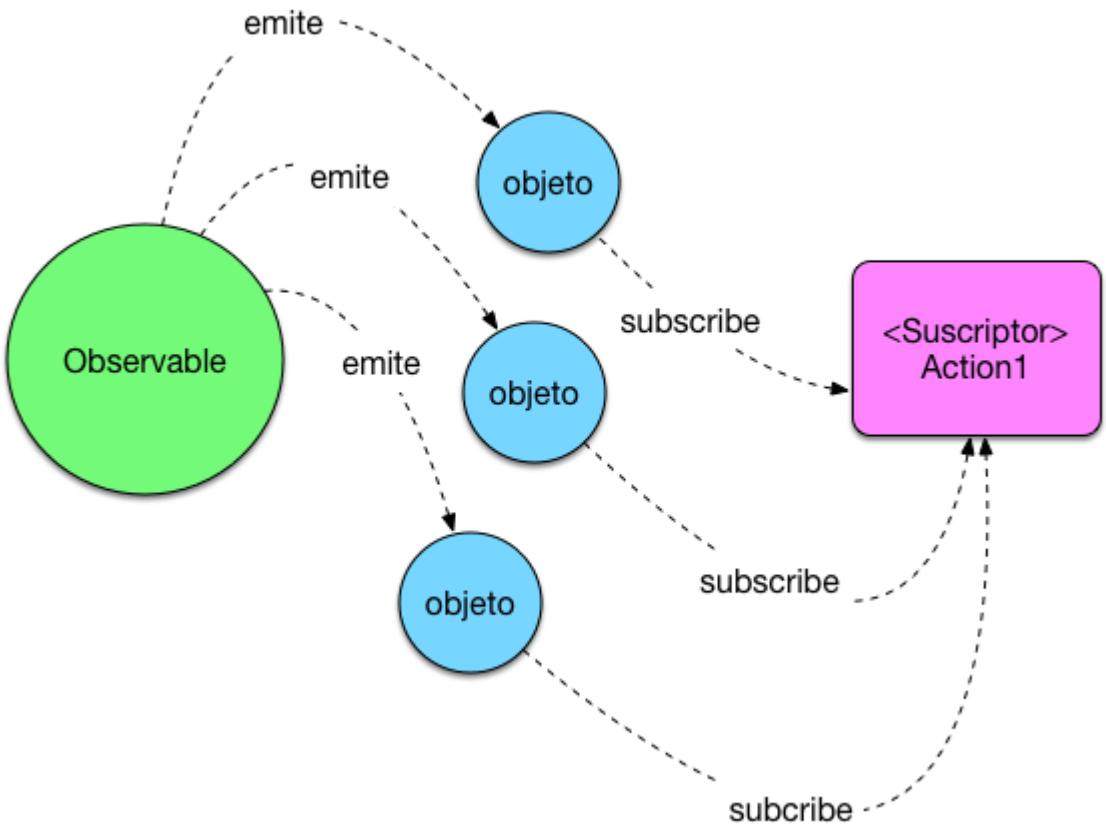
 }
 });

 }
}
```

La forma de trabajar es muy similar en este caso tenemos una lista que se convierte en un Observable. Este Observable emitirá los diferentes elementos de la lista.



En este caso tenemos un objeto Action1 que se subscribe a la lista de objetos que el observable emite.



El resultado que obtenemos por pantalla es idéntico al anterior.



¿Cual es entonces la ventaja de usar los Observables?. Los observables están orientados a gestionar flujos asíncronos. Vamos a ver un ejemplo:

```
package com.arquitecturajava;

import java.util.concurrent.TimeUnit;

import rx.Observable;
import rx.functions.Action1;

public class Principal2 {

 public static void main(String[] args) throws InterruptedException {


 Observable<Long> observable1=Observable.interval(500,
TimeUnit.MILLISECONDS).take(20);
 Observable<Long> observable2=Observable.interval(200,
TimeUnit.MILLISECONDS).take(20);

 Observable.merge(observable1,observable2).subscribe(new
Action1<Long>() {

 @Override
 public void call(Long arg0) {

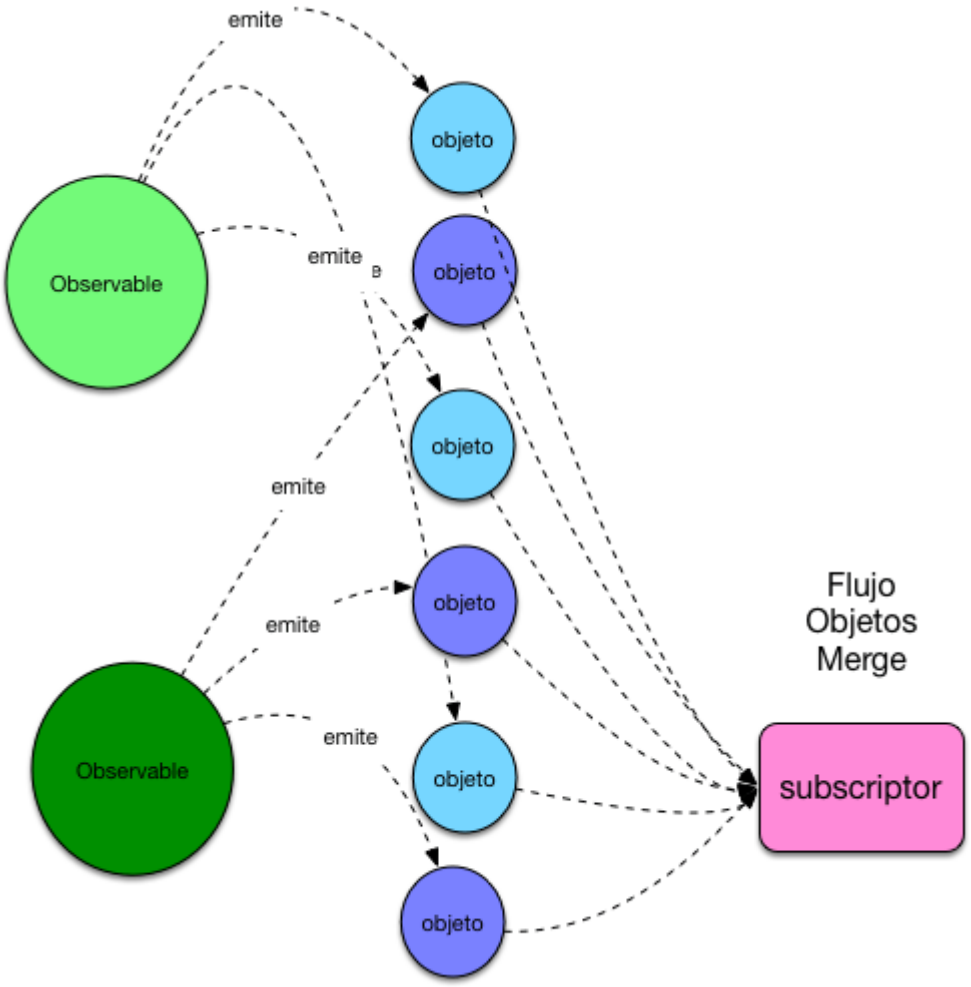

 System.out.println(arg0);

 }
 });

 Thread.sleep(20000);
 }
```

}

En este caso hemos construido dos observables. Cada uno de ellos emite 20 números a diferentes intervalos de tiempo (cada uno crea un Thread). Podríamos considerarlos dos tareas asíncronas diferentes. Vamos a usar RxJava para fusionar ambas tareas utilizando el método merge y tratarlas como si fuera una única.



El resultado de utilizar el método Merge es que podemos utilizar el mismo subscriptor para

ambas tareas y se irán intercalando en la consola.



RxJava es una de las librerías que más futuro tiene , recordemos que una parte importante de las Arquitecturas de microservicios serán asíncronas.

Otros artículos relacionados: Iterator , MicroServicios