Reattività

È una implementazione dell'Observer pattern done right. Un Observable in Rx è un oggetto concettualmente simile ad uno stream, che emette nel tempo una sequenza di valori.

E' possibile osservare i valori emessi da un Observable fornendo il comportamento da adottare in caso di:

- valore ricevuto
- eccezione lanciata da un precedente componente
- termine del flusso di dati

Evento	Iterable	Observable
successivo	T next()	onNext(T)
errore	lancia Exception	onError(E)
completamento	!hasNext()	onCompleted()

L'Observable ribalta il funzionamento dell'Iterable, secondo il paradigma dello Stream. Aggiunge però, rispetto alla libreria standard, la gestione esplicita di errori e completamento dello stream.

Il risultato è:

- una semantica più ricca
- maggiore regolarità nella composizione
- indipendenza dal modello di esecuzione (sincrono/asincrono)

La maggior parte degli operatori sugli Observable accettano uno Scheduler come parametro. Ogni operatore può così essere reso concorrente; lo Scheduler scelto permette di indicare il tipo di concorrenza desiderato.

Un Subscriber rappresenta un ascoltatore di un Observable: fornisce il codice che reagisce agli eventi per ottenere il risultato finale dalla catena di elaborazione.

Un Subject può consumare uno o più Observable, per poi comportarsi esso stesso da Observable e quindi introdurre modifiche sostanziali nel flusso degli eventi.

Reactive streams

```
public interface Publisher< T >
{
    public void subscribe(Subscriber< ? super T > s);
}

public interface Subscriber< T >
{
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

```
public interface Subscription
{
    public void request(long n);
    public void cancel();
}

public interface Processor< T, R >
{
    extends Subscriber< T >, Publisher< R > {
}
```

Operatori

Map

Trasforma gli elementi di uno stream, ottenendo uno stream di elementi trasformati.

Flatmap

Trasforma gli elementi di uno stream, concatenando i risultati in un solo stream.

Filter

Emette uno stream contenente solo gli elementi che soddisfano un predicato.

Skip

Emette uno stream saltando i primi N elementi della sorgente.

Zip

Emette uno stream combinando a coppie elementi di due stream in ingresso.

Debounce

Emette un elemento solo se è passato un lasso di tempo dall'ultimo elemento della sorgente.

Window

Emette uno stream di partizioni dello stream sorgente.

Parallelismo

L'asincronia nell'esecuzione dei vari operatori è definita dallo Schedulatore usato per osservare un Observable o definire un operatore di uno stream.

Metodo	Schedulatore
.io()	Per stream legati alle operazioni d IO
.single()	Usa un singolo thread
.computation()	Per operatori legati al calcolo
.from(ex)	Usa l'Executor fornito

L'operatore parallel() permette di indicare che uno stream, da un certo punto in poi. In questa modalità, solo alcuni operatori sono consentiti ed è necessario specificare lo schedulatore da usare con il metodo .runOn(scheduler).

Il metodo sequential() indica che da quel punto in poi la pipeline di elaborazione va nuovamente intesa come sequenziale.

A differenza degli Stream della libreria di base, è possibile indicare una precisa sezione della pipeline che viene configurata parallelamente.