30 天精通RxJS (25)：Subject 总结

2017年1月10日。3 分钟阅读

Subject 其实在RxJS 中最常被误解的一部份，因为Subject 可以让你用命令式的方式虽送值到一个observable 的串流中。

很多人会直接把Subject拿来用在**不知道如何建立Observable的状况**，比如我们在 30天精通RxJS(23) 中提到的可以用在ReactJS的Event中，来建立event的observable

```
class MyButton extends React.Component {
    constructor(props) {
        super(props);
        this.state = { count: 0 };
        this.subject = new Rx.Subject();

        this.subject
            .mapTo(1)
            .scan((origin, next) => origin + next)
            .subscribe(x => {
                this.setState({ count: x })
            })
    }
    render() {
        return <button onClick={event => this.subject.next(event)}>{th
    }
}
```

因为在React API的关系，如果我们想要把React Event转乘observable就可以用Subject帮我们做到这件事；但绝大多数的情况我们是可以透过 `Observable.create` 来做到这件事，像下面这样

```
const example = Rx.Observable.creator(observer => {
    const source = getSomeSource(); // 某個資料源
    source.addListener('some', (some) => {
        observer.next(some)
    })
```

Subject 与Observable 的差异

永远记得Subject 其实是Observer Design Pattern 的实作，所以当observer 订阅到subject 时，subject 会把订阅者塞到一份订阅者清单，在元素发送时就是在遍历这份清单，并把元素一一送出，这跟Observable 像是一个function 执行是完全不同的(请参考05 篇)。

Subject之所以具有Observable的所有方法，是因为Subject继承了Observable的型别，其实Subject型别中**主要**实做的方法只有next、error、 complete、subscribe及unsubscribe这五个方法，而这五个方法就是依照Observer Pattern下去实作的。

总而言之，Subject 是Observable 的子类别，这个子类别当中用上述的五个方法实作了Observer Pattern，所以他同时具有Observable 与Observer 的特性，而跟Observable 最大的差异就是Subject 是具有状态的，也就是储存的那份清单！

当前版本会遇到的问题

因为Subject 在订阅时，是把observer 放到一份清单当中，并在元素要送出(next)的时候遍历这份清单，大概就像下面这样

```
//...
next() {
    // observers 是一個陣列存有所有的 observer
    for (let i = 0; i < observers.length; i++) {
        observers[i].next(value);
    }
}
//...
```

这会衍伸一个大问题，就是在某个observer 发生错误却没有做错误处理时，就会影响到别的订阅，看下面这个例子

```
const source = Rx.Observable.interval(1000);
const subject = new Rx.Subject();

const example = subject.map(x => {
```

```
});
subject.subscribe(x => console.log('A', x));
example.subscribe(x => console.log('B', x));
subject.subscribe(x => console.log('C', x));

source.subscribe(subject);
```

JSBin

上面这个例子，大家可能会预期B 会在送出1 的时候挂掉，另外A 跟C 则会持续发送元素，确实正常应该像这样运席；但目前RxJS 的版本中会在B 报错之后，A 跟C 也同时停止运行。原因就像我前面所提的，在遍历所有observer 时发生了例外会导致之后的行为停止。

> 这个应该会在之后的版本中改掉的，前阵子才在 TC39 Observable proposal 中讨论完。

那要如何解决这个问题呢？目前最简单的方式当然是尽可能地把所有observer 的错误处理加进去，这样一来就不会有例外发生

```
const source = Rx.Observable.interval(1000);
const subject = new Rx.Subject();

const example = subject.map(x => {
    if (x === 1) {
        throw new Error('oops');
    }
    return x;
});
subject.subscribe(
    x => console.log('A', x),
    error => console.log('A Error:' + error));
example.subscribe(x => console.log('B', x),
    error => console.log('B Error:' + error));
subject.subscribe(x => console.log('C', x),
    error => console.log('C Error:' + error));

source.subscribe(subject);
```

JSBin

```
const source = Rx.Observable.interval(1000);
const subject = new Rx.Subject().observeOn(Rx.Scheduler.asap);

const example = subject.map(x => {
    if (x === 1) {
        throw new Error('oops');
    }
    return x;
});
subject.subscribe(x => console.log('A', x));
example.subscribe(x => console.log('B', x));
subject.subscribe(x => console.log('C', x));

source.subscribe(subject);
```

一定需要使用Subject 的时机？

Subject 必要的使用时机除了本篇文章一开始所提的之外，正常应该是当我们一个observable 的操作过程中发生了side-effect 而我们不希望这个side-effect 因为多个subscribe 而被触发多次，比如说下面这段程式码

```
var result = Rx.Observable.interval(1000).take(6)
             .map(x => Math.random()); // side-effect，平常有可能是呼叫 AP

var subA = result.subscribe(x => console.log('A: ' + x));
var subB = result.subscribe(x => console.log('B: ' + x));
```

JSBin

这段程式码A 跟B 印出来的乱数就不一样，代表random(side-effect) 被执行了两次，这种情况就一定会用到subject(或其相关的operators)

```
var result = Rx.Observable.interval(1000).take(6)
             .map(x => Math.random()) // side-effect
```

```
var subB = result.subscribe(x => console.log('B: ' + x));
```

JSBin

改成这样后我们就可以让side-effect 不会因为订阅数而多执行，这种情状就是一定要用subject的。

今日小结

今天总结了Subject 的使用情境，以及厘清跟Observable 的关系，并且指出在使用时要避免犯发生的错误。

这几点都非常的重要，不知道今天读者有没有收获呢？如果有任何问题，欢迎在下方留言给我，谢谢！

标签

JavaScript RxJS 可观察的 主题 运算符 RxJS 30天

上一页 下一个