30 天精通 RxJS (23)：Subject, BehaviorSubject, ReplaySubject, AsyncSubject

2017年1月8日。3 分钟阅读

昨天我们介绍了Subject 是什么，今天要讲Subject 一些应用方式，以及Subject 的另外三种变形。

学科

昨天我们讲到了Subject 实际上就是Observer Pattern 的实作，他会在内部管理一份observer 的清单，并在接收到值时遍历这份清单并送出值，所以我们可以这样用Subject

```
var subject = new Rx.Subject();

var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

subject.subscribe(observerA);
subject.subscribe(observerB);

subject.next(1);
// "A next: 1"
```

JSBin

这里我们可以直接用subject 的next 方法传送值，所有订阅的observer 就会接收到，又因为Subject 本身是Observable，所以这样的使用方式很适合用在某些无法直接使用Observable 的前端框架中，例如在React 想对DOM 的事件做监听

```
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
    }
}
```

JSBin | 的jsfiddle

从上面的程式码可以看出来，因为React 本身API 的关系，如果我们想要用React 自订的事件，我们没办法直接使用Observable 的creation operator 建立observable，这时就可以靠Subject 来做到这件事。

Subject 因为同时是observer 和observable，所以应用面很广除了前面所提的之外，还有上一篇文章讲的组播(multicase)特性也会在接下来的文章做更多应用的介绍，这里先让我们来看看Subject 的三个变形。

BehaviorSubject

很多时候我们会希望Subject 能代表当下的状态，而不是单存的事件发送，也就是说如果今天有一

```
var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}

var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

subject.subscribe(observerA);

subject.next(1);
// "A next: 1"
subject.next(2);
// "A next: 2"
subject.next(3);
// "A next: 3"

setTimeout(() => {
    subject.subscribe(observerB); // 3 秒後才訂閱，observerB 不會收到任何值
},3000)
```

以上面这个例子来说，observerB订阅的之后，是不会有任何元素送给observerB的，因为在这之后没有执行任何 `subject.next()` ，但很多时候我们会希望subject能够表达当前的状态，在一订阅时就能收到最新的状态是什么，而不是订阅后要等到有变动才能接收到新的状态，以这个例子来说，我们希望observerB订阅时就能立即收到 `3` ，希望做到这样的效果就可以用 BehaviorSubject 。

BehaviorSubject 跟Subject 最大的不同就是BehaviorSubject 是用来呈现当前的值，而不是单纯的发送事件。BehaviorSubject 会记住最新一次发送的元素，并把该元素当作目前的值，在使用上BehaviorSubject 建构式需要传入一个参数来代表起始的状态，范例如下

```
var subject = new Rx.BehaviorSubject(0); // 0 為起始值
var observerA = {
```

```
var observerB = {
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

subject.subscribe(observerA);
// "A next: 0"
subject.next(1);
// "A next: 1"
subject.next(2);
// "A next: 2"
subject.next(3);
// "A next: 3"

setTimeout(() => {
    subject.subscribe(observerB);
    // "B next: 3"
},3000)
```

JSBin | 的jsfiddle

从上面这个范例可以看得出来BehaviorSubject 在建立时就需要给定一个状态，并在之后任何一次订阅，就会先送出最新的状态。其实这种行为就是一种状态的表达而非单存的事件，就像是年龄跟生日一样，年龄是一种状态而生日就是事件；所以当我们想要用一个stream 来表达年龄时，就应该用BehaviorSubject 。

ReplaySubject

在某些时候我们会希望Subject 代表事件，但又能在新订阅时重新发送最后的几个元素，这时我们就可以用ReplaySubject，范例如下

```
var subject = new Rx.ReplaySubject(2); // 重複發送最後 2 個元素
var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
```





```
    next: value => console.log('B next: ' + value),
    error: error => console.log('B error: ' + error),
    complete: () => console.log('B complete!')
}

subject.subscribe(observerA);
subject.next(1);
// "A next: 1"
subject.next(2);
// "A next: 2"
subject.next(3);
// "A next: 3"

setTimeout(() => {
    subject.subscribe(observerB);
    // "B next: 2"
    // "B next: 3"
},3000)
```

JSBin |

可能会有人以为 `ReplaySubject(1)` 是不是就等同于BehaviorSubject，其实是不一样的，BehaviorSubject在建立时就会有起始值，比如 `BehaviorSubject(0)` 起始值就是 `0` ，BehaviorSubject是代表着状态而ReplaySubject只是事件的重放而已。

AsyncSubject

AsyncSubject是最怪的一个变形，他有点像是operator `last` ，会在subject结束后送出最后一个值，范例如下

```
var subject = new Rx.AsyncSubject();
var observerA = {
    next: value => console.log('A next: ' + value),
    error: error => console.log('A error: ' + error),
    complete: () => console.log('A complete!')
}
```

```
}

subject.subscribe(observerA);
subject.next(1);
subject.next(2);
subject.next(3);
subject.complete();
// "A next: 3"
// "A complete!"

setTimeout(() => {
    subject.subscribe(observerB);
    // "B next: 3"
    // "B complete!"
},3000)
```

JSBin |

从上面的程式码可以看出来，AsyncSubject 会在subject 结束后才送出最后一个值，其实这个行为跟Promise 很像，都是等到事情结束后送出一个值，但实务上我们非常非常少用到AsyncSubject，绝大部分的时候都是使用BehaviorSubject 跟ReplaySubject 或Subject。

~~我们把AsyncSubject 放在大脑的深处就好~~

今日小结

今天介绍了Subject 的一些应用方式，以及BehaviorSubject, ReplaySubject, AsyncSubject 三个变形各自的特性介绍，不知道读者么是否有收获呢？如果有任何问题，欢迎在下方留言给我！

标签

JavaScript RxJS 可观察的 主题 RxJS 30天

上一页 下一个

Series / 30 天精通 RxJS