30 天精通 RxJS (16)：Observable Operators - catch, retry, retryWhen, repeat

2017年1月1日。6 分钟阅读

我们已经快把所有基本的转换(Transformation)、过滤(Filter)和合并(Combination)的operators 讲完了。今天要讲错误处理(Error Handling)的operators，错误处理是非同步行为中的一大难题，尤其有多个交错的非同步行为时，更容易凸显错误处理的困难。就让我们一起来看看在RxJS 中能如何处理错误吧！

运营商

抓住

catch 是很常见的非同步错误处理方法，在RxJS 中也能够直接用catch 来处理错误，在RxJS 中的catch 可以回传一个observable 来送出新的值，让我们直接来看范例：

```
var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
                .map(x => x.toUpperCase())
                .catch(error => Rx.Observable.of('h'));

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | 的jsfiddle

catch 可以回傳一個新的Observable、Promise、Array 或任何Iterable 的物件，來傳送之後的元素。

以我们的例子来说最后就会在送出 X 就结束，画成Marble Diagram如下

```
source : ----a----b----c----d----2|
         map(x => x.toUpperCase())
          ----a----b----c----d----X|
         catch(error => Rx.Observable.of('h'))
example: ----a----b----c----d----h|
```

這裡可以看到，當錯誤發生後就會進到 catch 並重新處理一個新的 observable，我們可以利用這個新的 observable 來送出我們想送的值。

也可以在遇到錯誤後，讓 observable 結束，如下

```
var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
                .map(x => x.toUpperCase())
                .catch(error => Rx.Observable.empty());

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | JSFiddle

回傳一個 empty 的 observable 來直接結束(complete)。

另外 catch 的 callback 能接收第二個參數，這個參數會接收當前的 observalbe，我們可以回傳當前的 observable 來做到重新執行，範例如下

```
var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
```

```
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | JSFiddle

這裡可以看到我們直接回傳了當前的 obserable(其實就是 example)來重新執行，畫成 Marble Diagram 如下

```
source : ----a----b----c----d----2|
         map(x => x.toUpperCase())
          ----a----b----c----d----X|
         catch((error, obs) => obs)
example: ----a----b----c----d--------a----b----c----d--..
```

因為是我們只是簡單的示範，所以這裡會一直無限循環，實務上通常會用在斷線重連的情境。

另上面的處理方式有一個簡化的寫法，叫做 `retry()`。

retry

如果我們想要一個 observable 發生錯誤時，重新嘗試就可以用 retry 這個方法，跟我們前一個講範例的行為是一致

```
var source = Rx.Observable.from(['a','b','c','d',2])
           .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
              .map(x => x.toUpperCase())
              .retry();

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | JSFiddle

```
var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
                .map(x => x.toUpperCase())
                .retry(1);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
// a
// b
// c
// d
// a
// b
// c
// d
// Error: TypeError: x.toUpperCase is not a function
```

JSBin | JSFiddle

這裡我們對 retry 傳入一個數值 1，能夠讓我們只重複嘗試 1 次後送出錯誤，畫成 Marble Diagram 如下

```
source : ----a----b----c----d----2|
        map(x => x.toUpperCase())
         ----a----b----c----d----X|
               retry(1)
example: ----a----b----c----d--------a----b----c----d----X|
```

這種處理方式很適合用在 HTTP request 失敗的場景中，我們可以設定重新發送幾次後，再秀出錯誤訊息。





observable。

這裡我們直接來看程式碼

```
var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source
                .map(x => x.toUpperCase())
                .retryWhen(errorObs => errorObs.delay(1000));

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | JSFiddle

這裡 retryWhen 我們傳入一個 callback，這個 callback 有一個參數會傳入一個 observable，這個 observable 不是原本的 observable(example) 而是例外事件送出的錯誤所組成的一個 observable，我們可以對這個由錯誤所組成的 observable 做操作，等到這次的處理完成後就會重新訂閱我們原本的 observable。

這個範例我們是把錯誤的 observable 送出錯誤延遲 1 秒，這會使後面重新訂閱的動作延遲 1 秒才執行，畫成 Marble Diagram 如下

```
source : ----a----b----c----d----2|
        map(x => x.toUpperCase())
         ----a----b----c----d----X|
        retryWhen(errorObs => errorObs.delay(1000))
example: ----a----b----c----d-------------------a----b----c----d----...
```

從上圖可以看到後續重新訂閱的行為就被延後了，但實務上我們不太會用 retryWhen 來做重新訂閱的延遲，通常是直接用 catch 做到這件事。這裡只是為了示範 retryWhen 的行為，實務上我們通常會把 retryWhen 拿來做錯誤通知或是例外收集，如下

```
var source = Rx.Observable.from(['a','b','c','d',2])
```

```
                .retryWhen(
                errorObs => errorObs.map(err => fetch('...')));

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

這裡的 `errorObs.map(err => fetch('...'))` 可以把 errorObs 裡的每個錯誤變成 API 的發送，通常這裡個 API 會像是送訊息到公司的通訊頻道(Slack 等等)，這樣可以讓工程師馬上知道可能哪個 API 掛了，這樣我們就能即時地處理。

> retryWhen 實際上是在背地裡建立一個 Subject 並把錯誤放入，會在對這個 Subject 進行內部的訂閱，因為我們還沒有講到 Subject 的觀念，大家可以先把它當作 Observable 就好了，另外記得這個 observalbe 預設是無限的，如果我們把它結束，原本的 observable 也會跟著結束。

repeat

我們有時候可能會想要 retry 一直重複訂閱的效果，但沒有錯誤發生，這時就可以用 repeat 來做到這件事，範例如下

```
var source = Rx.Observable.from(['a','b','c'])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source.repeat(1);

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});

// a
// b
// c
// a
```

JSBin | JSFiddle

这里repeat 的行为跟retry 基本一致，只是retry 只有在例外发生时才触发，画成Marble Diagram 如下

```
source : ----a----b----c|
            repeat(1)
example: ----a----b----c----a----b----c|
```

同样的我们可以不给参数让他无限循环，如下

```
var source = Rx.Observable.from(['a','b','c'])
            .zip(Rx.Observable.interval(500), (x,y) => x);

var example = source.repeat();

example.subscribe({
    next: (value) => { console.log(value); },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | 的jsfiddle

这样我们就可以做动不断重复的行为，这个可以在建立轮询时使用，让我们不断地发request 来更新画面。

最后我们来看一个错误处理在实务应用中的小范例

```
const title = document.getElementById('title');

var source = Rx.Observable.from(['a','b','c','d',2])
            .zip(Rx.Observable.interval(500), (x,y) => x)
            .map(x => x.toUpperCase());
            // 通常 source 會是建立即時同步的連線，像是 web socket

var example = source.catch(
                (error, obs) => Rx.Observable.empty()
```





```
example.subscribe({
    next: (value) => { title.innerText = value },
    error: (err) => { console.log('Error: ' + err); },
    complete: () => { console.log('complete'); }
});
```

JSBin | 的jsfiddle

这个范例其实就是模仿在即时同步断线时，利用catch 返回一个新的observable，这个observable 会先送出错误讯息并且把原本的observable 延迟5 秒再做合并，虽然这只是一个模仿，但它清楚的展示了RxJS 在做错误处理时的灵活性。

今日小结

今天我们讲了三个错误处理的方法还有一个repeat operator，这几个方法都很有机会在实务上用到，不知道今天大家有没有收获呢？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 运算符 RxJS 30天

上一页 下一个