RxJS mergeMap和switchMap

https://cloud.tencent.com/developer/article/1533003

假设现在有一个简单的任务：页面上有一个按钮，当你点击按钮的时候，需要启动一个定时器。使用 RxJS 我们可以可以很方便地实现上述功能：

const button = document.querySelector('button');

fromEvent(button, 'click').subscribe(event => {

interval(1000).subscribe(num => {

console.log(num);

});

});

虽然以上代码能够正常运行，但仍存在两个问题：

存在类似于回调地域的问题。 我们必须手动处理每个订阅。

接下来让我们来介绍一下高阶 observable 及如何利用它使得事情变得更简单。

高阶 Observables

一个 Observable 对象可以发出任何类型的值：数值、字符串、对象等等。这意味着 Observable 对象也可以发出 Observable 类型的值。

与 JavaScript 高阶函数类似，一个高阶的 Observable 表示一个 Observable 对象内部会返回另一个 Observable 对象。此时我们来更新一下上面的示例，以便更加直观的了解上述概念：

import { fromEvent, interval } from 'rxjs';

import { map } from 'rxjs/operators';

const button = document.querySelector('button');

const click$ = fromEvent(button, 'click');

const interval$ = interval(1000);

const clicksToInterval$ = click$.pipe(map(event => {

return interval$;

}));

clicksToInterval$.subscribe(intervalObservable =>

console.log(intervalObservable)

);

当用户点击按钮时，我们的 map 操作符将返回一个 interval observable 对象。当我们订阅 clicksToInterval$ 对象时，将发出 intervalObservable 对象。

在你订阅 clicksToInterval$ 对象时，控制台输出的是 intervalObservable 对象。这里需要记住的是，observable 对象是 lazy 的，如果想要从一个 observable 对象中获取值，你必须执行订阅操作，比如：

clicksToInterval$.subscribe(intervalObservable => {

intervalObservable.subscribe(num => {

console.log(num);

});

});

在介绍高阶 observable 对象的概念之后，接下来让我们来介绍两个有用的操作符，用来帮助我们解决上面提到的问题。

mergeAll

When the inner observable emits, let me know by merging the value to the outer observable.

mergeAll() 操作符底层做的操作跟上面的例子一样，它获取 inner observable 对象，执行订阅操作，然后把值推给 observer （观察者）对象。

import { fromEvent, interval } from 'rxjs';

import { map, mergeAll } from 'rxjs/operators';

const click$ = fromEvent(button, 'click');

const interval$ = interval(1000);

const observable$ = click$.pipe(map(event => {

return interval$;

}));

observable$.mergeAll().subscribe(num => console.log(num));

在上面的示例中，source observable 对象是 clicks$ observable 对象，而 inner observable 对象是 interval$ observable 对象。

在 RxJS 中这是一个通用的模式，因此有一个快捷方式来实现相同的行为 —— mergeMap()：

mergeMap() <=> map() + mergeAll()

const button = document.querySelector('button');

const click$ = fromEvent(button, 'click');

const interval$ = interval(1000);

const observable$ = click$.pipe(mergeMap(event => {

return interval$;

}));

observable$.subscribe(num => console.log(num));

在上面的代码中，每当我们点击按钮，我们都会调用 interval$ 对象的 subscribe() 方法，这将导致在我们的页面中会存在多个独立的定时器。如果这是你期望实现的功能，那就没问题。但如果你只想保持一个数据源，你就需要使用 switch() 操作符。

switch

类似于mergeMap()，但是当源可观察对象发出时，取消之前对内部可观察对象的任何订阅。

switch() 用于取消前一个订阅，并切换至新的订阅。如果我们把代码更新为 switch() 操作符，当我们多次点击按钮时，我们可以看到每次点击按钮时，我们将获取新的 interval 对象，而上一个 interval 对象将会被自动取消。

const button = document.querySelector('button');

const click$ = fromEvent(button, 'click');

const interval$ = interval(1000);

const observable$ = click$.pipe(

map(event => {

return interval$;

}),

switchAll()

);

observable$.subscribe(num => console.log(num));

正如你说看到的，当点击三次按钮后，我们仅有一个 interval 对象。反之，使用 merge() 操作符，我们会有三个独立的 interval 对象。当源发出新值后，switch 操作符会对上一个内部的订阅对象执行取消订阅操作。

在 RxJS 中这也是一个通用的模式，因此也有一个快捷方式来实现相同的行为 —— switchMap()：

switchMap() <=> map() + switch()

const button = document.querySelector('button');

const click$ = fromEvent(button, 'click');

const interval$ = interval(1000);

const observable$ = click$.pipe(

switchMap(event => {

return interval$;

})

);

observable$.subscribe(num => console.log(num));

感兴趣的同学可以查看 Stackblitz 完整示例。