最简Rxjs入门教程--别再被Rxjs的概念淹没了

芋仔 2021-09-02 9,726 阅读7分钟

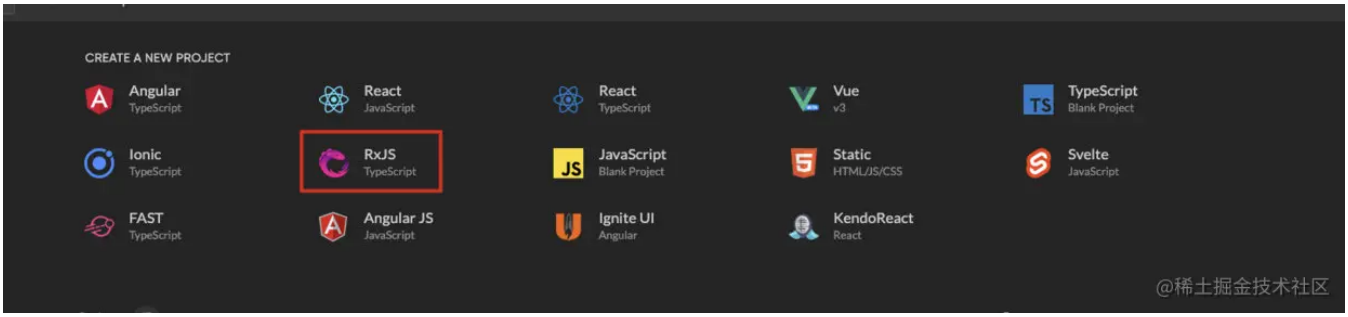
关注

最近一直在学习Rxjs，在学习过程中，踩了不少坑，也发现网上的文章都或多或少存在些问题，要么是内容过时了（现在已经Rxjs6），要么就是上来就讲原理（让我一脸懵逼），要么就是讲的不清楚，感觉踩了非常多的坑，学习曲线相对比较陡峭。理解基本概念之后再回头来看，其实并不难理解，但是确实走了很多弯路，所以整理下，同为打工人，能少踩点坑就少踩点坑把。

学习姿势

在我学习的时候，主要就是看官方文档，对一些含糊的地方，没来得及及时验证，导致半懂不懂，遇到实际文档需要反复查看。

最好的方法，就是边看边写，强烈推荐通过 stackblitz.com/ 中的Rxjs模版创建一个基础的版本，非常方便。下文的实例均是在此环境中运行。



坑点预警

1. 很多文章，会一上来就扯发布订阅、观察者模式，迭代器模式，增加学习成本也罢，关键是在实际学习中，总是想着这两个模式，反而会带来更高的理解成本。一个新玩意，你可能都还不会用，就开始尝试理解原理了，这怎么都讲不通。

2. 一个更常见的误解是认为 Rxjs 就是 addEventListener 那样的订阅者模式，subscribe 这个方法名也很有误导性。碰到subscribe，抛开脑子里的发布订阅，看实例，理解其本质即

可，不要纠结于叫法。这一点让我久久不能入门，其实这时候，抛开已有知识，直接去学习新知识，才是最佳的方法。

3. 官网文档中提到了非常多的难以理解的概念，会着重将设计思路，什么pull， push，相信我，这些等你真正用了之后，再回过头来看，一目了然，但是一开始要是陷进去了，那就难过了

4. 纠结在操作符上，Rxjs可以看作是流的lodash库，总不能一开始就看懂lodash里面的所有方法吧，所以不要纠结在操作符上，遇到了，查一下，看一看理解了就好

以上就是比较坑的点，在后续的学习中把避免这些坑，能少走很多弯路。

Rxjs介绍

Rxjs官方是这样说的: Think of RxJS as Lodash for events. 把Rxjs想像成针对events的lodash，也就是说，Rxjs本质是个工具库，处理的是事件。这里的events，可以称之为流。

那么流是指什么呢？举个例子，代码中每1s输出一个数字，用户每一次对元素的点击，就像是在时间这个维度上，产生了一个数据集。这个数据集不像数组那样，它不是一开始都存在的，而是随着时间的流逝，一个一个数据被输出出来。这种异步行为产生的数据，就可以被称之为一个流，在Rxjs中，称之为ovservalbe（抛开英文，本质其实就是一个数据的集合，只是这些数据不一定是一开始就设定好的，而是随着时间而不断产生的）。而Rxjs，就是为了处理这种流而产生的工具，比如流与流的合并，流的截断，延迟，消抖等等操作。

理解基本定义: observable, observer, subscription

可以通过如下的方式构建一个最基础的流，500ms时输出一个数组[1,2,3]，1s时输出一个对象{a: 1000}， 3s时，输出'end'， 然后在第4s终止该流。

```js
import { Observable } from "rxjs";

// stream$尾部的$是表示当前这个变量是个ovservable
const stream$ = new Observable(subscriber => {
  setTimeout(() => {
    subscriber.next([1, 2, 3]);
  }, 500);
  setTimeout(() => {
    subscriber.next({ a: 1000 });
```

```
  }, 1000);
  setTimeout(() => {
    subscriber.next("end");
  }, 3000);
  setTimeout(() => {
    subscriber.complete();
  }, 4000);
});

// 启动流
const subscription = stream$.subscribe({
  complete: () => console.log("done"),
  next: v => console.log(v),
  error: () => console.log("error")
});
// output
// [1,2,3]  // 500ms时
// {a:1000} // 1000ms时
// end // 3000ms时
// done // 4000ms时
```

在上述的代码中，通过`new Observalbe(fn)`定义了一个流，又通过`stream$.subscribe(obj)`启动了这个流，当500ms后，执行了```subsciber.next([1,2,3])，此时，通过传入的obj.next```方法输出了该值。

这里有几个点：

1. subscribe不是订阅，而是启动这个流，可以看到，subscribe后，才会执行next方法
2. 构建observable的时候，会有一个subscriber.next，这里就是控制这个流中数据的输出。

这里还有几个问题：

1. 能不能多次启动流，如果可以，那么多次启动时，相互之间的输出会不会干扰?
2. 既然有了启动流，那么能不能关闭流?

对于第一个问题，先上答案，可以多次启动，多次启动的流之间是相互独立的。可以通过下面这个例子验证：

```
js 复制代码

import { Observable } from "rxjs";
// 记录时间
const now = new Date().getTime();

```

```js
// 创建流
const stream$ = new Observable(subscriber => {
  setTimeout(() => {
    subscriber.next([1, 2, 3]);
  }, 500);
  setTimeout(() => {
    subscriber.next({ a: 1000 });
  }, 1000);
  setTimeout(() => {
    subscriber.next("end");
  }, 3000);
  setTimeout(() => {
    subscriber.complete();
  }, 4000);
});

// 启动流
const subscription1 = stream$.subscribe({
  complete: () => console.log("done"),
  next: v => console.log(new Date().getTime() - now, "ms stream1", v),
  error: () => console.log("error")
});

//  延时1s后，启动流
setTimeout(() => {
  const subscription2 = stream$.subscribe({
    next: v => console.log(new Date().getTime() - now, "ms stream2", v)
  });
}, 1000);

// output
// 506ms stream1 [1, 2, 3]
// 1002ms stream1 {a: 1000}
// 1505ms stream2 [1, 2, 3]
// 2009ms stream2 {a: 1000}
// 3002ms stream1 end
// done
// 4006ms stream 2 end
```

上面这个例子在最初启动了流1，延时1s后启动了流2，可以从输出看到，两个流的输出其实是相互独立的，而实际上也是这样设计的，流与流相互独立，互不干扰。

对于问题2，看到上述的写法应该能猜到，stream$.subscribe()的返回值subscription上存在一个方法unsubscribe，可以将流停止。演示代码如下：

```js
import { Observable } from "rxjs";

const now = new Date().getTime();

const stream$ = new Observable(subscriber => {
  setTimeout(() => {
    subscriber.next([1, 2, 3]);
  }, 500);
  setTimeout(() => {
    subscriber.next({ a: 1000 });
  }, 1000);
  setTimeout(() => {
    subscriber.next("end");
  }, 3000);
  setTimeout(() => {
    subscriber.complete();
  }, 4000);
});

// 启动流
const subscription = stream$.subscribe({
  complete: () => console.log("done"),
  next: v => console.log(v),
  error: () => console.log("error")
});

// 1s后，关闭流
setTimeout(() => {
  subscription.unsubscribe();
}, 1000);
// output
// [1,2,3]  // 500ms时
// {a: 1000} // 1000ms时
```

上述代码在1000ms时，执行了`subscription.unsubscribe()`，从而终止了该启动中的流，后续的输出都不会触发next函数，但是这并不意味着后续3000ms定时器，和4000ms定时器的解除，该定时器的回调依旧会执行，只是因为流已经关闭，不会执行next的回调。

以上，就是一个流的创建，启动和停止。在这里面，有好几个变量，重新整理下代码，如下：


```js
import { Observable } from "rxjs";

// 流的创建
const stream$ = new Observable(subscriber => {
```

```
  setTimeout(() => {
    subscriber.next([1, 2, 3]);
  }, 500);
});

// 如何消费流产生的数据，observer
const observer = {
  complete: () => console.log("done"),
  next: v => console.log(v),
  error: () => console.log("error")
};

// 启动流
const subscription = stream$.subscribe(observer);

setTimeout(() => {
  // 停止流
  subscription.unsubscribe();
}, 1000);
```

上述过程中，涉及到了3个变量

1. `stream$`，对应到Rxjs中，就是一个observable，单纯从英文翻译到中文的含义来看，基本很难理解。但是它的本质其实就是一个随时间不断产生数据的一个集合，称之为流更容易理解。而其对象存在一个subscribe方法，调用该方法后，才会启动这个流（也就是数据才会开始产生），这里需要注意的是多次启动的每一个流都是独立的，互不干扰。
2. `observer`，代码中是`stream$.subscribe(observer)`，对应到Rxjs中，也是称之为observer，从英文翻译到中文的含义来看，也很难理解。从行为上来看，无非就是定义了如何处理上述流产生的数据，称之为流的处理方法，更容易理解
3. `subscription`，也就是`const subscription = stream$.subscribe(observer);`，这里对应到Rxjs，英文也是称之为subscription，翻译过来是订阅，也是比较难以理解，其实它的本质就是暂存了一个启动后的流，之前提到，每一个启动后的流都是相互独立的，而这个启动后的流，就存储在subscription中，提供了unsubscribe，来停止这个流。

简单理解了这三个名词`observable, observer, subscription`后，从数据的角度来思考：observable定义了要生成一个什么样的数据，其subscribe方法，接收一个observer（定义了接收到数据如何处理），并开始产生数据，该方法的返回值，subscription, 存储了这个已经开启的流（暂时没想到啥好的中文名），同事具有unscbscribe方法，可以将这个流停止。整理成下面这个公式：

> Subscription = Observable.subscribe(observer) observable: 随着时间产生的数据集合，可以理解为流，其subscribe方法可以启动该流 observer: 决定如何处理数据

> subscription: 存储已经启动过的流，其unsubscribe方法可以停止该流

操作符

操作符对于Rxjs可以说是非常重要，但是对于初学者，可以先不沉浸于理解一个一个的操作符，先理解上文的概念更为重要，故这里不做详细的介绍，在实战中遇到了去官网查即可。

Subject

Subject也是Rxjs中比较重要的概念，从英文上不太好理解，直接上代码：

```js
import { Subject } from "rxjs";

// 创建subject
const subject = new Subject();

// 订阅一个observer
subject.subscribe(v => console.log("stream 1", v));
// 再订阅一个observer
subject.subscribe(v => console.log("stream 2", v));
// 延时1s再订阅一个observer
setTimeout(() => {
  subject.subscribe(v => console.log("stream 3", v));
}, 1000);
// 产生数据1
subject.next(1);
// 产生数据2
subject.next(2);
// 延时3s产生数据3
setTimeout(() => {
  subject.next(3);
}, 3000);
// output
// stream 1 1 //立即输出
// stream 2 1 //立即输出
// stream 1 2 //立即输出
// stream 2 2 //立即输出
// stream 1 3 //3s后输出
// stream 2 3 //3s后输出
// stream 3 3 //3s后输出
```

可以看到，Subject的行为和发布订阅模式非常接近，subscribe去订阅，next触发。事件的订阅通过subscribe，事件的触发使用next，从而实现一个发布订阅的模式。可以说，笔者本人是看到这个Subject，终于和已有的知识体系打通，之后才重新阅读官方文档，才算是弄懂了点皮毛。当然，subject还有别的用法，此处不再详细介绍

总结

讲了这么多，其实整个Rxjs的内容还非常多，本文的初衷只是坑点记录，帮助未入门的同学更快把握到精髓，在看官网的文档前读一读，比起上来就理解几个英文概念，能少走不少弯路。流启动、流的停止等都是笔者个人的理解，并非统一的叫法，仅仅是方便理解而起的，不要太过在意。

本人也还在学习中，共勉。

参考文章

1. blog.crimx.com/2018/02/16/...
2. blog.jerry-hong.com/series/rxjs...

标签： 前端 RxJS