RxJs--基础概念篇

《初识RxJs--基础铺垫篇》介绍了一些RxJs的一些基础概念，以及它的一个核心思想。今天正式进入RxJs库的学习，了解一下它的基础概念。

依据官网，我列出几个RxJs所需要了解的几个关键概念点，分别是Observable（可观察对象）、Observer（观察者）、Subscription（订阅）、Schedulers（调度器）、Subjects（主体）以及Operations（操作符）。今天首要说的是Observable（可观察对象）和Observer（观察者）以及Subscription（订阅）。

Observable（可观察对象），是一个可调用的未来值或者事件的集合。在《初识RxJs--基础铺垫篇》我们说过RxJs的一个核心思想就是观察者模式，那么Observable（可观察对象）就是数据的生产者，它拥有一个甚至多个值，并且在他有变化的时候，会主动推送消息给数据的消费者。我们在《初识RxJs--基础铺垫篇》介绍过两种数据传递交流的模式，Observable（可观察对象）就是典型的push推送模式。那么它与函数的相同点就在于，它可以像函数一样，同步的去传递一个值给数据的消费者，而不同点则是Observable（可观察对象）可以返回多个值，直到推送完成，而函数只能传递一个值给数据的消费者。

对于一个Observable（可观察对象）来说，最核心的关键点在于create Observable（创建可观察对象）、subscript Observable（订阅可观察对象）、execute Observable（执行可观察对象）以及dispose Observable（销毁可观察对象）。以下通过一个例子简单了解一下。

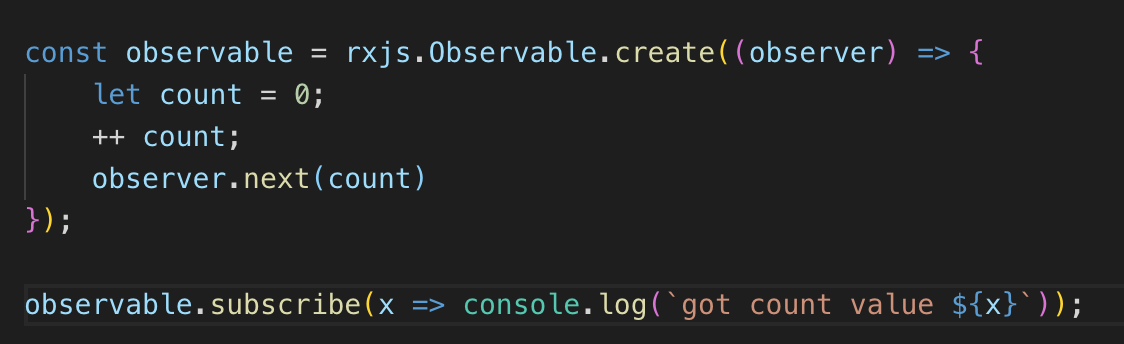
 上图是一个Observable的简单创建（我是通过script标签引入cdn的RxJs的Js文件），我们来分析一下。

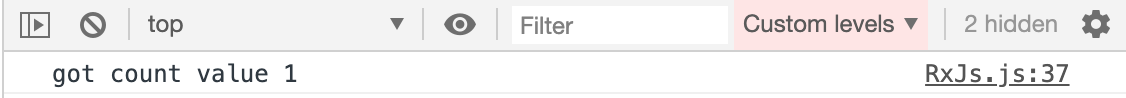
首先通过RxJs下的Observable对象的create方法创建一个Observable，create方法也被称作为创建操作符，操作符的概念，我们后面介绍。同时传入一个订阅函数，在订阅函数的形参对象中存在一个next方法，方法接收的参数，则是要发送给数据消费者的数据，在这个订阅函数中，可以拥有多个Observer.next()方法，每次创建一个next()方法，数据消费者都会收到对应的数据。除此以外，此函数下的形参对象还有其他几种方法，分别是error()方法，它意味着我们返回的是一个错误信息，还有一种方法是complete()方法，它意味着我们的数据推送完毕。在调用了error()方法和complete()方法后，数据将不再进行推送。

Observable.create()方法会返回一个Observable对象，通过这个对象下的subscribe()方法，我们可以订阅到Observable这个对象集合发送出来的数据，同时，subscribe()方法也会为我们返回出来一个订阅对象，在这个对象下，我们可以通过unsubscribe()方法去取消对Observable数据集合的订阅，停止接收Observable传递的数据。在我们取消订阅后，Observable数据集合发送数据的功能及被销毁，不会再执行。最终的一个结果如下图。

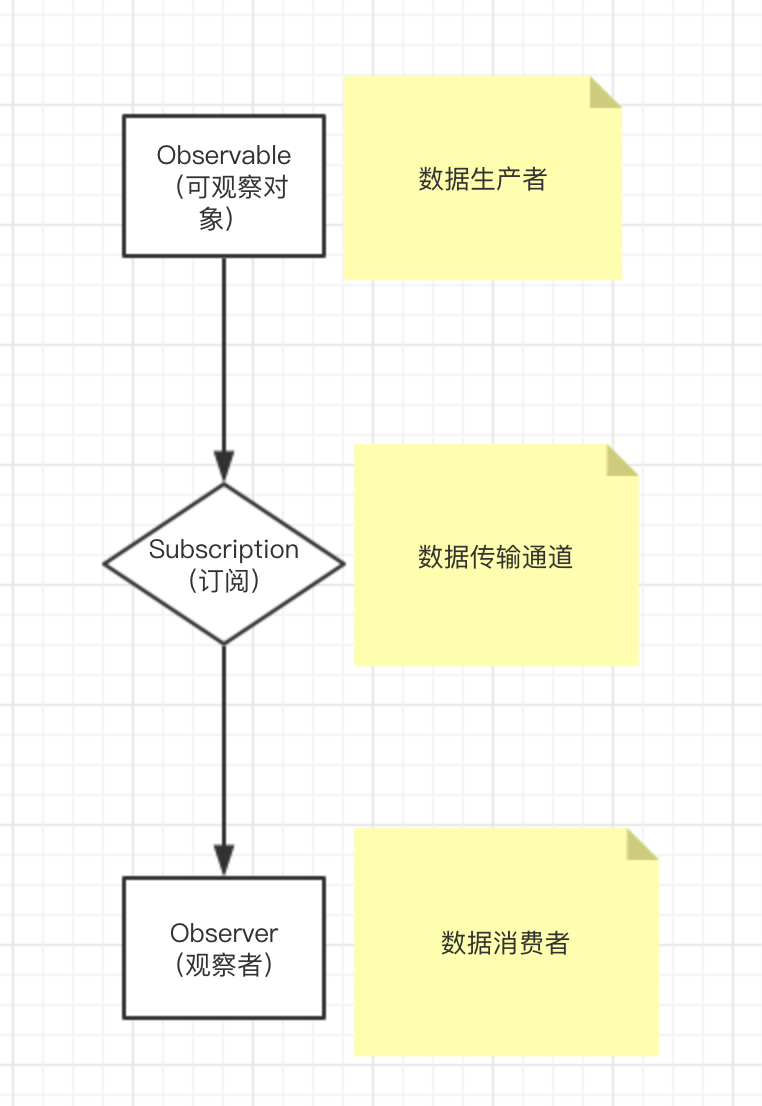
 这就是一个Observable（可观察对象）的核心关键点，同学们，是不是觉得很简单？

接着，我们来说一下Observer（观察者）。它是我们Observable（可观察对象）推送的数据的消费者，并且它是一组回调函数的集合，没一个回调函数都对应着我们上面说到的Observable对象的订阅方法中的三种状态，也就是next、error和complete.在Observer（观察者）对象的回调函数中，我们可以不需要全部写上这三种状态，缺省的状态会自动帮我们补全。同样的，如果我们在订阅的时候，只传入了一个回调函数，而不是一个回调函数集合时，RxJs会默认将我们的第一个回调函数当做Observable（可观察对象）的next()方法处理。举一个简单例子。



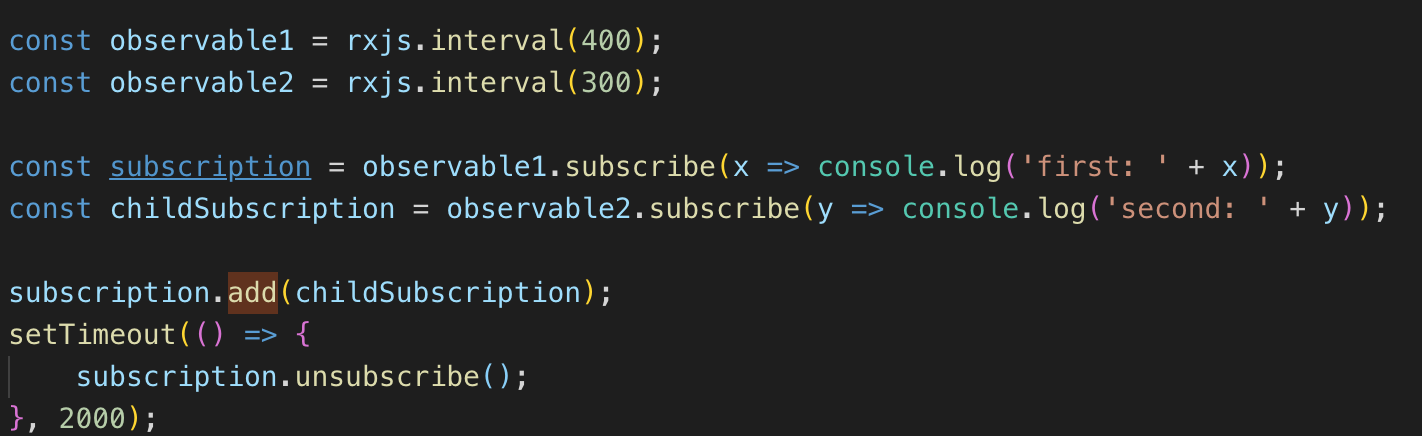


说完了Observable（可观察对象）和Observer（观察者）后，再来理解一下Subscription（订阅）。在前面的例子中，我们也一直在用到Subscription（订阅），它是Observable（可观察对象）和Observer（观察者）的一个桥梁。如果没有Subscription（订阅），Observable（可观察对象）这个数据生产者不会讲数据传递出去，而Observer（观察者）这个数据消费者也不会接收到数据。它们之间的关系如下图：



通过这张图，我们可以知道Subscription是数据的传输通道。在最初的Observable（可观察对象）的创建例子中，我们提到在订阅了Observable（可观察对象）后，会返回给我们一个订阅对象，我们通过订阅对象的unsubscribe()方法，销毁掉这个通道，停止我们的数据传输。除了这个方法，还有两个方法，分别是add()方法，可以将多个订阅添加到一个订阅中，这样我们在调用被添加其他订阅后的订阅对象中的unsubscribe()方法时，会将我们所有的订阅都取消掉。

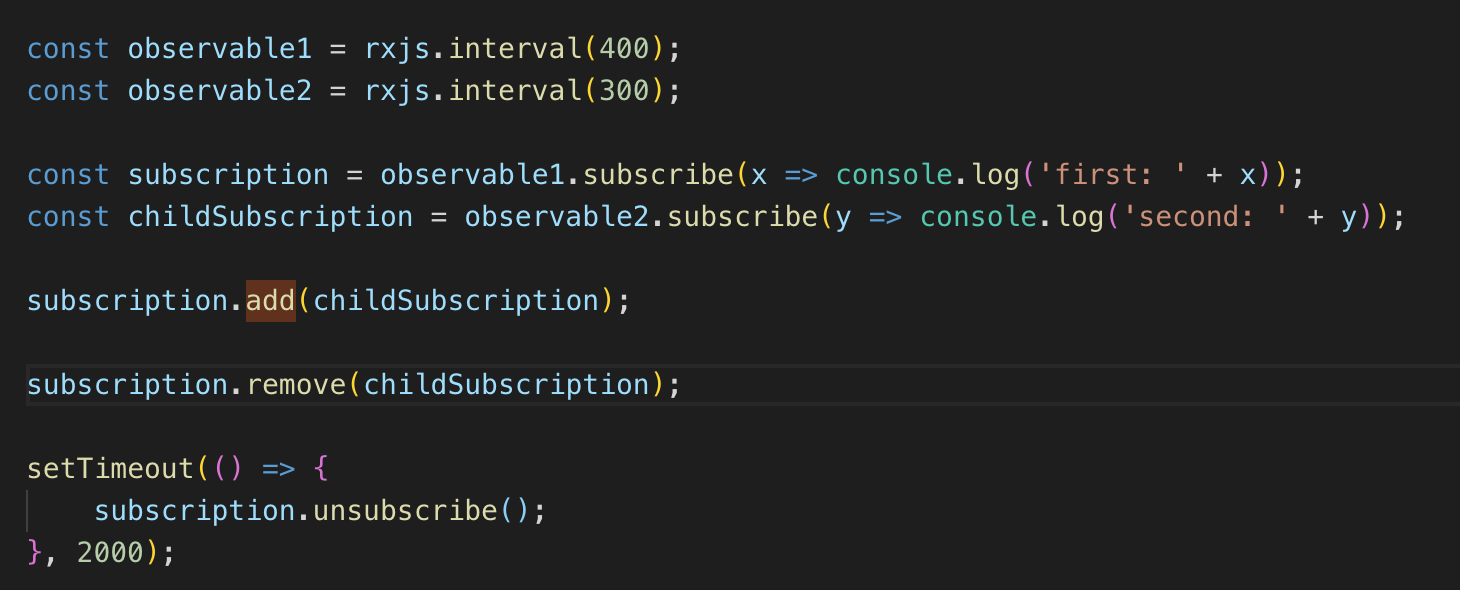
有添加，自然有移除，那就是remove()方法。这个方法就是将添加父Subscription对象后的子Subscription对象移除，从而在调用unsubscribe()方法时，被移除的订阅通道不会被销毁掉。简单看一个列子：

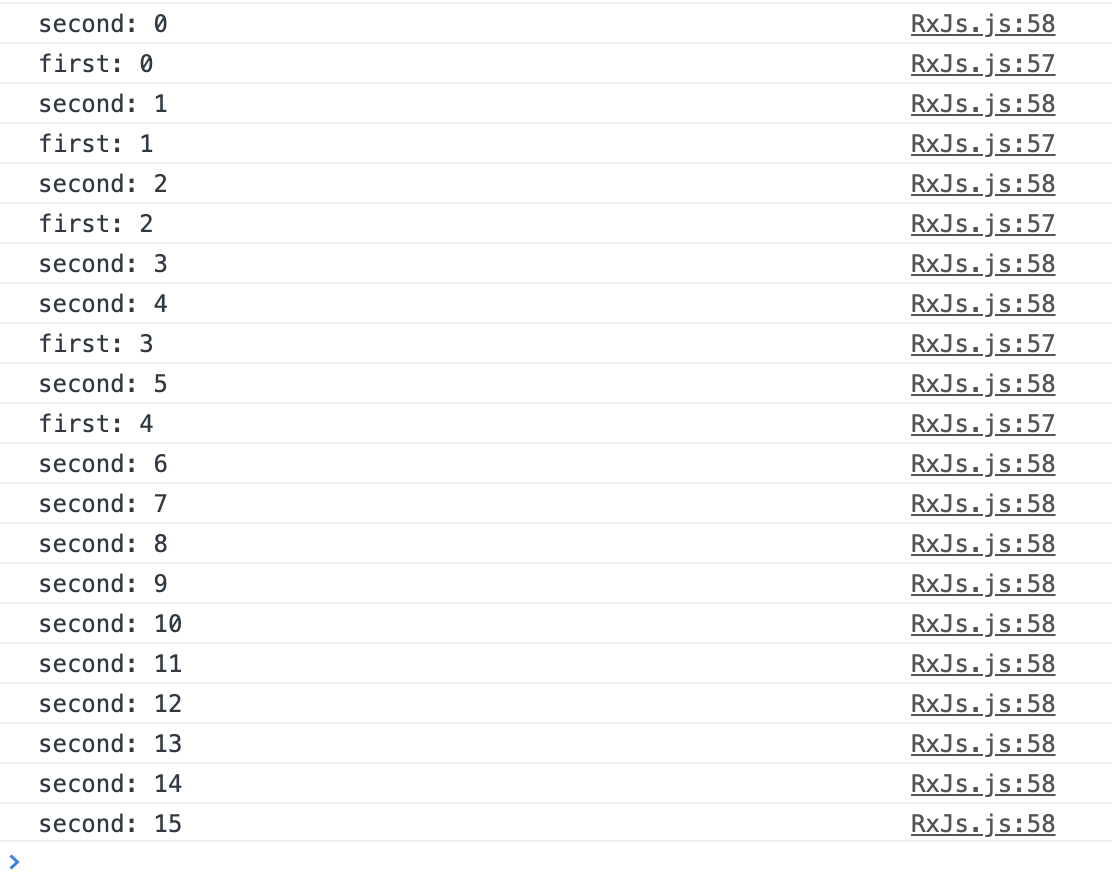


在这个例子中，我们通过RxJs的interval操作符，按照一定时间间隔发送一个数字序列。数字序列从0开始。也就是在400毫秒后，我们将会打印一个observable1对象将会发送出一个值为0，在300毫秒时，observable2对象会发送出一个值为0.之后定时输出加1后的值，直到2秒后，我们停止订阅。由于我们使用了add方法，将observable2的订阅对象添加到了observable1中，因此它们会同时停止消息的发送。结果如下图：



那么我们稍作修改，加入remove()方法后，看一下结果的变化。如图：





从方法图和结果图中我们可以看到，首先我们使用add方法添加了observable2的订阅对象值到observable1的订阅对象中，之后我们又调用observable1的订阅对象下的remove()方法，移除了observable2的订阅对象，造成的结果就是，在2秒后，obsevable1的消息推送结束了，而observable2的消息推送还在继续，也就是控制台中的”second: ...”

到此，一个RxJs从数据生产到数据消费的过程就结束了。后面面我们将介绍一下其他的辅助概念。接着期待吧~哈哈！如果觉得不错，点个赞咯！