## **<https://zhuanlan.zhihu.com/p/51767778>**

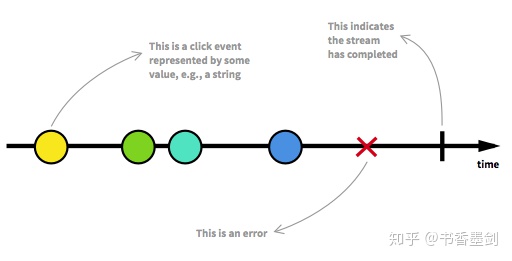
## **什么是响应式编程**

**响应式编程是针对异步数据流的编程方式**

从这个角度来说，响应式编程并不是一个新的概念。事件总线，或者典型的点击事件都是一个异步的事件流，你可以对其进行观察或者做一些操作。响应式编程也是一个类似的概念。你可以对任何事情创建一个流，而不仅仅是类似“点击”、“悬停”这一类的事件。流的创建是很轻易的，任何东西都可以是流，比如说：变量、用户输入、属性、缓存、数据结构等等。举个例子，想象你的 Twitter 摘要将会是一个类似点击事件的流，你可以监听它或者对其进行响应。

**重要的是，你会拥有强力的函数工具去合并、创建、过滤这些流。**这些特性是“函数式编程”带来的好处。一个流可以作为其他数据流的输入，甚至也可以将多条流作为其他流的输入。你可以合并2条流。也可也从一条流*过滤*出一条只有你需要的事件的流。你还可以从一条流展开数据给另外一条流。

既然流对于响应式编程如此重要，就让我们认真研究一下它们。还是从我们熟悉的“点击一个按钮”事件流开始吧。



一个流是按时间来排列的一系列事件队列。它可以发出三种不同的事物：值、错误、“已完成”信号。考虑下“已完成”的使用场景，比如说当包含这个按钮的窗口或者试图区域关闭了。

我们只能 **异步(asynchronously)** 地捕获这些发出的事件，并且对发出的值、错误、以及‘已完成’定义不同的函数，来执行不同的操作。有些时候你可以省略后面两种情况，只对关心的值定义函数。“监听”这些流的操作被称为 **订阅(subscribing)**， 而被定义的函数被称为 **观察者(observers)**，这些流被称为 **可观察对象(observable)**。这就是观察者设计模式(Observer Design Pattern)。

我们也可以用 ASCII 码来画出这个图，以便于在一些教程中使用：

--a---b-c---d---X---|->

a, b, c, d are emitted values

X is an error

| is the 'completed' signal

---> is the timeline

这个图已经非常的熟悉了，为了不让你觉得无聊，让我们做一些新的事情：我们打算创建一些新的点击事件流来改变原来的点击事件流。

首先，让我们创建一个计数流来记录按钮被点击了多少次。在通常的响应式库里，每个流都会有一些函数附加在上面，例如：map, fileter, scan 等等。当你调用这些函数中的某一个的时候，例如：clickStream.map(f)， 它会返回一个基于点击事件流的**新的流**。这个操作不会改变原来的点击事件流。这种属性被称为**不可变性**，它使得响应流用起来非常舒服。这样可以允许我们使用链式函数，例如：clickStream.map(f).scan(g)：

clickStream: ---c----c--c----c------c-->

vvvvv map(c becomes 1) vvvv

---1----1--1----1------1-->

vvvvvvvvv scan(+) vvvvvvvvv

counterStream: ---1----2--3----4------5-->

map(f) 函数替换了（到新的流）你在函数f中提供的每一个返回值。在我们的例子中，我们将每个点击展开成了数字1，scan(g)函数将之前的值累加到了流里面，从x = g(accumulated, current)中制造出了新的值，在这个例子中，其中 g 是一个简单的累加函数。然后，counterStream 在 click 发生的时候响应了数字和。

要看看响应式编程的真正力量，让我们假设你想要一个“双击”的事件流。为了让它更加有趣，假设还需要考虑一个 “三次点击”的新的事件流，或者通常来说叫做“多次点击”（2次或更多）。深呼吸并且想象接下来的事情和所有的状态。我打赌这个听起来很复杂并且需要许多变量来保持状态并且还有各种索索的时间间隔。

但是，在响应式编程中，这将变得十分简单。事实上，所有的逻辑仅仅只需要[四行代码](https://link.zhihu.com/?target=http://jsfiddle.net/staltz/4gGgs/27/" \t "https://zhuanlan.zhihu.com/p/_blank)。

让我们暂时忽略代码。用图片的方式来理解并创建流，不管你是初学者还是专家。

