**<https://blog.strongbrew.io/safe-http-calls-with-rxjs/>**

介绍  
这篇文章都是关于如何确保HTTP调用不会在坏的连接上终止，因为奇怪的是，当使用RxJS时，404响应会导致应用程序终止。

还记得RxJS可观测事件有三种类型吗?

next: passing in a new value into the observable

error: when an error occurs

complete: When the observable is completed

我们不应该忘记错误事件实际上会停止可观察对象。它将不复存在。

你可能会说:“这不是那么糟糕，我们只是在每次想要获取数据时创建一个新的。”

当您以被动的方式处理应用程序时，这个场景可能会有问题:想象一下，我们希望检索每个输入值的结果的typeahead搜索。我们有一个searchterms的可观察对象，我们使用一个switchMap为每个值触发HTTP请求。

// 这个可观察对象包含这些值

// 用户正在搜索的内容

// 超时

const searchTerm$: Observable<string>;

//当term接收到一个新值时…

//去取一些数据

const results$ = searchTerm$.pipe(

switchMap(term => fetchData(term))

)

// 订阅可观察对象，开始倾听

results$.subscribe((response: Result[]) => {

console.log(response);

})

这一切都很好，直到一个错误发生在观察。这可能来自坏连接、服务器宕机或HTTP请求期间发生的任何错误(500、404，……)。如果用户的连接坏了，可能导致404错误，观察将停止，应用程序将被破坏。用户可以搜索任何他或她想要的结果，HTTP调用不会再发生。

catchError

我们可以使用catchError操作符，它将基本上为我们捕获错误，并返回一个全新的可观察对象(包含错误)。该可观察对象将错误作为一个值，而不是再次将其作为错误抛出。这样我们就可以向用户展示一个很好的信息。这可能看起来像这样:

const results$ = searchTerm$.pipe(

switchMap(term =>

fetchData(term)*.pipe(*

*// return an observable with the error inside*

*catchError(e => of(e))*

*)*

)

)

results$.subscribe(

*(response: Result[] | HttpErrorResponse) => {*

*if(response instanceof HttpErrorResponse){*

*console.log('oh no:(');*

*return;*

*}*

*console.log('do something fancy');*

*});*

)

*Ps:我并不是说这是捕捉错误的最佳方法。我只是向你介绍一些基本知识。*

请注意，catchError操作符应用于fetchData()返回的可观察结果，而不是作为第一个管道的第二个操作符添加。从被观察对象接收到错误的那一刻起，它就会死亡……这就是为什么在被观察对象内部捕获错误非常重要的原因。

retryWhen

好的,太棒了!应用程序不会再中断，但是现在设想一下下面的场景:我们的用户坐在火车上，开车穿过隧道。连接中断了几秒钟，用户不会得到结果。

我们可以通过让RxJS重试几次来解决这个问题

const results$ = searchTerm$.pipe(

switchMap(term =>

fetchData(term).pipe(

retryWhen(e$ => e$.pipe(

// 2秒后再试

delay(2000),

// 5次后停止尝试

take(4)

)

// 仍然保持可观察的存在

// 前5次失败了

catchError(e => of(e))

)

)

)

你可以在*这里*找到更多关于重试的信息

*Using the online event*

尽管这是一个很好的解决方案，但仍有改进的空间。一个不错的选择是使用HTML5的在线事件告诉浏览器当用户重新获得互联网连接时重试。它比以前更短，而且非常优雅

const results$ = searchTerm$.pipe(

switchMap(term =>

fetchData(term).pipe(

retryWhen(() => fromEvent(window, 'online'))

//仍然保持可观察的存在

//服务器将返回一个不同的

// HTTP错误

catchError(e => of(e))

)

)

)

结论  
RxJS给我们很大的控制HTTP调用!如果我们知道错误处理是如何工作的，就可以轻而易举地将HTTP调用提升到下一个层次。这不仅适用于typeahead搜索，也适用于我们将现有流与受错误影响的流(如HTTP)组合在一起的每个可观察对象。例如:这也可以发生在NgRx效果或angular路由器。  
在那里，我告诉过你它会很短，我希望你学到了一些东西。  
另外，请务必检查本文:使用指数后退时RxJS的强大功能