# [谜之RxJava （一） —— 最基本的观察者模式](https://segmentfault.com/a/1190000004049490)

## Demo

Observable.create(new Observable.OnSubscribe<String>() {  
 @Override  
 public void call(Subscriber<? super String> subscriber) {  
 subscriber.onNext("hello");  
 }  
}).subscribe(new Subscriber<String>() {  
 @Override  
 public void onCompleted() {  
 }  
 @Override  
 public void onError(Throwable e) {  
 }  
 @Override  
 public void onNext(String s) {  
 Log.*d*("rx", s);  
 }  
});

这段代码产生的最终结果就是在Log里会出现hello。

看下这段代码的具体流程吧。  
这里有2个函数create和subscribe，我们看看create里面干了啥。

## OnSubscribe对象

public final static <T> Observable<T> create(OnSubscribe<T> f) {  
 return new Observable<T>(hook.onCreate(f));  
}  
// constructor  
protected Observable(OnSubscribe<T> f) {  
 this.onSubscribe = f;  
}

这里的hook是一个默认实现，里面不做任何事，就是返回f。我们看见create只是给Observable的onSubscribe赋值了我们定义的OnSubscribe。

## Subscriber对象

来看下subscribe这个函数做了什么事

public final Subscription subscribe(Subscriber<? super T> subscriber) {  
 return Observable.subscribe(subscriber, this);  
}  
  
private static <T> Subscription subscribe(Subscriber<? super T> subscriber, Observable<T> observable) {  
 // validate and proceed  
 if (subscriber == null) {  
 throw new IllegalArgumentException("observer can not be null");  
 }  
 if (observable.onSubscribe == null) {  
 throw new IllegalStateException("onSubscribe function can not be null.");  
 /\*  
 \* the subscribe function can also be overridden   
 \* but generally that's not the appropriate approach  
 \* so I won't mention that in the exception  
 \*/  
 }  
  
 // new Subscriber so onStart it  
 subscriber.onStart();  
  
/\*  
 \* See https://github.com/ReactiveX/RxJava/issues/216   
 \* for discussion on "Guideline 6.4: Protect calls  
 \* to user code from within an Observer"  
 \*/  
 // if not already wrapped  
 if (!(subscriber instanceof SafeSubscriber)) {  
 // assign to `observer` so we return the protected version  
 subscriber = new SafeSubscriber<T>(subscriber);  
 }  
  
 // The code below is exactly the same an unsafeSubscribe   
 // but not used because it would add a sigificent depth to alreay huge call stacks.  
 try {  
 // allow the hook to intercept and/or decorate  
 hook.onSubscribeStart(observable, observable.onSubscribe).call(subscriber);  
 return hook.onSubscribeReturn(subscriber);  
 } catch (Throwable e) {  
 // special handling for certain Throwable/Error/Exception types  
 Exceptions.*throwIfFatal*(e);  
 // if an unhandled error occurs executing the onSubscribe we will propagate it  
 try {  
 subscriber.onError(hook.onSubscribeError(e));  
 } catch (OnErrorNotImplementedException e2) {  
 // special handling when onError is not implemented ... we just rethrow  
 throw e2;  
 } catch (Throwable e2) {  
 // if this happens it means the onError itself failed   
 // (perhaps an invalid function implementation)  
 // so we are unable to propagate the error correctly and will just throw  
 RuntimeException r = new RuntimeException("Error occurred attempting to subscribe ["   
 + e.getMessage() + "] and then again while trying to pass to onError.", e2);  
 // *TODO could the hook be the cause of the error in the on error handling.* hook.onSubscribeError(r);  
 // *TODO why aren't we throwing the hook's return value.* throw r;  
 }  
 return Subscriptions.*unsubscribed*();  
 }  
}

我们看到，这里我们的subscriber被SafeSubscriber包裹了一层。

if (!(subscriber instanceof SafeSubscriber)) {  
 // assign to `observer` so we return the protected version  
 subscriber = new SafeSubscriber<T>(subscriber);  
}

然后开始执行工作流

hook.onSubscribeStart(observable, observable.onSubscribe).call(subscriber);  
return hook.onSubscribeReturn(subscriber);

默认的hook只是返回我们之前定义的onSubscribe，这里调用的call方法就是我们在外面定义的

new Observable.OnSubscribe<String>() {  
 @Override  
 public void call(Subscriber<? super String> subscriber) {  
 subscriber.onNext("hello");  
 }  
})

我们调用传入的subscriber对象的onNext方法，这里的subscriber是SafeSubscriber  
在SafeScriber中

public void onNext(T args) {  
 try {  
 if (!done) {  
 actual.onNext(args);  
 }  
 } catch (Throwable e) {  
 // we handle here instead of another method so we don't add stacks to the frame  
 // which can prevent it from being able to handle StackOverflow  
 Exceptions.*throwIfFatal*(e);  
 // handle errors if the onNext implementation fails, not just if the Observable fails  
 onError(e);  
 }  
}

actual就是我们自己定义的subscriber。 原来SafeSubscriber只是为了帮我们处理好异常，以及防止工作流的重复。

这是RxJava最最基本的工作流，让我们认识到他是怎么工作的。之后我们来讲讲其中的细节和其他神奇的内容。