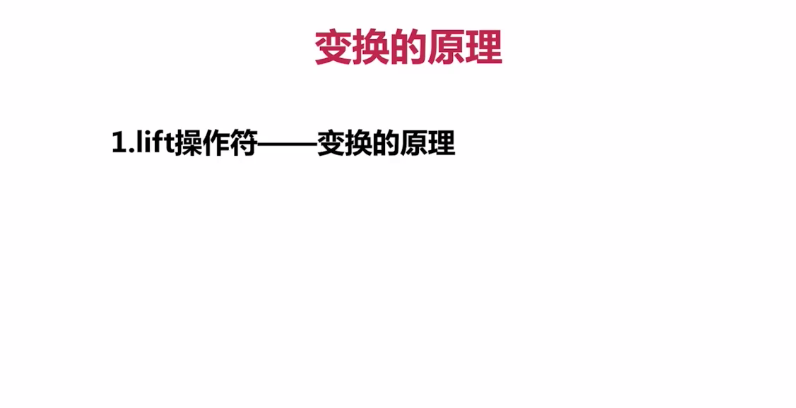
# 操作符导引：

操作符的概念：

# 

操作符变换的原理：

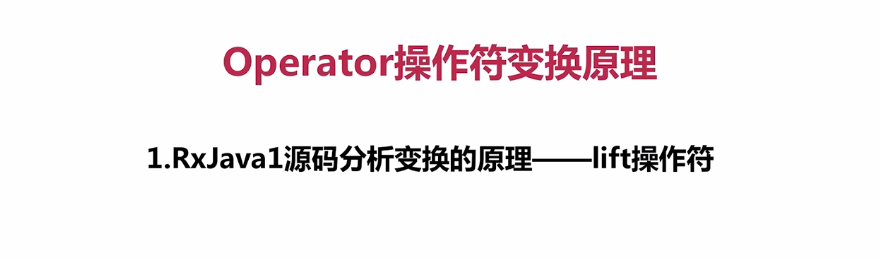


# RxJava1操作符部分：

注：

RxJava1源码分析所使用的版本是 1.1.6

RxJava1操作符变换原理：



lift操作符的步骤：



具体源码解析，结合实例分析源码，结合的小例子如下：

Observable.create(new Observable.OnSubscribe<String>() {

@Override

public void call(Subscriber<? super String> subscriber) {

if (!subscriber.isUnsubscribed()) {

subscriber.onNext("1");

subscriber.onNext("2");

subscriber.onCompleted();

}

}

}).map(new Func1<String, Integer>() {

@Override

public Integer call(String s) {

return Integer.parseInt(s);

}

}).subscribe(new Observer<Integer>() {

@Override

public void onCompleted() {

Log.d(TAG, "onCompleted");

}

@Override

public void onError(Throwable e) {

}

@Override

public void onNext(Integer s) {

Log.d(TAG, "onNext " + s + " instanceOf " + s.getClass());

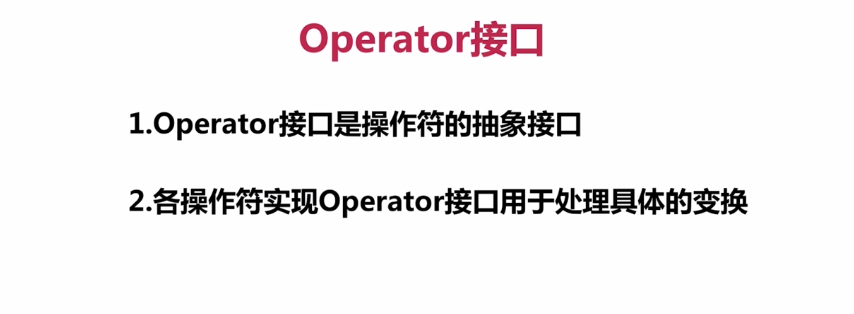
}

});

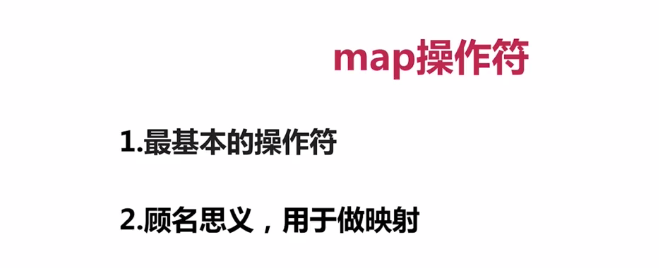
可以更好理解 lift操作符的步骤图中的逻辑。下面将模拟实现RxJava1的操作符变换部分。

RxJava1操作符变换实战：

RxJava1操作符实现中的源码符号抽取：







注：仿RxJava1操作符的实现代码部分，特别是父类，子类泛型关系需要再梳理。

类关系见课堂手记。

public <R> Caller<R> lift(final Operator<R, T> operator) {

return create(new OnCallLift<T, R>(onCall, operator));

}

public final <R> Caller<R> map(Func1<T, R> func) {

return lift(new MapOperator<T, R>(func));

}

比如上两处代码，调用map时，传入是<T,R>泛型，但调用lift时在Operator<R,T>中是<R,T>,但实际仍为<T,R>顺序，因为MapOperator在实现Operator时，顺序是反的。

实在不行，可使用带入具体类型的方法，其实也就是在使用时带入具体类型，进行分析。

具体代码参见下面目录：

<https://github.com/rongwenzhao/ImoocRxJavaStudy/commit/229e345d3aa8609daf950f7a0227e27c75a2787f>

提交内容：

1、RxJava1操作符变换实例demo；

2、仿RxJava1操作符变换效果以及仿操作符变换效果的使用。

3、更改RxJava1的版本信息，使之与案例版本一致。

# RxJava2操作符部分：

## RxJava2操作符源码分析，无背压版：

结合小例子分析，小例子如下：

Observable.

create(new ObservableOnSubscribe<String>() {

@Override

public void subscribe(ObservableEmitter<String> e) throws Exception {

if (!e.isDisposed()) {

e.onNext("1");

e.onNext("2");

e.onComplete();

}

}

}).

map(new Function<String, Integer>() {

@Override

public Integer apply(String s) throws Exception {

return Integer.parseInt(s);

}

}).

subscribe(new Observer<Integer>() {

@Override

public void onSubscribe(Disposable d) {

Log.d(TAG, "onSubscribe");

}

@Override

public void onNext(Integer value) {

Log.d(TAG, "onNext " + value);

}

@Override

public void onError(Throwable e) {

}

@Override

public void onComplete() {

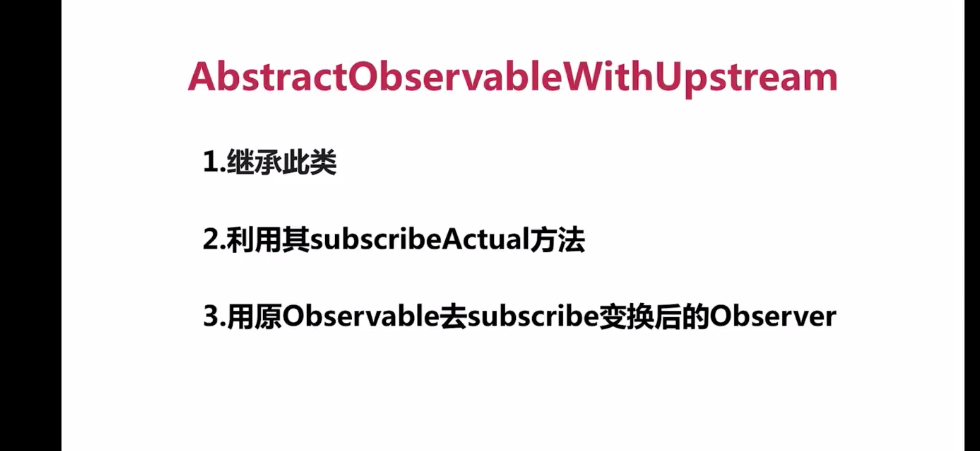
Log.d(TAG, "onComplete");

}

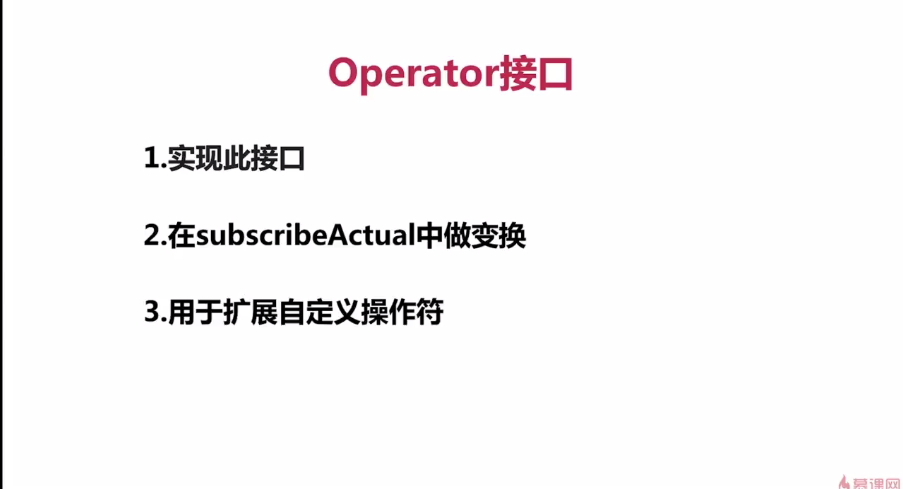
});

源码分析结论：

一、map等操作符实现原理：



二、lift操作符实现扩展自定义操作符：



在源码中为ObservableOperator接口。

注：

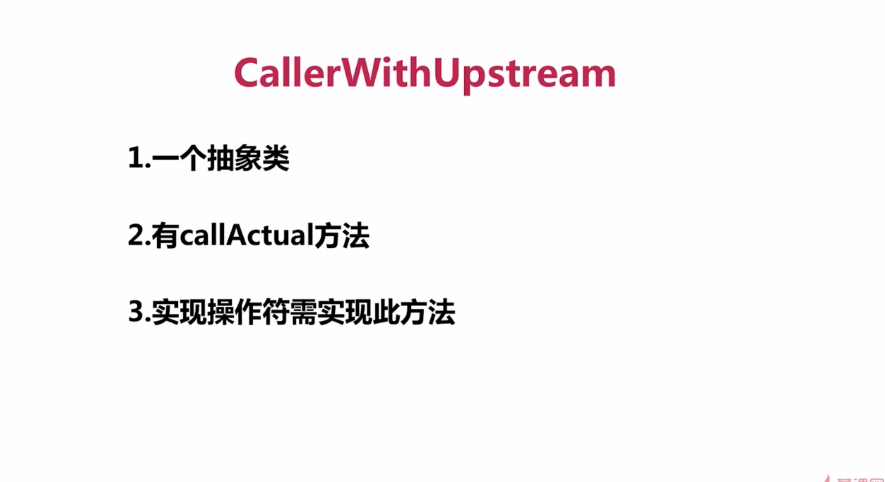
1. RxJava2中的lift实现，跟RxJava1中的lift效果基本一致；
2. RxJava2中，将map操作符单独分离出来，其他自定义扩展的操作符可通过使用lift方式，实现ObservableOperator接口来实现；
3. 具体使用，后续会说明。

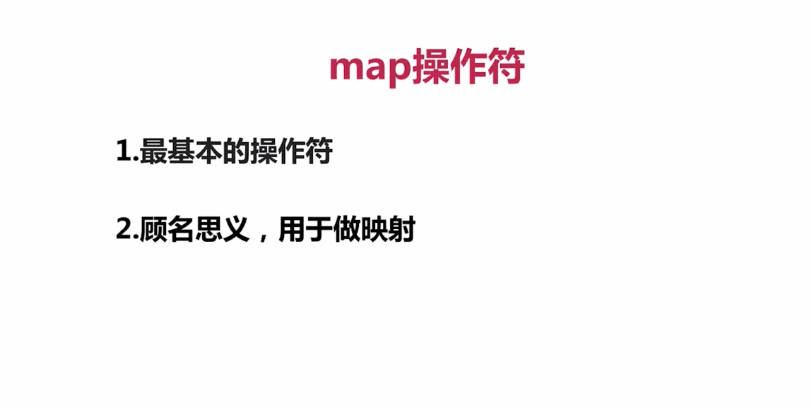
即：Lift是核心操作符，map只是分离出去的一种实现。

具体结合代码理解。

仿写：

一、模拟map操作符：





Map操作符实现代码见下条提交记录：

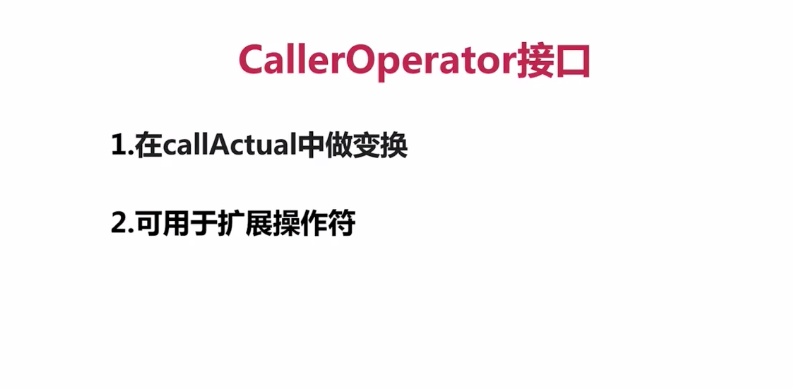
<https://github.com/rongwenzhao/ImoocRxJavaStudy/commit/4523a714fc407b13828c11a8a317b880cb45a680>

提交日志：

1、RxJava2无背压版操作符变换实例demo；

2、仿RxJava2无背压版操作符变换效果以及仿操作符变换效果的使用。

二、实现扩展自定义操作符：



扩展操作符实现及使用代码见下面的提交记录：

<https://github.com/rongwenzhao/ImoocRxJavaStudy/commit/979e3fc9eb7044833c08e850aada2ade2612bed4>

提交日志：

扩展RxJava2无背压情况下自定义操作符的实现及使用实例：

1、RxJava2无背压版lift操作符源码分析；

2、仿RxJava2无背压版lift操作符变换效果以及仿操作符变换效果的使用。

## RxJava2操作符源码分析，有背压版：

结合小例子分析，小例子如下：

Flowable.

create(new FlowableOnSubscribe<String>() {

@Override

public void subscribe(FlowableEmitter<String> e) throws Exception {

if (!e.isCancelled()) {

e.onNext("1");

e.onNext("2");

e.onNext("3");

e.onNext("4");

e.onNext("5");

e.onNext("6");

e.onComplete();

}

}

}, BackpressureStrategy.DROP).

map(new Function<String, Integer>() {

@Override

public Integer apply(String s) throws Exception {

return Integer.parseInt(s);

}

}).

subscribe(new Subscriber<Integer>() {

@Override

public void onSubscribe(Subscription s) {

Log.d(TAG, "onSubscribe");

s.request(Long.MAX\_VALUE);

}

@Override

public void onNext(Integer integer) {

Log.d(TAG, "onNext " + integer);

}

@Override

public void onError(Throwable t) {

}

@Override

public void onComplete() {

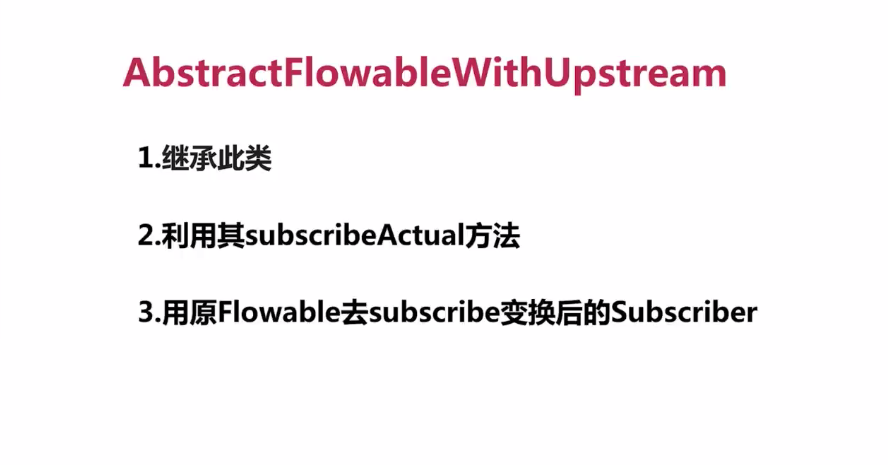
Log.d(TAG, "onComplete");

}

});

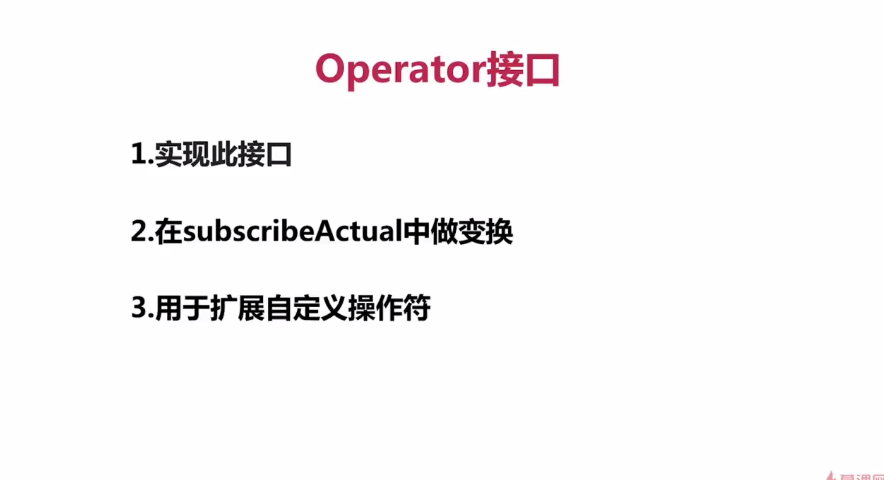
源码中关键类：

一、结合map操作符分析如下：



1. operator接口，用于扩展自定义操作符。

用到此功能的是lift操作符。

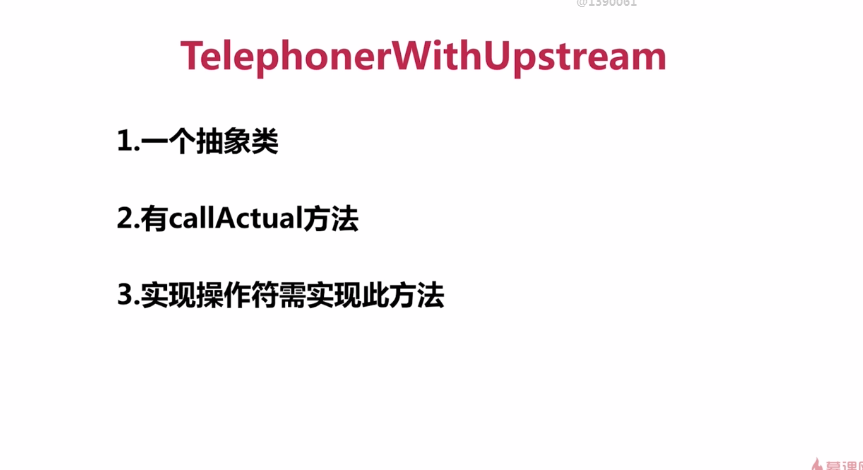


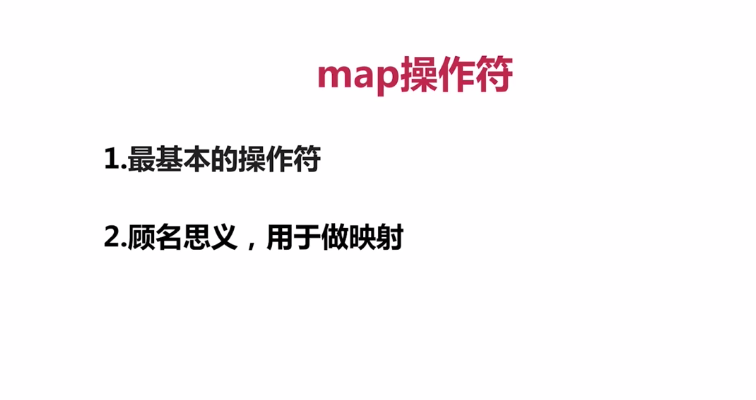
注：

1. 有背压版的operator在源码中为 FlowableOperator接口；
2. 我们自己可以通过使用lift操作符，通过实现FlowableOperator接口，来扩展我们自定义的变换操作符。

仿写实战：

一、仿写map操作符





代码提交如下：

<https://github.com/rongwenzhao/ImoocRxJavaStudy/commit/788cba7f1b4e9ecef96a2a8147d002f763f7795d>

提交日志：

1、仿写RxJava2有背压情况下的map操作符的实现；

2、仿写的map操作符的使用。

1. 扩展的操作符（重点是lift操作符）



通过lift操作符，结合实现TelephonerOperator接口，可实现操作符的扩展。

Lift操作符的实现代码在下面的一条提交记录中：

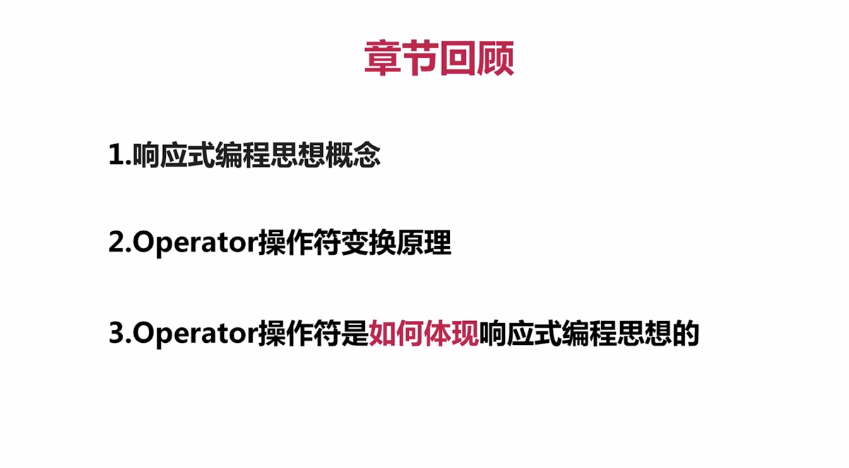
<https://github.com/rongwenzhao/ImoocRxJavaStudy/commit/3c442db74b5a2b61b69f43d54cac5ccd30379907>

提交日志：

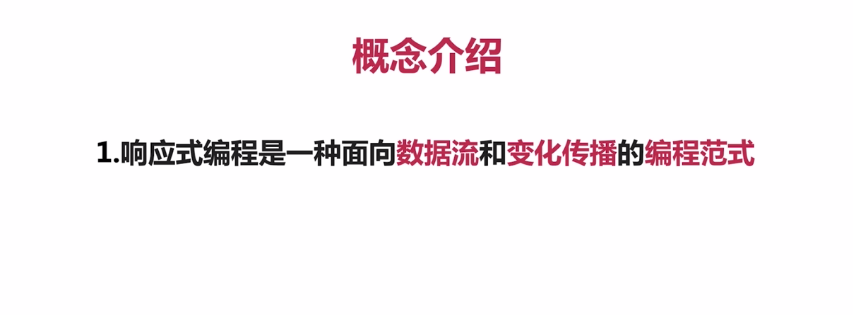
1、仿写RxJava2有背压情况下的lift操作符的实现；

2、仿写的lift操作符的使用，通过使用lift操作符达到与map操作符一样的效果。

# 章节回顾:



1、响应式编程思想概念：



1. operator操作符变换原理：

详见上文的细节，本章重点即为操作符变换。

1. operator操作符是如何体现响应式编程思想的：

