Observable -> Observable.create -> Observable.OnSubscribe -> Subscriber

Observable -> Observable.range -> Action1，依次发送range中指定的范围，如range(5, 3)，则从5开始发送3个递增数字：5、6、7

Obserable -> Observable.just -> Action1，每次订阅得到同一个Obserable中返回的数据

Obserable -> Observable.defer -> Func0 -> Action1，每次订阅得到新的Obserable中返回的数据

Obserable -> Observable.*from* -> Action1，一个一个的发送数组或集合中的数据

Observable -> Observable.*interval*(1, TimeUnit.***SECONDS***, Schedulers.*trampoline*()) -> Subscriber， 从0开始发送数据，每一秒发一个递增数字

Observable -> Observable.*just*(1,2,3).repeat(3) -> Action1,just会依次发送1、2、3，repeat(3)，则是把这个事情重复3次

Observable -> Observable.*timer*(1, TimeUnit.***SECONDS***, Schedulers.*trampoline*()) -> Action1，1秒后发送数字0

Observable -> Observable.*just*(1, 2, 3, 4, 5, 6, 7, 8, 9).buffer(2, 3) -> Action1，每满两个输出两个，然后隔3个，从第4个开始满两个又输出两个：输出为：1 2、4 5、7 8

Observable.*interval*(1, TimeUnit.***SECONDS***, Schedulers.*trampoline*()).buffer(3, TimeUnit.***SECONDS***) -> Action1

Observable.*just*(1, 2, 3, 4, 5, 6, 7, 8, 9).window(3)，把9个数字，每3个创建成一个新的Observable

Observable.*interval*(1000, TimeUnit.***MILLISECONDS***, Schedulers.*trampoline*()).window(3000, TimeUnit.***MILLISECONDS***) 每3秒发出一个小的Observable，小的Observable会输出其收到的2个或3个数据

Observable.*just*(1, 2, 3, 4, 4, 3, 2, 1).distinct()-> Action1，过滤掉重复的，这里只会输出1， 2， 3，4

Observable.*just*(1, 2, 3, 4, 4, 3, 2, 1).distinctUntilChanged()-> Action1，可以有重复的，但是边续的不会有重复，这里输出：1，2， 3，4， 3， 2， 1

Observable.*just*(1, 2, 3, 4, 4, 3, 2, 1).elementAt(0) -> Action1，获取第1个数据，输出1

Observable.*just*(1, 2, 3, 4, 4, 3, 2, 1).filter -> Func1 -> Action1 过滤Func1中符合条件的所有数据

Observable.*just*(1, 2, 3, 4, 5, 6, 7, 8, 9).first -> Func1 -> Action1 过滤出Func1中符合条件的第一条数据

Observable.*just*(1, 2, 3, 4, 5, 6, 7, 8, 9).first() -> Action1，只输出第1条数据：1

对应的有last，只输出最后一条，或者满足条件的最后一条

Observable.*merge*(Observable.*just*(1, 2, 3), Observable.*just*(4, 5, 6)) 把两个流合并成一个，但是数据的顺序没有保证，如果想保证顺序可使用concat操作符

delay(2000, TimeUnit)，将Observable延迟2秒后才发送数据，delaySubscription与其功能类似，它的实现是延迟注册Subscriber，也相当于是延迟发送数据了

all，判断所有数据是否都满足条件，都满足返回true，否则返回false

count 统计Observable总共发送了多少个数据，在Observable没有停止发送前，count是不会发送统计数据

reduce，与Lambda中的reduce一样

Observable.*just*(1, 2, 3).map -> Func1 -> Action1 映射，是1对1的转换

Observable.*just*(1, 2, 3).flatMap -> Func1 -> Action1 映射，1对多的转换

Observable.*just*(1, 2, 3).flatMapIterable -> Func1 -> Action1 把多个集合合并成一个

CharSequence charSequence = **new** String(**"Haha"**);  
Observable.*just*(charSequence).cast(String.**class**) -> Action1 类型转换

Observable.*from*(list).scan -> Func2 -> Action1 类型Lambda中的reduce

Single -> Single.create -> Single.OnSubscribe -> SingleSubscriber 有一个接收数据的方法和一个onError方法

Completable -> Completable.error/Completable.complete -> CompletableSubscriber 没有接收数据的方法，只能知道成功或失败