|  |  |  |
| --- | --- | --- |
|  | 1.0 | 2.0 |
| 被观察者/观察者 | Observable/Observer(eg1.1);  /Subcriber | Observable/Observer(eg:2.1)  Flowable/Subscriber(eg2.2)  （支持背压）  Single/SingleObserver(eg2.3)  Completable/CompletableObserver(eg2.4)  Maybe/MaybeObserver(eg:2.5) |
| Action | Action0---Action9;  Action1 (eg1.2) | Action1🡪Consumer(2.2)  Action2--------BiConsumer()  Action3-9(舍弃) |
| Function | Function1—Function9  (eg:1.3) | Function1🡪Function(2.6)  Function2🡪BiFunction  Function3--9🡪不变 |

Eg1.1:

@Test  
public void testObserver1(){  
   
 Observable<Integer> observerable = Observable.*create*(new Observable.OnSubscribe<Integer>() {  
 @Override  
 public void call(Subscriber<? super Integer> subscriber) {  
 subscriber.onNext(0);  
 subscriber.onNext(1);  
 subscriber.onNext(2);  
 subscriber.onCompleted();  
 }  
 });  
   
 Observer<Integer> observer = new Observer<Integer>() {  
 @Override  
 public void onCompleted() {  
  
 System.*out*.println("complete");  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
 }  
  
 @Override  
 public void onNext(Integer integer) {  
 System.*out*.println(integer);  
  
 }  
 };  
   
 observerable.subscribe(observer);  
   
  
}

或：

Subscriber<Integer> subscriber = new Subscriber<Integer>() {  
 @Override  
 public void onCompleted() {  
 System.*out*.println("complete");  
  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
 }  
  
 @Override  
 public void onNext(Integer integer) {  
 System.*out*.println(integer);  
  
 }  
};

Eg1.2:

@Test  
public void testRange1() {  
  
 Observable.*range*(0, 3)  
 .subscribe(new Action1<Integer>() {  
 @Override  
 public void call(Integer integer) {  
 System.*out*.println(integer);  
 }  
 });  
}

eg:1.3

@Test  
public void testMap() {  
 Observable.*just*("1", "2")  
 .map(new Func1<String, Integer>() {  
  
 @Override  
 public Integer call(String s) {  
 return Integer.*parseInt*(s);  
 }  
 }).subscribe(new Action1<Integer>() {  
 @Override  
 public void call(Integer integer) {  
 System.*out*.println(integer);  
 }  
 });  
}

Eg2.1:

@Test  
public void testObserver2(){  
  
 Observable<Integer> observerable = Observable.*create*(new ObservableOnSubscribe<Integer>() {  
 @Override  
 public void subscribe(ObservableEmitter<Integer> e) throws Exception {  
 e.onNext(0);  
 e.onNext(1);  
 e.onNext(2);  
 e.onComplete();  
 }  
 });  
  
 Observer<Integer> observer = new Observer<Integer>() {  
 @Override  
 public void onSubscribe(Disposable d) {  
  
 System.*out*.println("disposable");  
 }  
  
 @Override  
 public void onNext(Integer integer) {  
  
  
 System.*out*.println(integer);  
  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
 }  
  
 @Override  
 public void onComplete() {  
  
 System.*out*.println("oncomplete");  
 }  
 };  
  
 observerable.subscribe(observer);  
  
}

或：

Subscriber<Integer> subscriber = new Subscriber<Integer>() {  
 @Override  
 public void onSubscribe(Subscription s) {  
 System.*out*.println("disposable");  
 }  
  
 @Override  
 public void onNext(Integer integer) {  
 System.*out*.println(integer);  
 }  
  
 @Override  
 public void onError(Throwable t) {  
  
 }  
  
 @Override  
 public void onComplete() {  
 System.*out*.println("oncomplete");  
 }  
};

eg2.2:

@Test  
public void testFlowable2(){  
  
 Flowable.*range*(0,1000)  
 .subscribe(new Subscriber<Integer>() {  
 Subscription subscription;  
 @Override  
 public void onSubscribe(Subscription s) {  
 subscription=s;  
 System.*out*.println("ss");

//从上游主动拉取数据  
 subscription.request(1);  
 }  
  
 @Override  
 public void onNext(Integer integer) {  
  
 System.*out*.println(integer);

//处理完任务后再接着拉取  
 subscription.request(1);  
 }  
  
 @Override  
 public void onError(Throwable t) {  
  
 }  
  
 @Override  
 public void onComplete() {  
  
 System.*out*.println("oncom");  
 subscription.cancel();  
 }  
 });  
  
}

eg2.3

@Test  
public void testSingle2(){  
 Single single = Single.*create*(new SingleOnSubscribe() {  
 @Override  
 public void subscribe(SingleEmitter e) throws Exception {  
 e.onSuccess(0);  
 }  
 });  
  
 single.subscribe(new SingleObserver<Integer>() {  
 @Override  
 public void onSubscribe(Disposable d) {  
 System.*out*.println("d");  
 }  
  
 @Override  
 public void onSuccess(Integer o) {  
 System.*out*.println(o);  
  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
 }  
 });  
}

2.4

@Test  
public void testCompletable(){  
  
 Completable.*create*(new CompletableOnSubscribe() {  
 @Override  
 public void subscribe(CompletableEmitter e) throws Exception {  
 e.onComplete();  
 }  
 }).subscribe(new CompletableObserver() {  
 @Override  
 public void onSubscribe(Disposable d) {  
   
 }  
  
 @Override  
 public void onComplete() {  
  
 System.*out*.println("complete");  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
 }  
 });  
  
}

eg2.5:

@Test  
public void testMaybe(){  
  
 Maybe.*create*(new MaybeOnSubscribe<Integer>() {  
 @Override  
 public void subscribe(MaybeEmitter<Integer> e) throws Exception {  
 e.onSuccess(1);  
 e.onComplete();  
 }  
 }).subscribe(new MaybeObserver<Integer>() {  
 @Override  
 public void onSubscribe(Disposable d) {  
 System.*out*.println("d");  
 }  
  
 @Override  
 public void onSuccess(Integer integer) {  
  
 System.*out*.println(integer);  
 }  
  
 @Override  
 public void onError(Throwable e) {  
  
  
 }  
  
 @Override  
 public void onComplete() {  
  
 System.*out*.println("com");  
 }  
 });  
}

eg:2.6

@Test  
public void testFunction2(){  
  
 Flowable.*range*(1,3)  
 .map(new Function<Integer, Integer>() {  
 @Override  
 public Integer apply(Integer integer) throws Exception {  
 return integer+10;  
 }  
 }).subscribe(new Consumer<Integer>() {  
 @Override  
 public void accept(Integer integer) throws Exception {  
 System.*out*.println(integer);  
  
 }  
 });  
}