RxJava on Android

扔物线 

RxJava on Android

RxJava on Android

- RxJava 是什么

RxJava on Android

- RxJava 是什么
- RxJava 的优势

RxJava on Android

- RxJava 是什么
- RxJava 的优势
- API 介绍

RxJava on Android

- RxJava 是什么
- RxJava 的优势
- API 介绍
- 适用场景

RxJava on Android

- RxJava 是什么：异步
- RxJava 的优势
- API 介绍
- 适用场景

RxJava on Android

- RxJava 是什么：异步
- RxJava 的优势：简洁
- API 介绍
- 适用场景

API 介绍

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

概念：扩展的观察者模式

概念：扩展的观察者模式

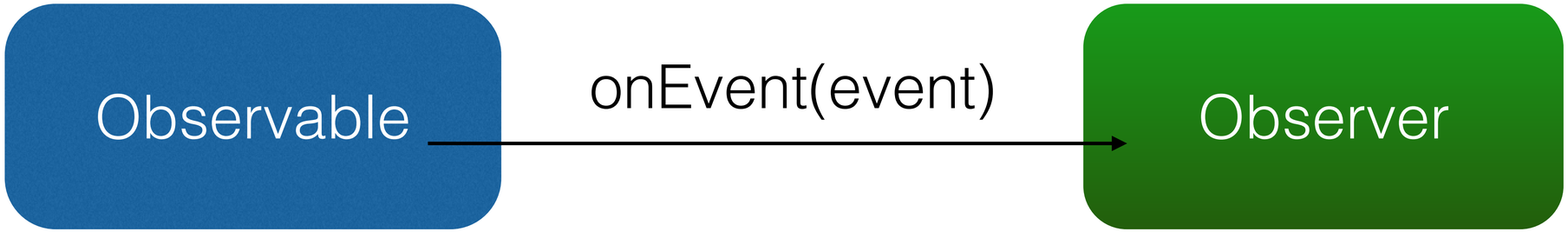
- 观察者模式

概念：扩展的观察者模式

- 观察者模式

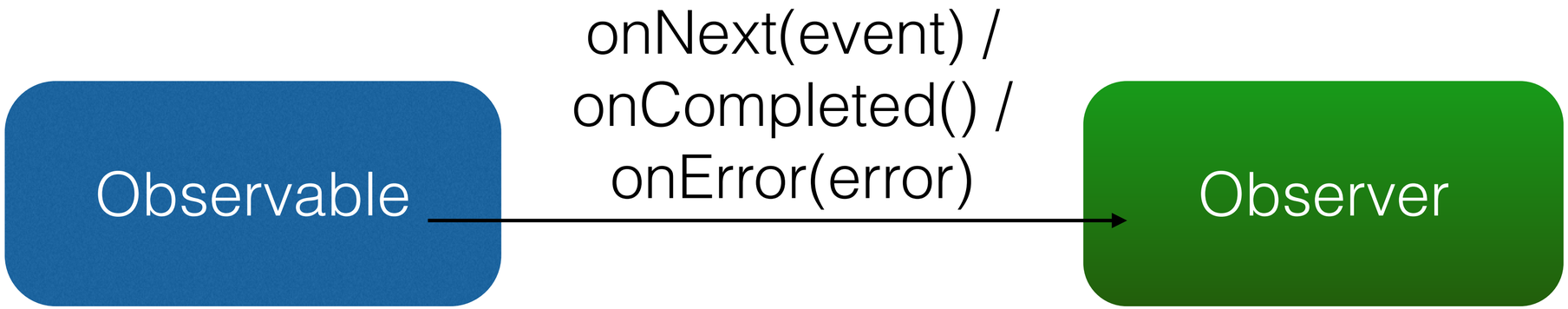
概念：扩展的观察者模式

- 观察者模式



概念：扩展的观察者模式

- 观察者模式
- RxJava 的观察者模式



API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

基本实现

基本实现

- 创建 Observer

基本实现

- 创建 Observer
- 创建 Observable

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

创建 Observer

创建 Observer

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Observer<String> observer = new Observer<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Subscriber<String> observer = new Subscriber<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

创建 Observer

```
Subscriber<String> observer = new Subscriber<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

onStart()

创建 Observer

```
Subscriber<String> observer = new Subscriber<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

onStart()

unsubscribe()

创建 Observer

```
Subscriber<String> observer = new Subscriber<String>() {
    @Override
    public void onNext(String s) {
        Log.d(TAG, "Item: " + s);
    }

    @Override
    public void onCompleted() {
        Log.d(TAG, "Completed!");
    }

    @Override
    public void onError(Throwable e) {
        Log.d(TAG, "Error!");
    }
};
```

onStart()

unsubscribe()

isUnsubscribed()

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

创建 Observable

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});
```

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});
```

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});
```

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});
```

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});

Observable observable = Observable.just("Hello", "Hi", “Aloha");
```

创建 Observable

```
Observable observable = Observable.create(new OnSubscribe<String>()
{
    @Override
    public void call(Subscriber<? super String> subscriber) {
        subscriber.onNext("Hello");
        subscriber.onNext("Hi");
        subscriber.onNext("Aloha");
        subscriber.onCompleted();
    }
});

Observable observable = Observable.just("Hello", "Hi", “Aloha");

String[] words = {"Hello", "Hi", "Aloha"};
Observable observable = Observable.from(words);
```

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

subscribe (订阅)

subscribe (订阅)

```
observable.subscribe(observer);
```

subscribe (订阅)

```
observable.subscribe(observer);

observable.subscribe(subscriber);
```

subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

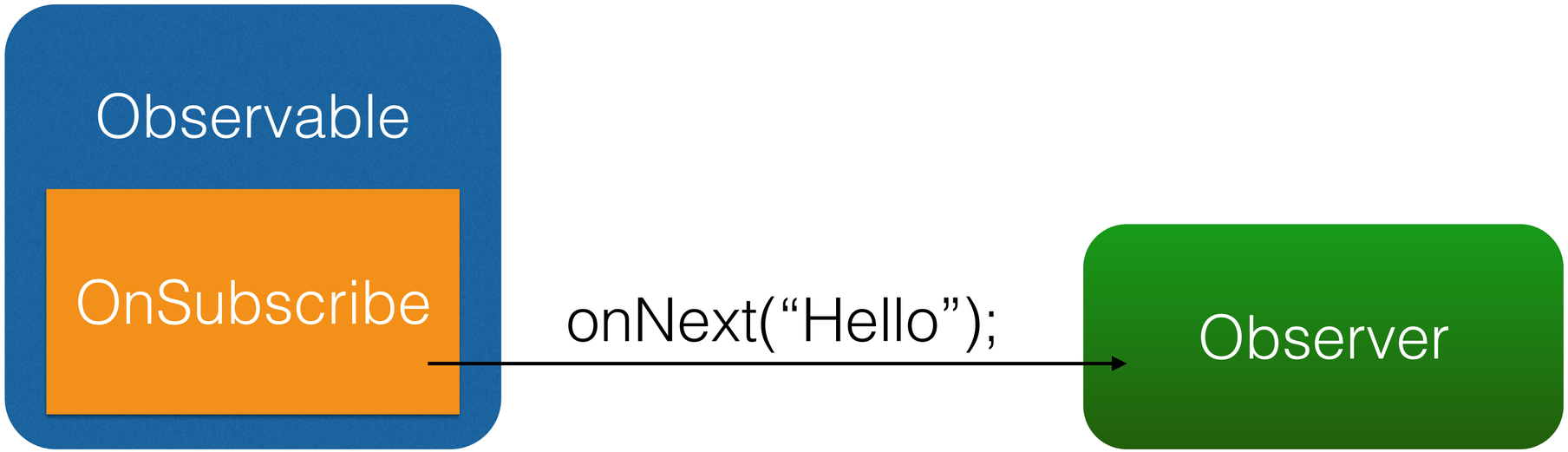
subscribe (订阅)

```
public Subscription subscribe(Subscriber subscriber) {
    subscriber.onStart();
    onSubscribe.call(subscriber);
    return subscriber;
}
```

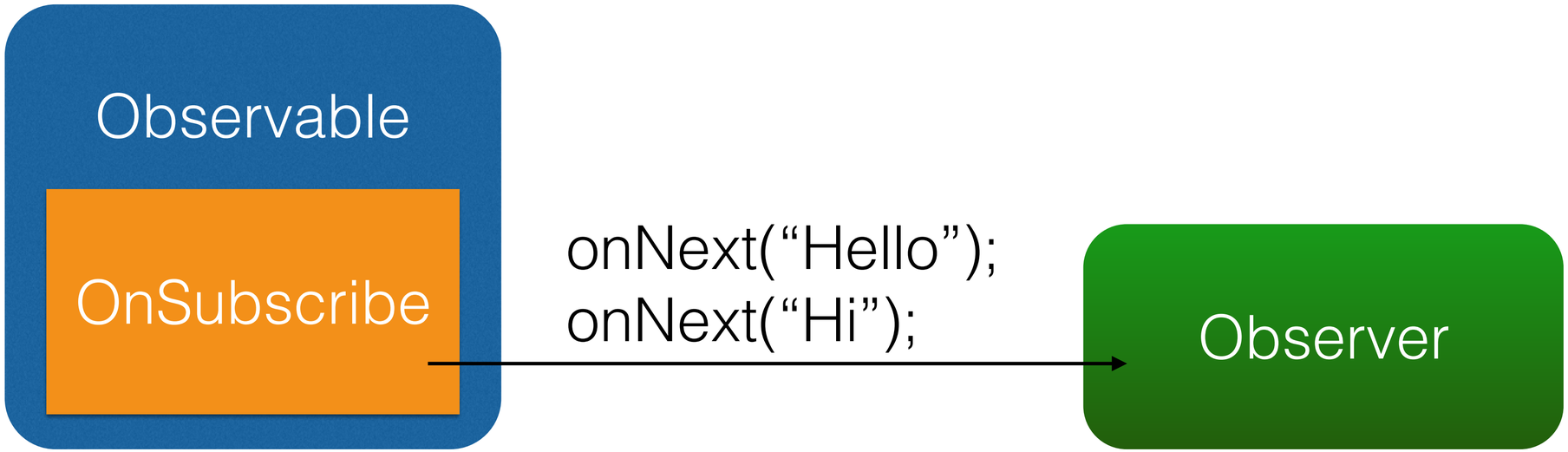
subscribe (订阅)

subscribe (订阅)

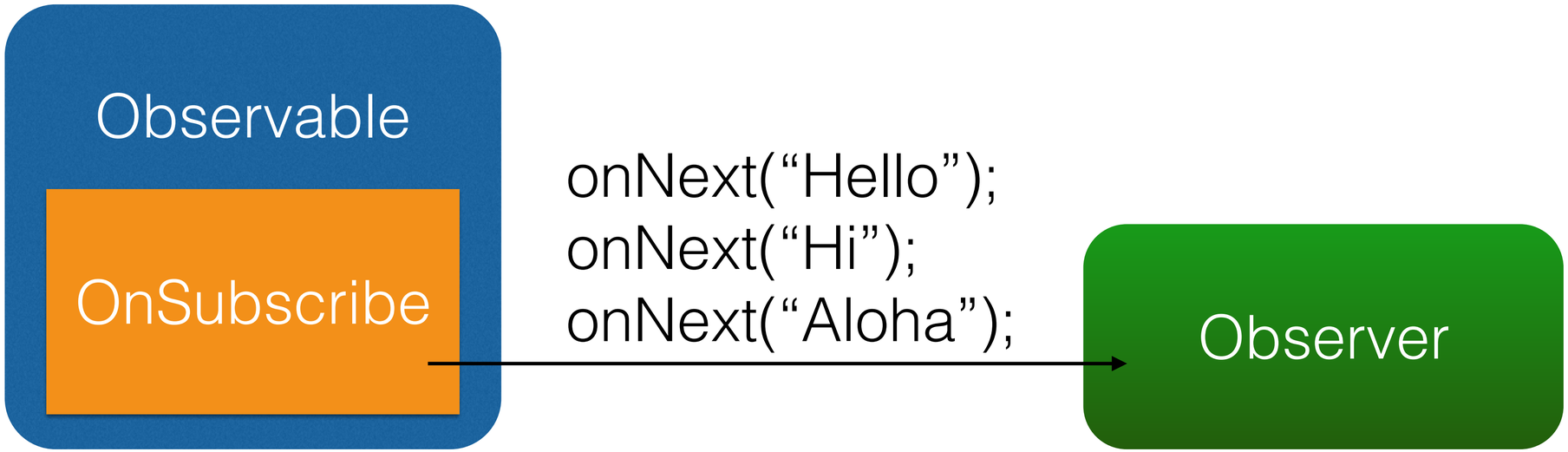
subscribe (订阅)



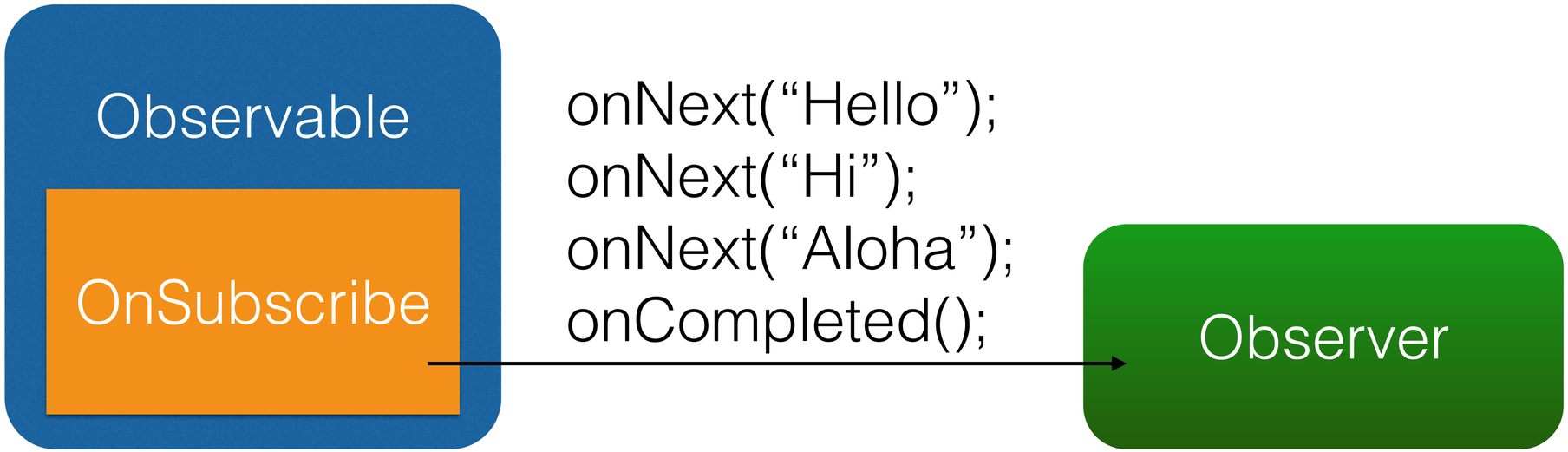
subscribe (订阅)



subscribe (订阅)



subscribe (订阅)



基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

基本实现

- 创建 Observer
- 创建 Observable
- subscribe (订阅)

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

线程控制：Schedulers

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)


    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

```
Observable.just(1, 2, 3, 4)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Integer>() {
        @Override
        public void call(Integer number) {
            Log.d(tag, "number:" + number);
        }
    });
```

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

线程控制：Schedulers

- Schedulers.io()
- AndroidSchedulers.mainThread()
- Schedulers.newThread()
- Schedulers.calculation()

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

変換

変換

```
Observable.just(“images/image1.png”, “images.image2.png)
    .subscribe(new Action1<String>() {
        @Override
        public void call(String path) {
            Bitmap bitmap = getBitmapFromPath(filePath);
            addBitmapToView(bitmap);
        }
    });
```

変換

```
Observable.just(“images/image1.png”, “images.image2.png)
```

```
    .subscribe(new Action1<String>() {
        @Override
        public void call(String path) {
            Bitmap bitmap = getBitmapFromPath(filePath);
            addBitmapToView(bitmap);
        }
    });
```

変換

```
Observable.just("images/image1.png", "images.image2.png")
    .map(new Func1<String, Bitmap>() {
        @Override
        public Bitmap call(String path) {

        }
    .subscribe(new Action1<String>() {
        @Override
        public void call(String path) {
            Bitmap bitmap = getBitmapFromPath(filePath);
            addBitmapToView(bitmap);
        }
    });
```

変換

```
Observable.just(“images/image1.png”, “images.image2.png)
    .map(new Func1<String, Bitmap>() {
        @Override
        public Bitmap call(String path) {
            return getBitmapFromPath(filePath);
        }
    .subscribe(new Action1<Bitmap>() {
        @Override
        public void call(Bitmap path) {
            addBitmapToView(bitmap);
        }
    });
```

変換

```
Observable.just(“images/image1.png”, “images.image2.png)
    .map(new Func1<String, Bitmap>() {
        @Override
        public Bitmap call(String path) {
            return getBitmapFromPath(filePath);
        }


    .subscribe(new Action1<Bitmap>() {
        @Override
        public void call(Bitmap path) {
            addBitmapToView(bitmap);
        }
    });
```

変換

```
Observable.just(“images/image1.png”, “images.image2.png)
    .map(new Func1<String, Bitmap>() {
        @Override
        public Bitmap call(String path) {
            return getBitmapFromPath(filePath);
        }
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(new Action1<Bitmap>() {
        @Override
        public void call(Bitmap path) {
            addBitmapToView(bitmap);
        }
    });
```

変換

- map()

変換

- map()
- flatMap()

flatMap()

flatMap()

```
twitterApi.getTweets(bearerToken, guestToken, country, callback)
```

flatMap()

```
twitterApi.getTweets(bearerToken, guestToken, country)
```

flatMap()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

flatMap()

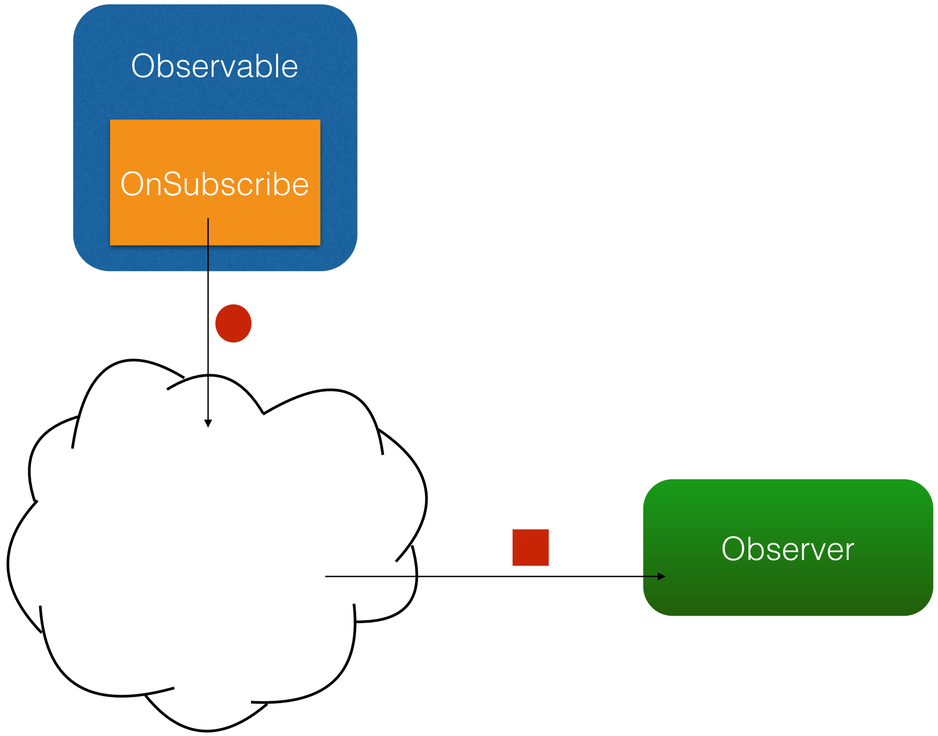
```
twitterApi.getGuestToken(bearerToken, body)

twitterApi.getTweets(bearerToken, guestToken, country)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

flatMap()

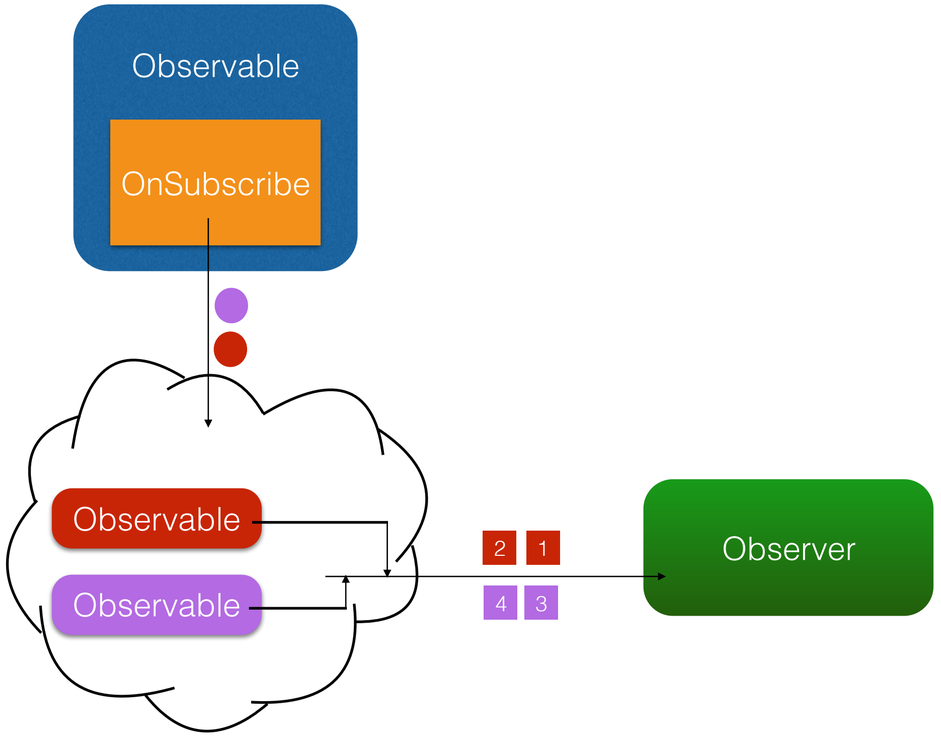
```
twitterApi.getGuestToken(bearerToken, body)
    .flatMap(new Func1<TwitterGuestToken, Observable<Tweets>>() {
        @Override
        public Observable<Tweets> call(GuestToken guestToken) {
            return twitterApi.getTweets(bearerToken, guestToken,
country);
        }
    })
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

flatMap()

map()



flatMap()



変換

- map()
- flatMap()

変換

- map()
- flatMap()
- doOnNext()

doOnNext()

doOnNext()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnNext()

```
twitterApi.getTweets(bearerToken, guestToken, country)

        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(observer);
```

doOnNext()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .doOnNext(new Action1<Tweets>() {
        @Override
        public void call(Tweets tweets) {
            saveTweetsToDb();
        }
    })
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

変換

- map()
- flatMap()
- doOnNext()

変換

- map()
- flatMap()
- doOnNext()
- doOnSubscribe()

doOnSubscribe()

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)

    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())

    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint("Flipboard 北京正在招 Android 工程师！！！");
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint("Flipboard 北京正在招 Android 工程师！！！");
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint(“Flipboard 北京正在招 Android 工程师！！！”);
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint("Flipboard 北京正在招 Android 工程师！！！");
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint("Flipboard 北京正在招 Android 工程师！！！");
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

doOnSubscribe()

```
twitterApi.getTweets(bearerToken, guestToken, country)
    .subscribeOn(Schedulers.io())
    .doOnSubscribe(new Action0() {
        @Override
        public void call() {
            showHint("Flipboard 北京正在招 Android 工程师！！！");
        }
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(observer);
```

変換

- map()
- flatMap()
- doOnNext()
- doOnSubscribe()

変換

- map()
- flatMap()
- doOnNext()
- doOnSubscribe()
- …

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

API 介绍

- 概念：扩展的观察者模式
- 基本实现
- 线程控制：Schedulers
- 变换

RxJava on Android

- RxJava 是什么：异步
- RxJava 的优势：简洁
- API 介绍
- 适用场景

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

适用场景

- 与 Retrofit 的结合
- RxBinding
- 各种异步操作
- RxBus

RxJava on Android

- RxJava 是什么：异步
- RxJava 的优势：简洁
- API 介绍
- 适用场景

问题?