Rx 在 Zhihu 的历史

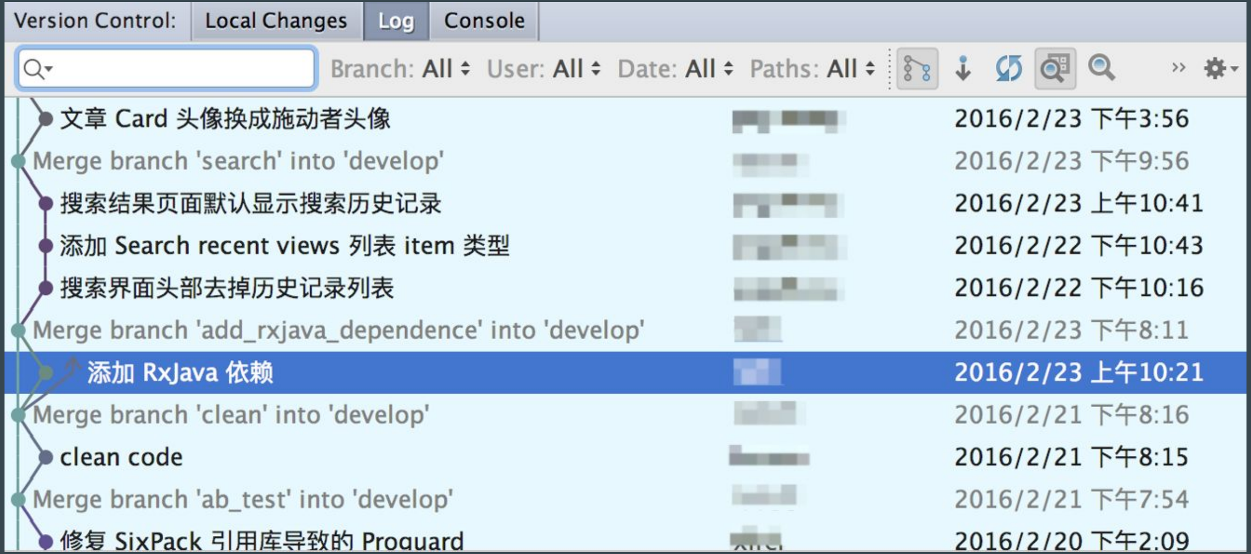
...

@杨凡-知乎

知乎

Rx 版本号的变迁

引入 Rx 1.1.0



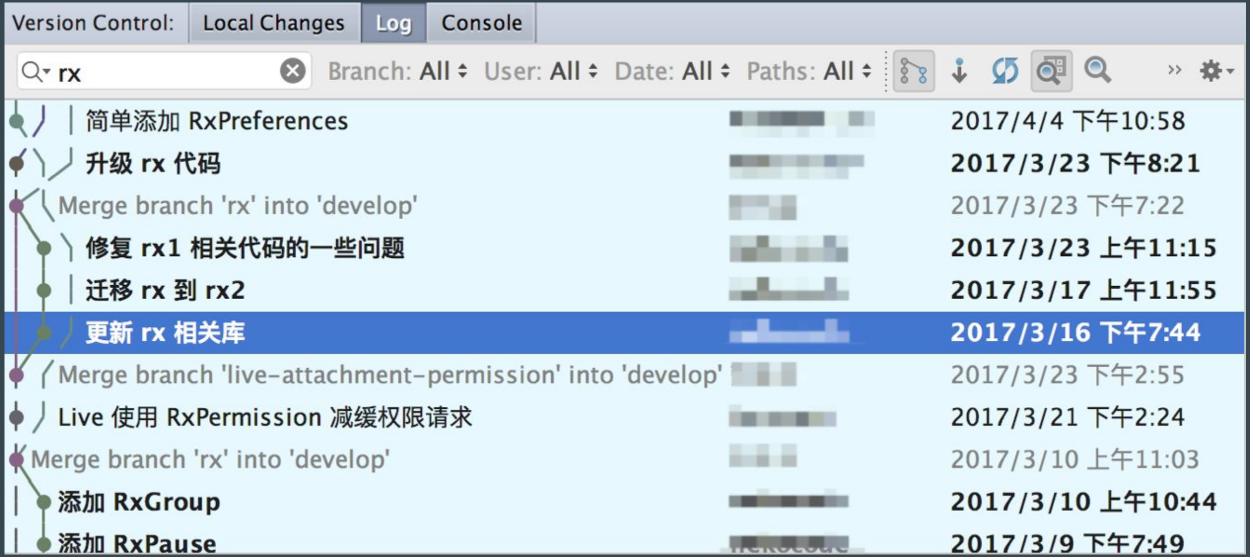
Version Control: Local Changes Log Console

Branch: All User: All Date: All Paths: All

```
文章 Card 头像换成施动者头像                          2016/2/23 下午3:56
Merge branch 'search' into 'develop'                2016/2/23 下午9:56
搜索结果页面默认显示搜索历史记录                      2016/2/23 上午10:41
添加 Search recent views 列表 item 类型              2016/2/22 下午10:43
搜索界面头部去掉历史记录列表                          2016/2/22 下午10:16
Merge branch 'add_rxjava_dependence' into 'develop' 2016/2/23 下午8:11
添加 RxJava 依赖                                     2016/2/23 上午10:21
Merge branch 'clean' into 'develop'                 2016/2/21 下午8:16
clean code                                          2016/2/21 下午8:15
Merge branch 'ab_test' into 'develop'               2016/2/21 下午7:54
修复 SixPack 引用库导致的 Proguard                    2016/2/20 下午2:09
```

知乎

升级到 Rx 2.0.7

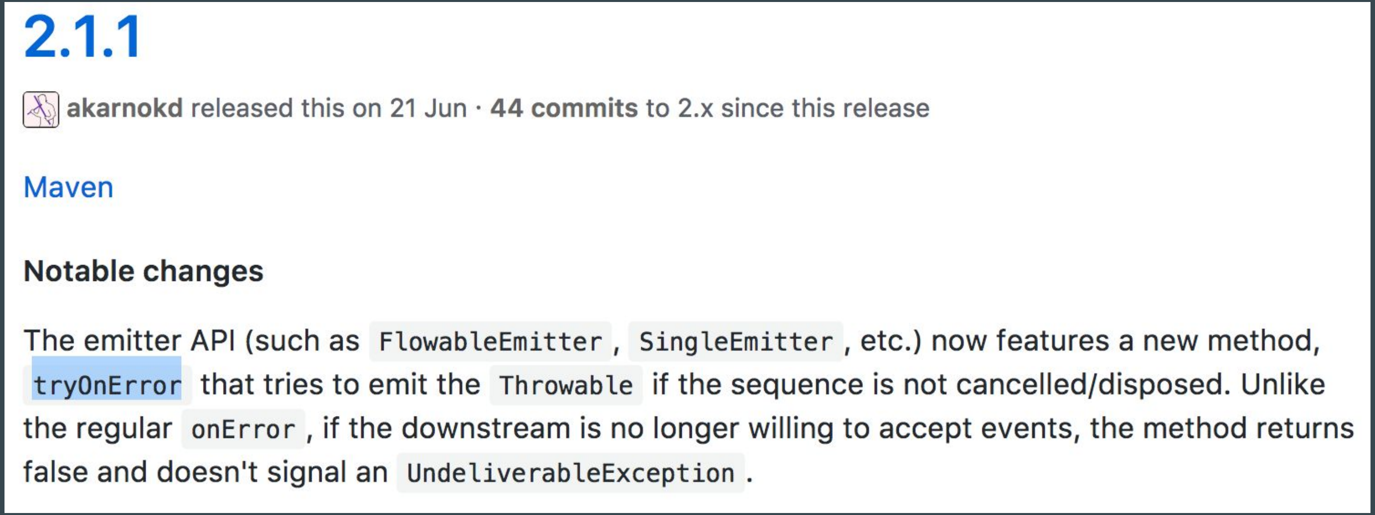


Version Control: Local Changes Log Console

rx   Branch: All   User: All   Date: All   Paths: All

简单添加 RxPreferences	[redacted]	2017/4/4 下午10:58
升级 rx 代码	[redacted]	**2017/3/23 下午8:21**
Merge branch 'rx' into 'develop'	[redacted]	2017/3/23 下午7:22
修复 rx1 相关代码的一些问题	[redacted]	**2017/3/23 上午11:15**
迁移 rx 到 rx2	[redacted]	**2017/3/17 上午11:55**
更新 rx 相关库	[redacted]	**2017/3/16 下午7:44**
Merge branch 'live-attachment-permission' into 'develop'	[redacted]	2017/3/23 下午2:55
Live 使用 RxPermission 减缓权限请求	[redacted]	2017/3/21 下午2:24
Merge branch 'rx' into 'develop'	[redacted]	2017/3/10 上午11:03
添加 RxGroup	[redacted]	**2017/3/10 上午10:44**
添加 RxPause	[redacted]	**2017/3/9 下午7:49**

目前使用的版本是 2.1.1



2.1.1

akarnokd released this on 21 Jun · 44 commits to 2.x since this release

Maven

Notable changes

The emitter API (such as `FlowableEmitter`, `SingleEmitter`, etc.) now features a new method, `tryOnError` that tries to emit the `Throwable` if the sequence is not cancelled/disposed. Unlike the regular `onError`, if the downstream is no longer willing to accept events, the method returns false and doesn't signal an `UndeliverableException`.

知乎

为什么会选择 Rx

一些现有方案的对比

	使用 Thread	使用 AsyncTask	使用 RxJava
封装异步任务	繁琐	繁琐	***简单***
更新 UI	通过 Handler	在回调函数里	***通过 Scheduler***
线程池	手动实现	提供接口	***Scheduler 内部封装***
线程间同步	繁琐	繁琐	***简单***

知乎

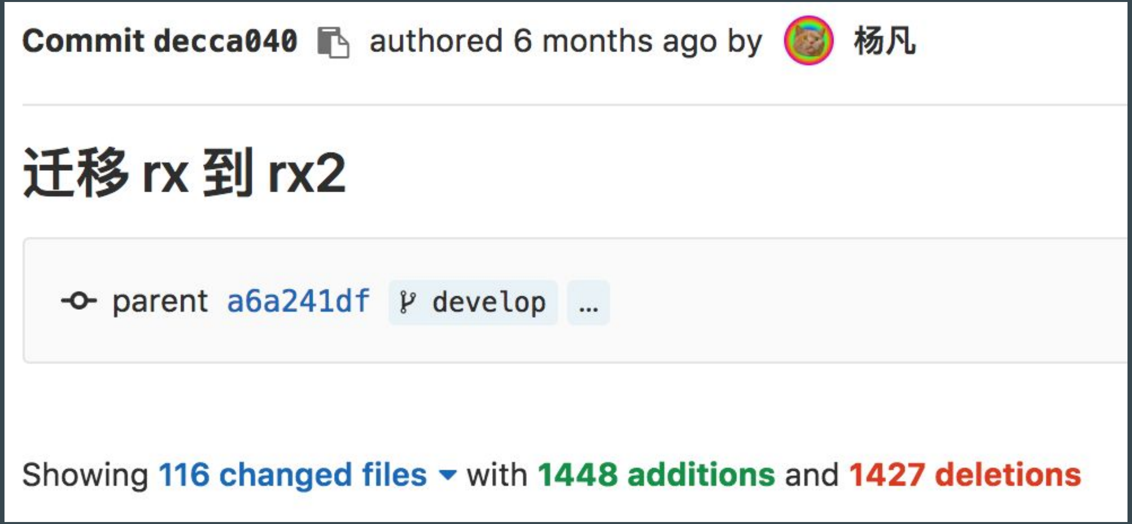
Rx1 迁移到 Rx2

为什么迁移到 Rx2

- RxJava 2.0 has been completely rewritten from scratch on top of the Reactive-Streams specification. The specification itself has evolved out of RxJava 1.x and provides a common baseline for reactive systems and libraries.

- Rx2 基于最新 Reactive-Streams 规范完全重写

共计 1,400 余行代码改动

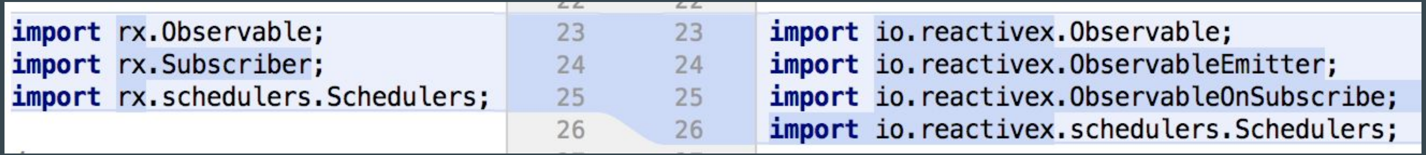

Commit decca040 authored 6 months ago by 杨凡

迁移 rx 到 rx2

parent a6a241df develop ...

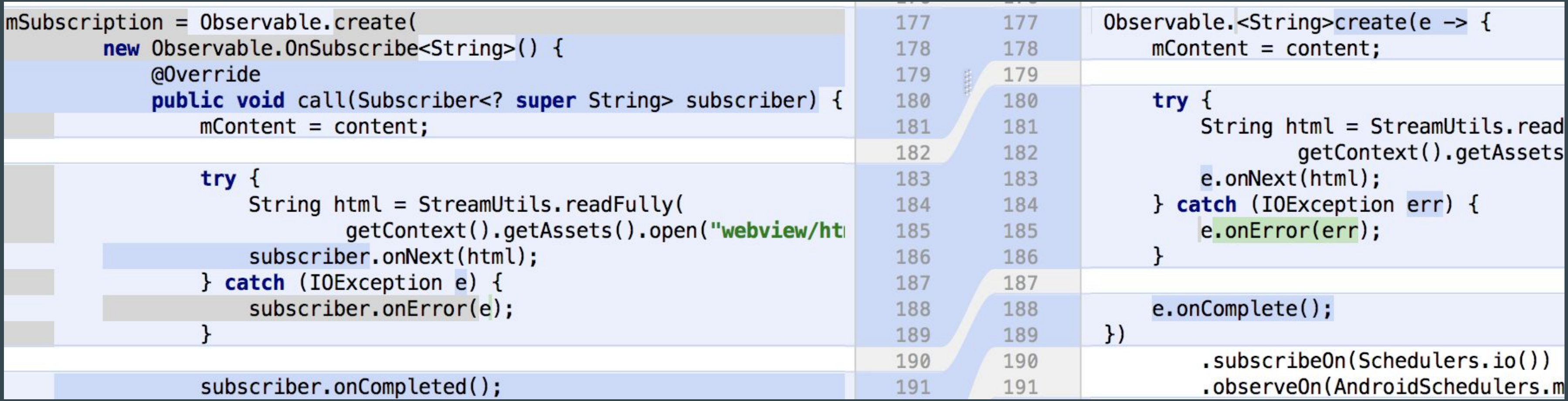
Showing 116 changed files with 1448 additions and 1427 deletions

Package Path、Class/Method Name 改动



```
import rx.Observable;
import rx.Subscriber;
import rx.schedulers.Schedulers;
```

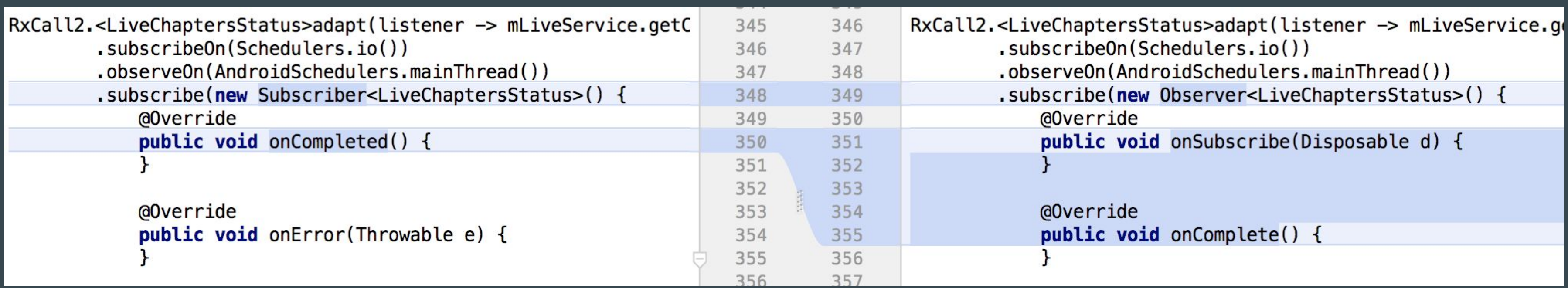
```
import io.reactivex.Observable;
import io.reactivex.ObservableEmitter;
import io.reactivex.ObservableOnSubscribe;
import io.reactivex.schedulers.Schedulers;
```



```
mSubscription = Observable.create(
        new Observable.OnSubscribe<String>() {
            @Override
            public void call(Subscriber<? super String> subscriber) {
                mContent = content;

                try {
                    String html = StreamUtils.readFully(
                            getContext().getAssets().open("webview/ht
                    subscriber.onNext(html);
                } catch (IOException e) {
                    subscriber.onError(e);
                }

                subscriber.onCompleted();
```

```
Observable.<String>create(e -> {
    mContent = content;

    try {
        String html = StreamUtils.read
                getContext().getAssets
        e.onNext(html);
    } catch (IOException err) {
        e.onError(err);
    }

    e.onComplete();
})
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.m
```

Observer 增加了 onSubscribe() 方法

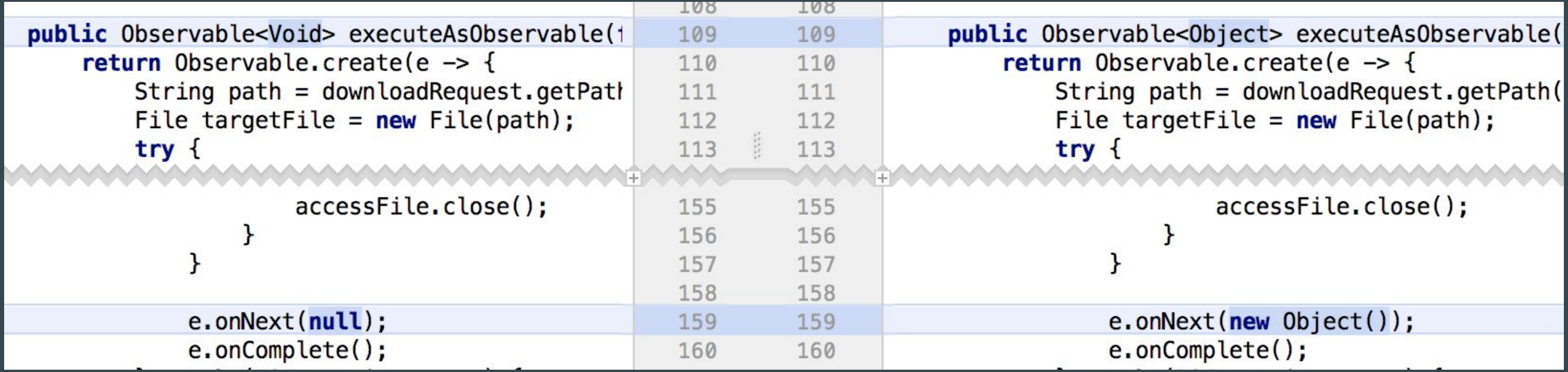


```
RxCall2.<LiveChaptersStatus>adapt(listener -> mLiveService.getC
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<LiveChaptersStatus>() {
            @Override
            public void onCompleted() {
            }

            @Override
            public void onError(Throwable e) {
            }
```

```
RxCall2.<LiveChaptersStatus>adapt(listener -> mLiveService.ge
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Observer<LiveChaptersStatus>() {
            @Override
            public void onSubscribe(Disposable d) {
            }

            @Override
            public void onComplete() {
            }
```

知乎

不允许发射 Null



```
public Observable<Void> executeAsObservable(
    return Observable.create(e -> {
        String path = downloadRequest.getPat
        File targetFile = new File(path);
        try {

                accessFile.close();
            }
        }

        e.onNext(null);
        e.onComplete();
```

```
public Observable<Object> executeAsObservable(
    return Observable.create(e -> {
        String path = downloadRequest.getPath(
        File targetFile = new File(path);
        try {

                accessFile.close();
            }
        }

        e.onNext(new Object());
        e.onComplete();
```

知乎

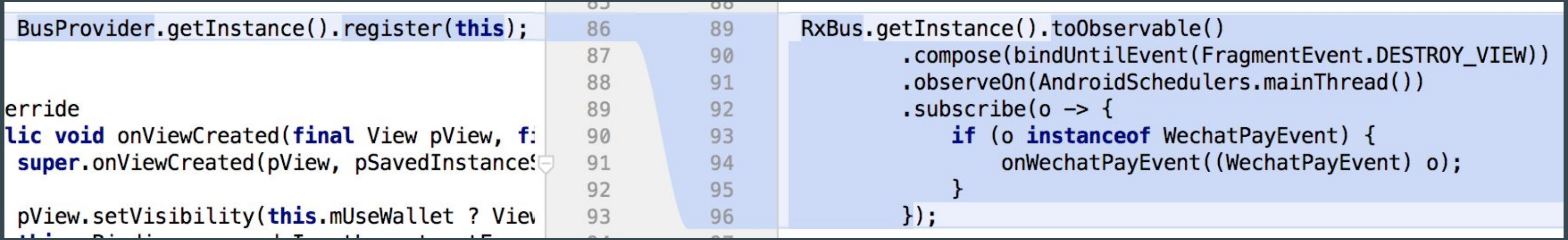
Otto Bus 迁移到 Rx Bus

为什么要迁移到 Rx Bus

- 两个方案一直并存，长远来说需要进行统一
- 发散 Rx 的应用
- Rx Bus 可以组合 RxLifecycle 等操作符使用

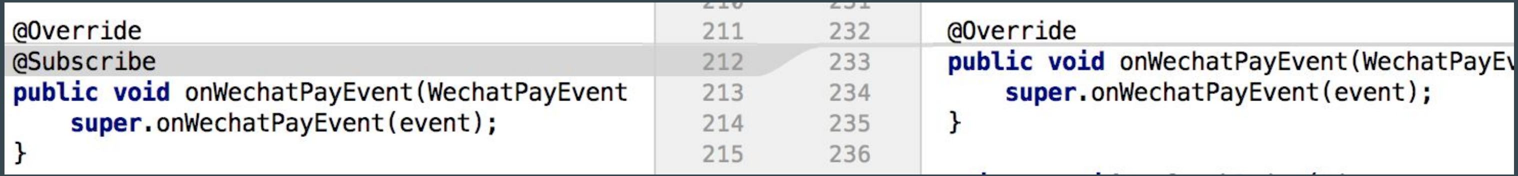
知乎

迁移后的代码



```
 BusProvider.getInstance().register(this);


erride
lic void onViewCreated(final View pView, f:
 super.onViewCreated(pView, pSavedInstance

 pView.setVisibility(this.mUseWallet ? Vie
```

```
RxBus.getInstance().toObservable()
        .compose(bindUntilEvent(FragmentEvent.DESTROY_VIEW))
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(o -> {
            if (o instanceof WechatPayEvent) {
                onWechatPayEvent((WechatPayEvent) o);
            }
        });
```

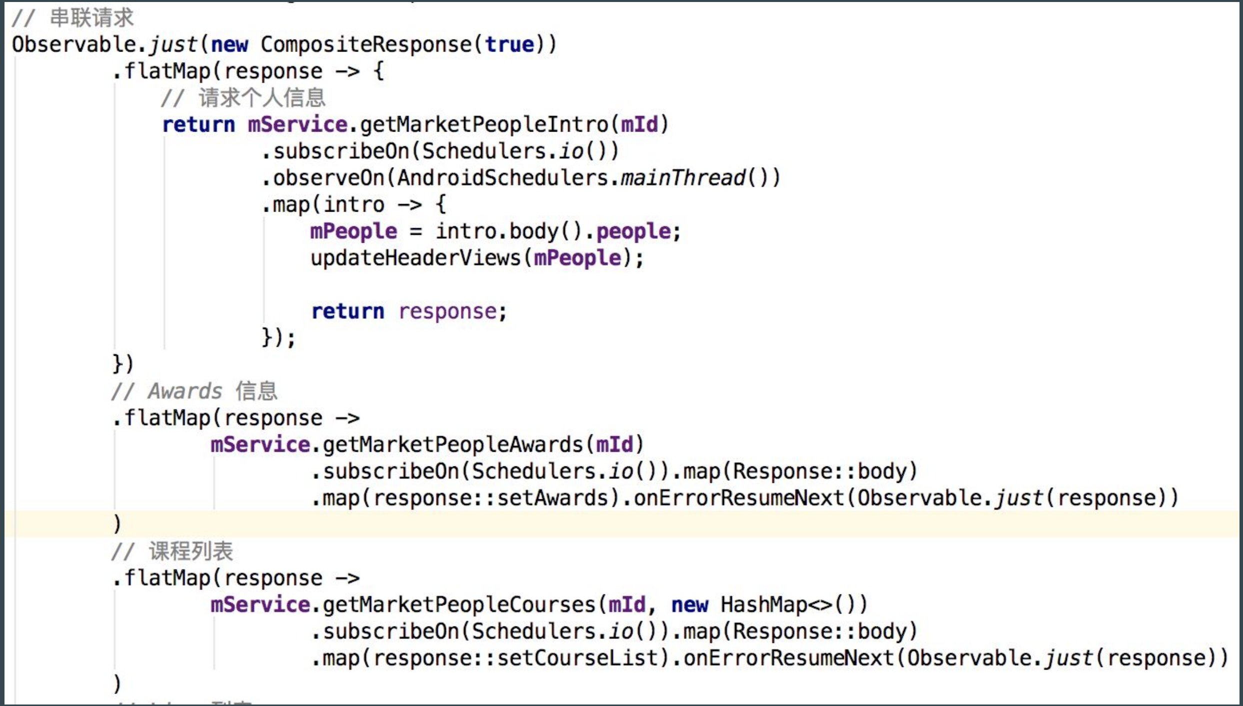


```
@Override
@Subscribe
public void onWechatPayEvent(WechatPayEvent
    super.onWechatPayEvent(event);
}
```

```
@Override
public void onWechatPayEvent(WechatPayEv
    super.onWechatPayEvent(event);
}
```

网络库迁移到 Retrofit

使用 Retrofit 后



```
// 串联请求
Observable.just(new CompositeResponse(true))
        .flatMap(response -> {
            // 请求个人信息
            return mService.getMarketPeopleIntro(mId)
                    .subscribeOn(Schedulers.io())
                    .observeOn(AndroidSchedulers.mainThread())
                    .map(intro -> {
                        mPeople = intro.body().people;
                        updateHeaderViews(mPeople);

                        return response;
                    });
        })
        // Awards 信息
        .flatMap(response ->
                mService.getMarketPeopleAwards(mId)
                        .subscribeOn(Schedulers.io()).map(Response::body)
                        .map(response::setAwards).onErrorResumeNext(Observable.just(response))
        )
        // 课程列表
        .flatMap(response ->
                mService.getMarketPeopleCourses(mId, new HashMap<>())
                        .subscribeOn(Schedulers.io()).map(Response::body)
                        .map(response::setCourseList).onErrorResumeNext(Observable.just(response))
        )
```

知乎

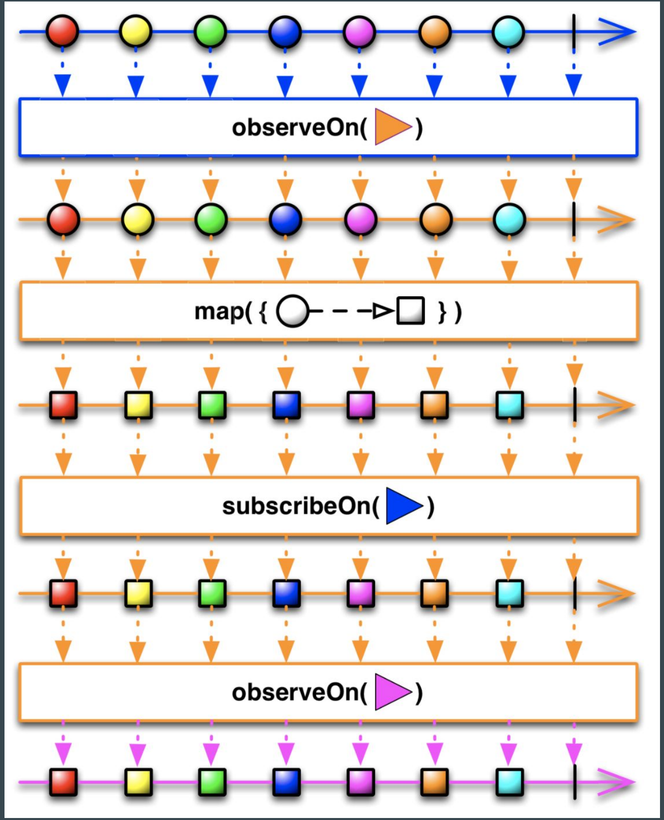
使用中常见的几个问题

Scheduler 相关问题 (1)

```
Observable.create(e -> {
    A();
})
    .observeOn(Schedulers.io())
    .flatMap(s -> {
        B();
        return Observable.create(e -> {
            C();
        })
        .subscribeOn(
            AndroidSchedulers.mainThread());
    })
    .map(s -> {
        D();
    })
    .subscribeOn(Schedulers.computation())
```

知乎

Scheduler 相关问题 (2)



首先，可以把每个 observeOn() 看作一块挡板

observeOn() 调度的是：
当前这块挡板，到一下挡板之间的操作

subscribeOn() 调度的是：
最上游，到第一块挡板之间的操作
(有多个 subscribeOn() 的话，取最上游的哪个)

知乎

Scheduler 相关问题 (3)

```
Observable.create(e -> {
    A();
})
    .observeOn(Schedulers.io())
    .flatMap(s -> {
        B();
        return Observable.create(e -> {
            C();
        })
        .subscribeOn(
            AndroidSchedulers.mainThread());
    })
    .map(s -> {
        D();
    })
    .subscribeOn(Schedulers.computation())
```

- 类似的操作符包括 defer()、onErrorResumeNext() ...

Scheduler 相关问题 (4)

```
Observable.create(e -> {
    A();
})
    .observeOn(Schedulers.io())
    .flatMap(s -> {
        B();
        return Observable.create(e -> {
            C();
        });
    })
    .subscribeOn(AndroidSchedulers.mainThread())
    .map(s -> {
        D();
    })
    .subscribeOn(Schedulers.computation())
```

知乎

Scheduler 相关问题 (5)

- 尽量多用 observeOn()
- 保证整个流里只有一个 subscribeOn()，放在越前面越好

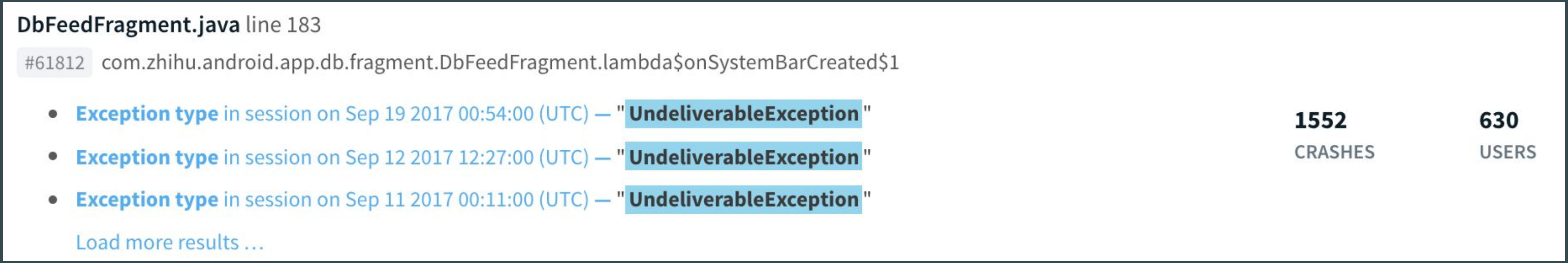
Scheduler 相关问题 (5)

```
Observable.create(e -> {
    A();
})
    .observeOn(Schedulers.io())
    .flatMap(s -> {
        B();
        return Observable.create(e -> {
            C();
        })
        .subscribeOn(
           AndroidSchedulers.mainThread());
    })
    .map(s -> {
        D();
    })
    .subscribeOn(Schedulers.computation())
```

```
Observable.create(e -> {
    A();
})
    .subscribeOn(Schedulers.computation())
    .observeOn(Schedulers.io())
    .map(s -> { B(); })
    .observeOn(
        AndroidSchedulers.mainThread())
    .flatMap(s -> {
        return Observable.create(e -> {
            C();
        })
    })
    .map(s -> {
        D();
    })
```

知乎

Undeliverable Exception (1)



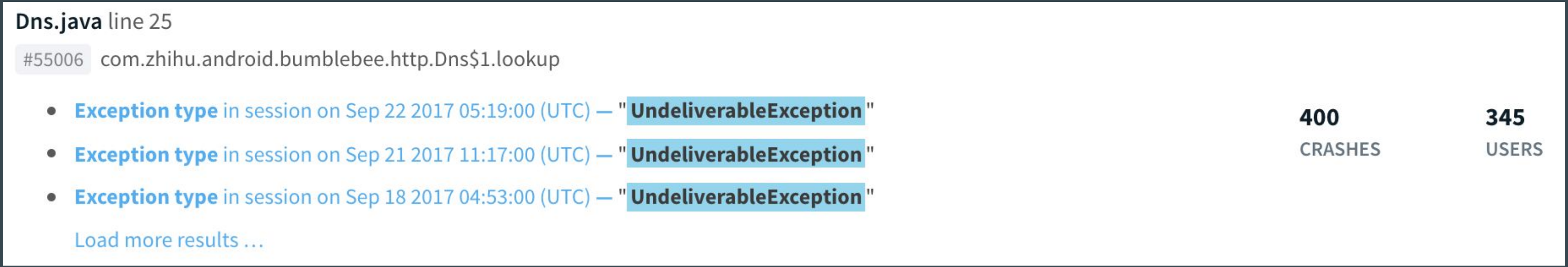
DbFeedFragment.java line 183

#61812 com.zhihu.android.app.db.fragment.DbFeedFragment.lambda$onSystemBarCreated$1

- **Exception type** in session on Sep 19 2017 00:54:00 (UTC) — "**UndeliverableException**"
- **Exception type** in session on Sep 12 2017 12:27:00 (UTC) — "**UndeliverableException**"
- **Exception type** in session on Sep 11 2017 00:11:00 (UTC) — "**UndeliverableException**"

Load more results …

1552 CRASHES **630** USERS



Dns.java line 25

#55006 com.zhihu.android.bumblebee.http.Dns$1.lookup

- **Exception type** in session on Sep 22 2017 05:19:00 (UTC) — "**UndeliverableException**"
- **Exception type** in session on Sep 21 2017 11:17:00 (UTC) — "**UndeliverableException**"
- **Exception type** in session on Sep 18 2017 04:53:00 (UTC) — "**UndeliverableException**"

Load more results …

400 CRASHES **345** USERS

知乎

Undeliverable Exception (2)

```
Observable.create(emitter -> {
    try {
        doSomething();
    } catch (Exception e) {
        emitter.onError(e);
    }
})
    .subscribe(rlt -> {
        // ...
    });
```

```
Observable.create(emitter -> {
    try {
        doSomething();
    } catch (Exception e) {
        if (!emitter.isDisposed()) {
            emitter.onError(e);
        }
        // emitter.tryOnError(e);
    }
})
    .subscribe(rlt -> {
        // ...
    }, throwable -> {
        // 处理错误
    });
```

知乎

关于 Dispose (1)

- dispose() 方法和 Thread.interrupt() 方法很类似
    - 只起到通知作用
    - 已经被 dispose() 的流，不能再次被 dispose()

- 异步任务内，应当通过 isDisposed() 方法判断是否要提前终止任务

关于 Dispose (2)

```
// Create 操作符
Disposable disposable =
    Observable.create(emitter -> {
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            System.out.print(“Interrupted”);
        }
    })
    .subscribeOn(Schedulers.computation())
    .subscribe();


disposable.dispose();
```

- 结果：输出 “Interrupted”
- Create 操作符创建的异步任务，在被 dispose() 时，其实内部调用了 Thread.interrupt()

知乎

关于 Dispose (3)

```
Observable.create(emitter -> {
    for (int i=0; i<1000; i++) {
        // CPU 密集操作

        if (emitter.isDisposed()) {
            // 如果流被 Dispose, 提前终止
            break;
        }
    }
});
```

- 通过 **isDisposed()** 判断是否应该提前结束任务，从而节省 **CPU** 计算资源

知乎

Rx 包裹异步操作 (1)

```
Disposable disposable =
    Observable.create(emitter -> {
        AsyncWork work = new AsyncWork()
        work.execute(rlt -> {
            emitter.onNext(rlt);
            emitter.onComplete();
        })
    })
    .subscribeOn(Schedulers.computation())
    .subscribe();
```

```
if (disposable.isDisposed()) {
    disposable.dispose();
}
```

Rx 包裹异步操作 (2)

```
class AsyncWorkDisposable implements
    Disposable {
    private AsyncWork work;

    AsyncWorkDisposable(AsyncWork work) {
        this.work = work;
    }

    @Override
    public void dispose() {
       work.cancel();
    }

    @Override
    public void dispose() {
        return work.isCanceled();
    }
}
```

```
class AsyncWorkObservable extends Observable<Rlt> {
    private AsyncWork work;

    AsyncWorkObservable(AsyncWork work) {
        this.work = work;
    }

    @Override
    public void subscribeActual(Observer observer) {
        AsyncWorkDisposable disposable =
                new AsyncWorkDisposable(work);
        observer.onSubscribe(disposable);

        work.execute(rlt -> {
            if (!disposable.isDisposed()) {
                emitter.onNext(rlt);
                emitter.onComplete();
            }
        })
    }
}
```

知乎

Thanks for watching

知乎

加入知乎

Android Team

HR Email：mifa@zhihu.com