RxAndroid基础知识

SpikeKing Readme — Latest commit 78cb911 on Jan 27

.idea	添加Menu和Fab, 设置点击事件.	2 months ago
app	最终实现.	2 months ago
gradle/wrapper	更新Gradle, 添加RxBinding功能	2 months ago
.gitignore	first commit	2 months ago
README.md	Readme	a month ago
build.gradle	更新Gradle, 添加RxBinding功能	2 months ago
gradle.properties	first commit	2 months ago
gradlew	first commit	2 months ago
gradlew.bat	first commit	2 months ago
settings.gradle	first commit	2 months ago

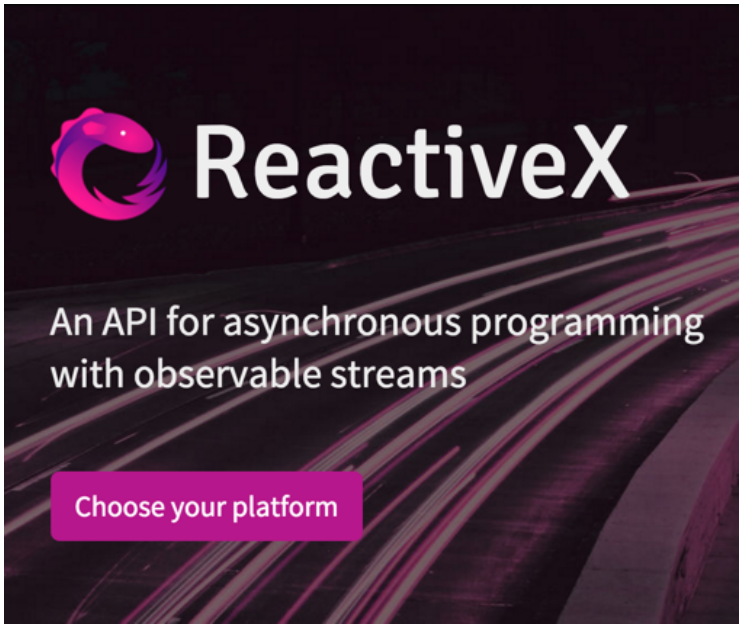
README.md

参考网址

> 欢迎Follow我的GitHub.

RxAndroid是RxJava的扩展, 可以优雅地处理异步请求. 以前的文章讲述过一些, 这次再补充些内容, 熟悉RxAndroid的使用方法.

> 要点包含: (1) 链式表达式的使用方式. (2) Lambda的应用. (3) Rx处理网络请求. (4) 线程自动管理, 防止内存泄露. (5) RxBinding绑定控件的异步事件.



1. 基础

当然, 从一个崭新的HelloWorld项目开始.

添加Gradle配置.

```
    compile 'com.jakewharton:butterknife:7.0.1'
    compile 'io.reactivex:rxandroid:1.1.0' // RxAndroid
    compile 'io.reactivex:rxjava:1.1.0' // 推荐同时加载RxJava
```

> RxAndroid是本文的核心依赖, 同时添加RxJava. 还有ButterKnife注解库.

Lambda表达式, 是写出优雅代码的关键, 参考.

```
plugins {
    id "me.tatarka.retrolambda" version "3.2.4"
}

android {
    ...

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

> `Gradle 2.1+` 以上, 配置非常简单, 添加一个plugin和一个Java1.8兼容即可.

从主 `MainActivity` 跳转至 `SimpleActivity` .

```
/**
 * 主Activity, 用于跳转各个模块.
 *
 * @author wangchenlong
 */
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    // 跳转简单的页面
    public void gotoSimpleModule(View view) {
        startActivity(new Intent(this, SimpleActivity.class));
    }

    // 跳转复杂的页面
    public void gotoMoreModule(View view) {
        startActivity(new Intent(this, MoreActivity.class));
    }

    // 跳转Lambda的页面
    public void gotoLambdaModule(View view) {
        startActivity(new Intent(this, LambdaActivity.class));
    }

    // 跳转网络的页面
    public void gotoNetworkModule(View view) {
        startActivity(new Intent(this, NetworkActivity.class));
    }

    // 跳转线程安全的页面
    public void gotoSafeModule(View view) {
        startActivity(new Intent(this, SafeActivity.class));
    }
}
```

在 `SimpleActivity` 中, 创建一个观察者, 收到字符串的返回.

```
    // 观察事件发生
    Observable.OnSubscribe mObservableAction = new Observable.OnSubscribe<String>() {
        @Override public void call(Subscriber<? super String> subscriber) {
            subscriber.onNext(sayMyName()); // 发送事件
            subscriber.onCompleted(); // 完成事件
        }
    };

...

    // 创建字符串
    private String sayMyName() {
        return "Hello, I am your friend, Spike!";
    }
```

创建两个订阅者, 使用字符串输出信息.

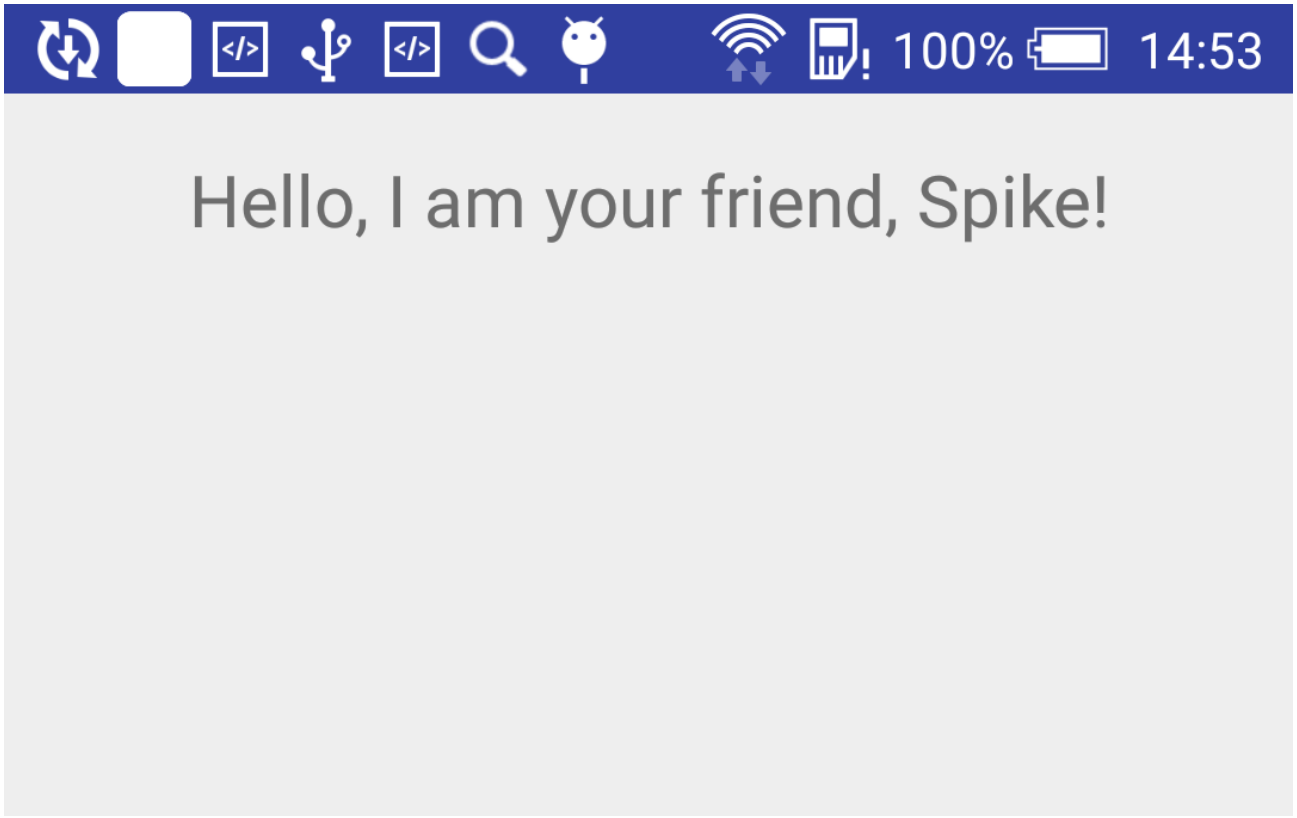
```
    // 订阅者, 接收字符串, 修改控件
    Subscriber<String> mTextSubscriber = new Subscriber<String>() {
        @Override public void onCompleted() {

        }

        @Override public void onError(Throwable e) {

        }

        @Override public void onNext(String s) {
            mTvText.setText(s); // 设置文字
```

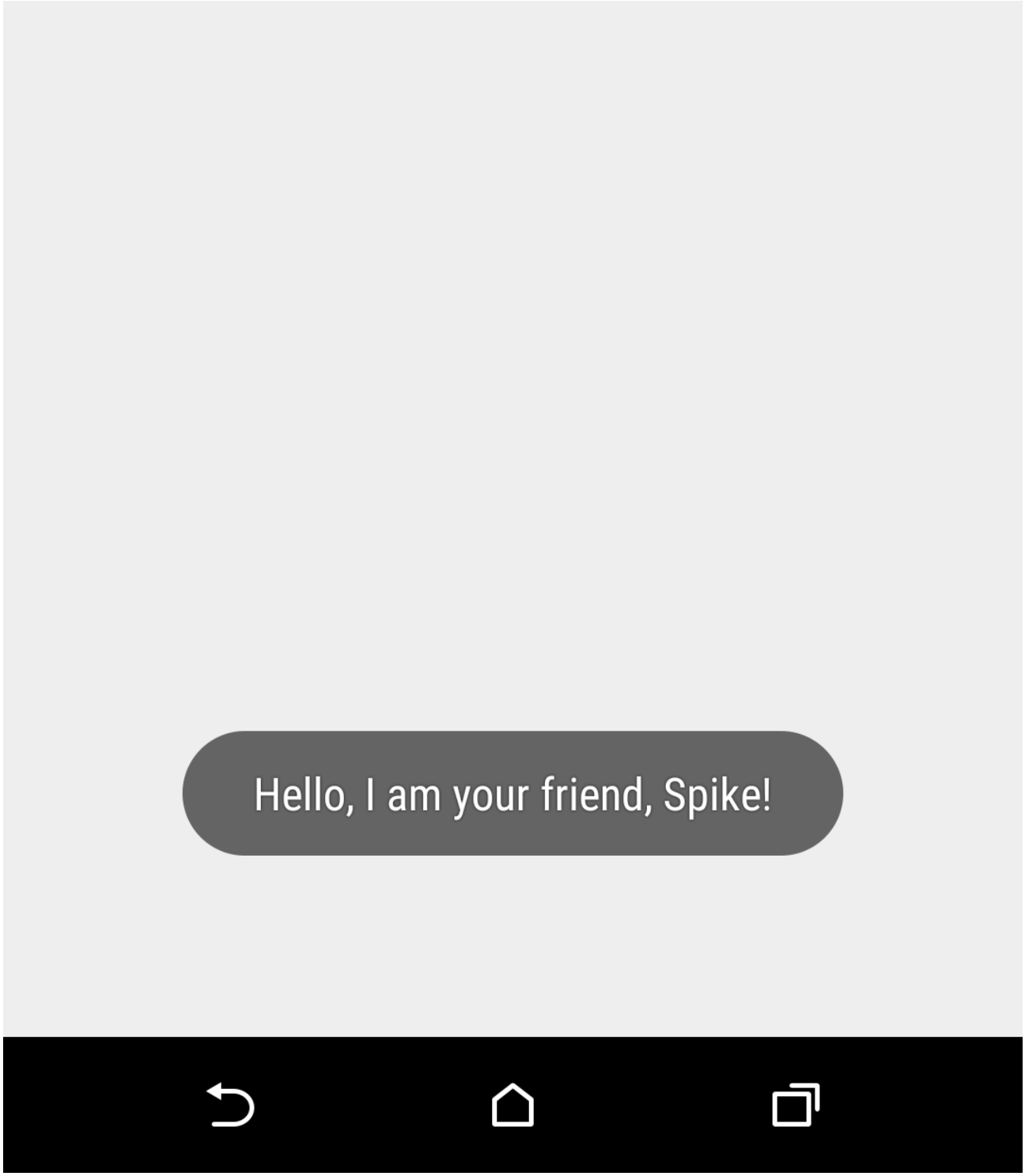
```
        }
    };

    // 订阅者, 接收字符串, 提示信息
    Subscriber<String> mToastSubscriber = new Subscriber<String>() {
        @Override public void onCompleted() {

        }

        @Override public void onError(Throwable e) {

        }

        @Override public void onNext(String s) {
            Toast.makeText(SimpleActivity.this, s, Toast.LENGTH_SHORT).show();
        }
    };
```

在页面中, 观察者接收信息, 发送至主线程 `AndroidSchedulers.mainThread()` , 再传递给订阅者, 由订阅者最终处理消息. 接收信息可以是同步, 也可以是异步.

```
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple);
        ButterKnife.bind(this);

        // 注册观察活动
        @SuppressWarnings("unchecked")
        Observable<String> observable = Observable.create(mObservableAction);

        // 分发订阅信息
        observable.observeOn(AndroidSchedulers.mainThread());
        observable.subscribe(mTextSubscriber);
        observable.subscribe(mToastSubscriber);
    }
```

最基础的RxAndroid使用.





2. 更多

我们已经熟悉了初步的使用方式, 在接着学习一些其他方法, 如

`just`：获取输入数据, 直接分发, 更加简洁, 省略其他回调. `from`：获取输入数组, 转变单个元素分发. `map`：映射, 对输入数据进行转换, 如大写. `flatMap`：增大, 本意就是增肥, 把输入数组映射多个值, 依次分发. `reduce`：简化, 正好相反, 把多个数组的值, 组合成一个数据.

来看看这个示例, 设置两个不同类型数组, 作为输入源, 根据不同情况分发数据.

```
/**
 * 更多的RxAndroid的使用方法.
 * <p>
 * Created by wangchenlong on 15/12/30.
 */
public class MoreActivity extends Activity {

    @Bind(R.id.simple_tv_text) TextView mTvText;

    final String[] mManyWords = {"Hello", "I", "am", "your", "friend", "Spike"};
    final List<String> mManyWordList = Arrays.asList(mManyWords);

    // Action类似订阅者, 设置TextView
    private Action1<String> mTextViewAction = new Action1<String>() {
        @Override public void call(String s) {
            mTvText.setText(s);
        }
    };

    // Action设置Toast
    private Action1<String> mToastAction = new Action1<String>() {
        @Override public void call(String s) {
            Toast.makeText(MoreActivity.this, s, Toast.LENGTH_SHORT).show();
        }
    };

    // 设置映射函数
    private Func1<List<String>, Observable<String>> mOneLetterFunc = new Func1<List<String>, Observable<Str
        @Override public Observable<String> call(List<String> strings) {
            return Observable.from(strings); // 映射字符串
        }
    };

    // 设置大写字母
    private Func1<String, String> mUpperLetterFunc = new Func1<String, String>() {
        @Override public String call(String s) {
            return s.toUpperCase(); // 大小字母
        }
    };

    // 连接字符串
    private Func2<String, String, String> mMergeStringFunc = new Func2<String, String, String>() {
        @Override public String call(String s, String s2) {
            return String.format("%s %s", s, s2); // 空格连接字符串
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple);
        ButterKnife.bind(this);

        // 添加字符串, 省略Action的其他方法, 只使用一个onNext.
        Observable<String> obShow = Observable.just(sayMyName());

        // 先映射, 再设置TextView
        obShow.observeOn(AndroidSchedulers.mainThread())
                .map(mUpperLetterFunc).subscribe(mTextViewAction);

        // 单独显示数组中的每个元素
        Observable<String> obMap = Observable.from(mManyWords);

        // 映射之后分发
        obMap.observeOn(AndroidSchedulers.mainThread())
                .map(mUpperLetterFunc).subscribe(mToastAction);
```

```
        // 优化过的代码, 直接获取数组, 再分发, 再合并, 再显示toast, Toast顺次执行.
        Observable.just(mManyWordList)
                .observeOn(AndroidSchedulers.mainThread())
                .flatMap(mOneLetterFunc)
                .reduce(mMergeStringFunc)
                .subscribe(mToastAction);
    }

    // 创建字符串
    private String sayMyName() {
        return "Hello, I am your friend, Spike!";
    }
}
```

> 这次简化调用代码, 因为有时候我们对异常并不是很关心, 只要能 `catch` 异常即可, 因此流仅仅关注真正需要的部分.

输入字符串, 变换大写, 输出至控件中显示.

```
        // 添加字符串, 省略Action的其他方法, 只使用一个onNext.
        Observable<String> obShow = Observable.just(sayMyName());

        // 先映射, 再设置TextView
        obShow.observeOn(AndroidSchedulers.mainThread())
                .map(mUpperLetterFunc).subscribe(mTextViewAction);
```

> `just` 可以非常简单的获取任何数据, 分发时, 选择使用的线程. `map` 是对输入数据加工, 转换类型, 输入 `Func1` , 准换大写字母. `Func1` 代表使用一个参数的函数, 前面是参数, 后面是返回值. `Action1` 代表最终动作, 因而不需要返回值, 并且一个参数.

输入数组, 单独分发数组中每一个元素, 转换大写, 输入Toast连续显示.

```
        // 单独显示数组中的每个元素
        Observable<String> obMap = Observable.from(mManyWords);

        // 映射之后分发
        obMap.observeOn(AndroidSchedulers.mainThread())
                .map(mUpperLetterFunc).subscribe(mToastAction);
```

> `from` 是读取数组中的值, 每次单独分发, 并分发多次, 其余类似.

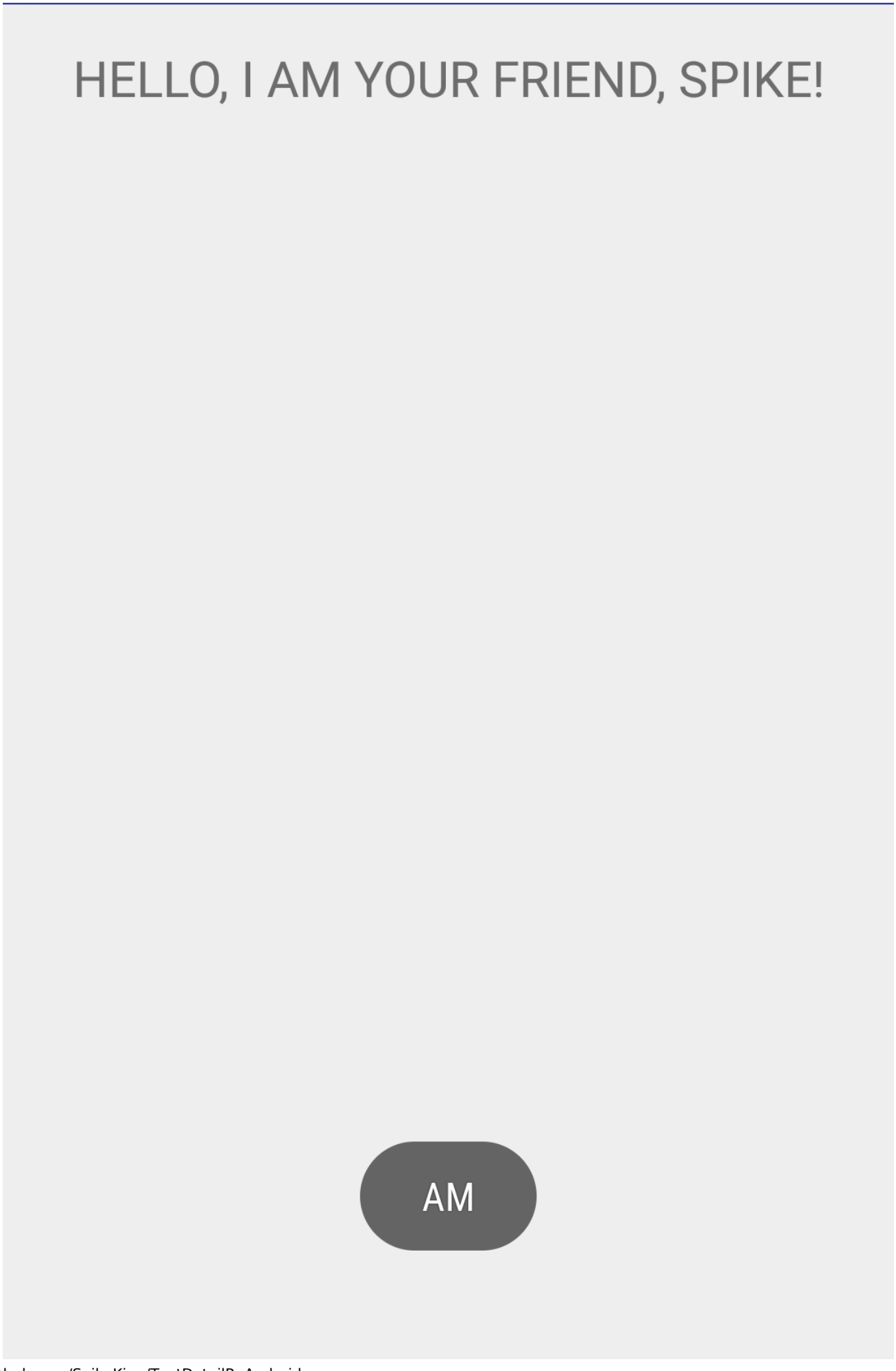
输入数组, 映射为单独分发, 并组合到一起, 集中显示.

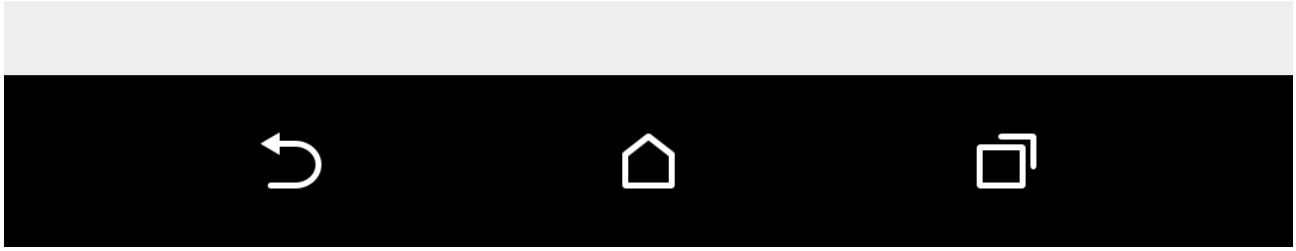
```
        // 优化过的代码, 直接获取数组, 再分发, 再合并, 再显示toast, Toast顺次执行.
        Observable.just(mManyWordList)
                .observeOn(AndroidSchedulers.mainThread())
                .flatMap(mOneLetterFunc)
                .reduce(mMergeStringFunc)
                .subscribe(mToastAction);
```

> 这次是使用 `just` 分发数组, 则分发数据就是数组, 并不是数组中的元素. `flatMap` 把数组转换为单独分发, `Func1` 内部使用 `from` 拆分数组. `reduce` 把单独分发数据集中到一起, 再统一分发, 使用 `Func2` . 最终使用 `Action1` 显示获得数据. 本次代码也更加简洁.

由此我们可以观察到, Rx的写法可以是多种多样, 合理的写法会更加优雅.

效果


HELLO, I AM YOUR FRIEND, SPIKE!
AM



3. Lambda

Lambda表达式和Rx非常契合, 可以省略大量的内部类, 如Func和Action. 我们把上个示例, 用Lambda再写一次, 功能相同.

```
/**
 * Lambda表达式写法
 * <p>
 * Created by wangchenlong on 15/12/31.
 */
public class LambdaActivity extends Activity {

    @Bind(R.id.simple_tv_text) TextView mTvText;

    final String[] mManyWords = {"Hello", "I", "am", "your", "friend", "Spike"};
    final List<String> mManyWordList = Arrays.asList(mManyWords);

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple);
        ButterKnife.bind(this);

        // 添加字符串, 省略Action的其他方法, 只使用一个onNext.
        Observable<String> obShow = Observable.just(sayMyName());

        // 先映射, 再设置TextView
        obShow.observeOn(AndroidSchedulers.mainThread())
                .map(String::toUpperCase).subscribe(mTvText::setText);

        // 单独显示数组中的每个元素
        Observable<String> obMap = Observable.from(mManyWords);

        // 映射之后分发
        obMap.observeOn(AndroidSchedulers.mainThread())
                .map(String::toUpperCase)
                .subscribe(this::showToast);

        // 优化过的代码, 直接获取数组, 再分发, 再合并, 再显示toast, Toast顺次执行.
        Observable.just(mManyWordList)
                .observeOn(AndroidSchedulers.mainThread())
                .flatMap(Observable::from)
                .reduce(this::mergeString)
                .subscribe(this::showToast);
    }

    // 创建字符串
    private String sayMyName() {
        return "Hello, I am your friend, Spike!";
    }

    // 显示Toast
    private void showToast(String s) {
        Toast.makeText(LambdaActivity.this, s, Toast.LENGTH_SHORT).show();
    }
```

```
    // 合并字符串
    private String mergeString(String s1, String s2) {
        return String.format("%s %s", s1, s2);
    }
}
```

> 这次没有使用常规的Lambda表达式, 而是更简单的 `方法引用(Method References)` . 方法引用: 方法参数和返回值与Lambda表达式相同时, 使用方法名代替.

4. 网络请求

Retrofit是网络请求库, 刚推出2.0版本. Rx的一个核心应用就是处理异步网络请求, 结合Retrofit, 会更加方便和简洁. 参考.

引入库

```
    compile 'com.android.support:recyclerview-v7:23.1.1' // RecyclerView

    compile 'com.squareup.retrofit:retrofit:2.0.0-beta2' // Retrofit网络处理
    compile 'com.squareup.retrofit:adapter-rxjava:2.0.0-beta2' // Retrofit的rx解析库
    compile 'com.squareup.retrofit:converter-gson:2.0.0-beta2' // Retrofit的gson库

    compile 'com.squareup.picasso:picasso:2.5.2' // Picasso网络图片加载
```

> `recyclerview` 和 `picasso` 为了显示. `retrofit` 系列是网络请求.

主页使用一个简单的列表视图, 展示Github的用户信息.

```
/**
 * Rx的网络请求方式
 * <p>
 * Created by wangchenlong on 15/12/31.
 */
public class NetworkActivity extends Activity {

    @Bind(R.id.network_rv_list) RecyclerView mRvList; // 列表

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_network);
        ButterKnife.bind(this);

        // 设置Layout管理器
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        layoutManager.setOrientation(LinearLayoutManager.VERTICAL);
        mRvList.setLayoutManager(layoutManager);

        // 设置适配器
        UserListAdapter adapter = new UserListAdapter(this::gotoDetailPage);
        NetworkWrapper.getUsersInto(adapter);
        mRvList.setAdapter(adapter);
    }

    // 点击的回调
    public interface UserClickCallback {
        void onItemClicked(String name);
    }

    // 跳转到库详情页面
    private void gotoDetailPage(String name) {
```

```
            startActivity(NetworkDetailActivity.from(NetworkActivity.this, name));
        }
    }
```

> 在列表中提供点击用户信息跳转至用户详情. `NetworkWrapper.getUsersInto(adapter)` 请求网络, 设置适配器信息.

关键部分, 适配器, 其中包含ViewHolder类和数据类.

```
/**
 * 显示列表
 * <p>
 * Created by wangchenlong on 15/12/31.
 */
public class UserListAdapter extends RecyclerView.Adapter<UserListAdapter.UserViewHolder> {

    private List<GitHubUser> mUsers; // 用户名集合

    private NetworkActivity.UserClickCallback mCallback; // 用户点击项的回调

    public UserListAdapter(NetworkActivity.UserClickCallback callback) {
        mUsers = new ArrayList<>();
        mCallback = callback;
    }

    public void addUser(GitHubUser user) {
        mUsers.add(user);
        notifyItemInserted(mUsers.size() - 1); // 最后一位
    }

    @Override public UserViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View item = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_network_user, parent, false);
        return new UserViewHolder(item, mCallback);
    }

    @Override public void onBindViewHolder(UserViewHolder holder, int position) {
        holder.bindTo(mUsers.get(position));
    }

    @Override public int getItemCount() {
        return mUsers.size();
    }

    // Adapter的ViewHolder
    public static class UserViewHolder extends RecyclerView.ViewHolder {

        @Bind(R.id.network_item_iv_user_picture) ImageView mIvUserPicture;
        @Bind(R.id.network_item_tv_user_name) TextView mTvUserName;
        @Bind(R.id.network_item_tv_user_login) TextView mTvUserLogin;
        @Bind(R.id.network_item_tv_user_page) TextView mTvUserPage;

        public UserViewHolder(View itemView, NetworkActivity.UserClickCallback callback) {
            super(itemView);
            ButterKnife.bind(this, itemView);
            // 绑定点击事件
            itemView.setOnClickListener(v ->
                    callback.onItemClicked(mTvUserLogin.getText().toString()));
        }

        // 绑定数据
        public void bindTo(GitHubUser user) {
            mTvUserName.setText(user.name);
            mTvUserLogin.setText(user.login);
            mTvUserPage.setText(user.repos_url);
```

```
                Picasso.with(mIvUserPicture.getContext())
                        .load(user.avatar_url)
                        .placeholder(R.drawable.ic_person_black_24dp)
                        .into(mIvUserPicture);
            }
        }

        // 用户类, 名称必须与Json解析相同
        public static class GitHubUser {
            public String login;
            public String avatar_url;
            public String name;
            public String repos_url;
        }
    }
```

> 添加数据 `addUser` , 其中 `notifyItemInserted` 通知更新. 可以自动生成Json解析类的网站.

首先创建 `Retrofit` `服务, 通过服务获取数据, 再依次分发给适配器.

```
    /**
     * 用户获取类
     * <p>
     * Created by wangchenlong on 15/12/31.
     */
    public class NetworkWrapper {
        private static final String[] mFamousUsers =
                {"SpikeKing", "JakeWharton", "rock3r", "Takhion", "dextorer", "Mariuxtheone"};

        // 获取用户信息
        public static void getUsersInto(final UserListAdapter adapter) {
            GitHubService gitHubService =
                    ServiceFactory.createServiceFrom(GitHubService.class, GitHubService.ENDPOINT);

            Observable.from(mFamousUsers)
                    .flatMap(gitHubService::getUserData)
                    .subscribeOn(Schedulers.newThread())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(adapter::addUser);
        }

        // 获取库信息
        public static void getReposInfo(final String username, final RepoListAdapter adapter) {
            GitHubService gitHubService =
                    ServiceFactory.createServiceFrom(GitHubService.class, GitHubService.ENDPOINT);

            gitHubService.getRepoData(username)
                    .flatMap(Observable::from)
                    .subscribeOn(Schedulers.newThread())
                    .observeOn(AndroidSchedulers.mainThread())
                    .subscribe(adapter::addRepo);
        }
    }
```

> 网络请求无法在主线程上执行, 需要启动异步线程, 如 `Schedulers.newThread()` . 使用工厂模式 `ServiceFactory` 创建服务, 也可以单独创建服务.

创建 `Retrofit` 服务的工厂类.

```
    /**
     * 工厂模式
     * <p>
     * Created by wangchenlong on 15/12/31.
     */
```

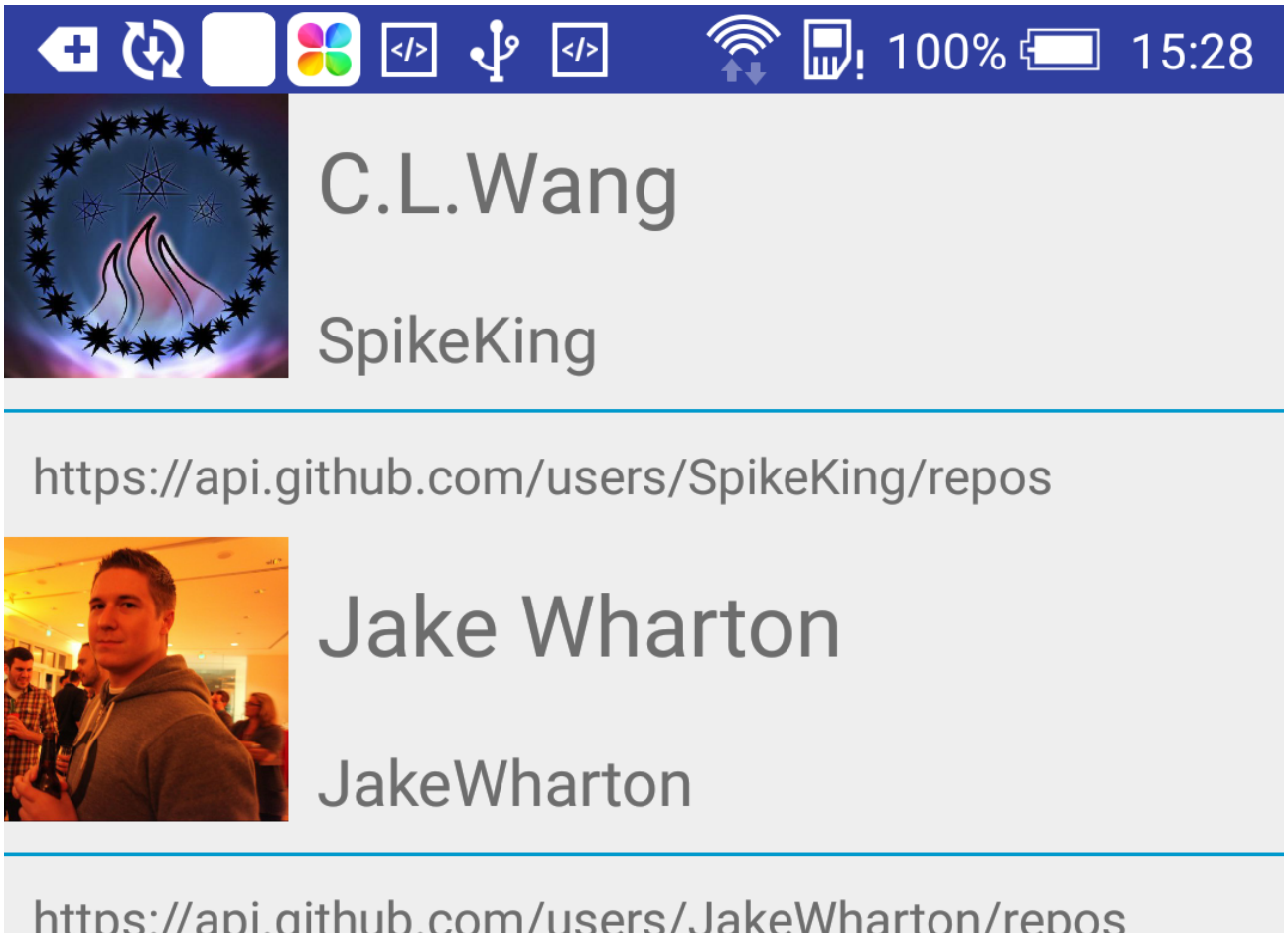
```
public class ServiceFactory {
    public static <T> T createServiceFrom(final Class<T> serviceClass, String endpoint) {
        Retrofit adapter = new Retrofit.Builder()
                .baseUrl(endpoint)
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create()) // 添加Rx适配器
                .addConverterFactory(GsonConverterFactory.create()) // 添加Gson转换器
                .build();
        return adapter.create(serviceClass);
    }
}
```

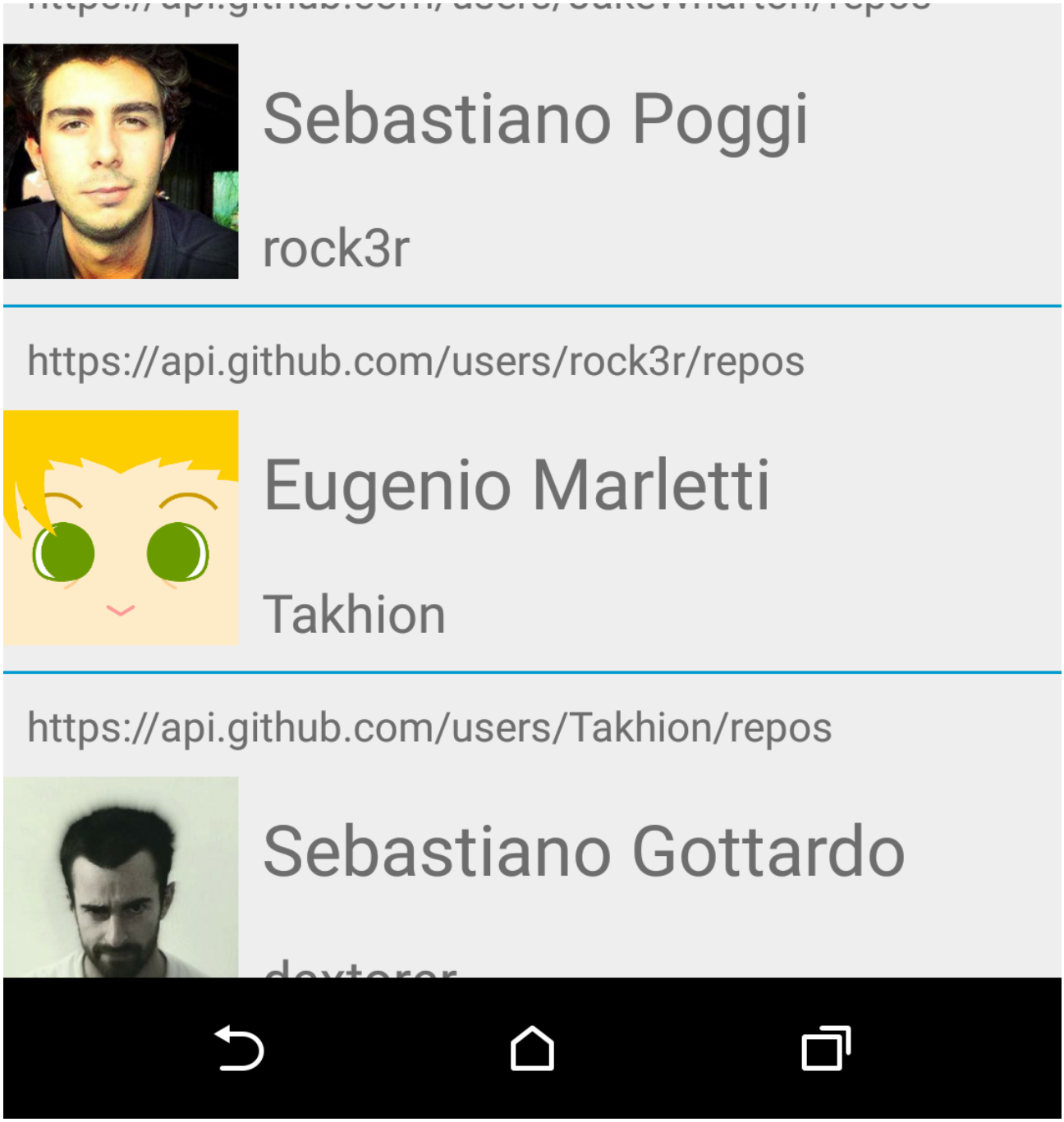
> 这是Retrofit 2.0的写法, 注意需要添加Rx和Gson的解析.

设置网络请求的Url.

```
/**
 * GitHub的服务
 * <p>
 * Created by wangchenlong on 15/12/31.
 */
public interface GitHubService {
    String ENDPOINT = "https://api.github.com";

    // 获取个人信息
    @GET("/users/{user}")
    Observable<UserListAdapter.GitHubUser> getUserData(@Path("user") String user);

    // 获取库, 获取的是数组
    @GET("/users/{user}/repos")
    Observable<RepoListAdapter.GitHubRepo[]> getRepoData(@Path("user") String user);
}
```

显示用户





> 详情页面与主页类似, 参考代码, 不做细说.

5. 线程安全

Rx的好处之一就是可以防止内存泄露, 即根据页面生命周期, 处理异步线程的结束. 可以使用RxLifecycle库处理生命周期.

`Activity` 类继承 `RxAppCompatActivity` , 替换 `AppCompatActivity` .

启动一个循环线程.

```
/**
 * Rx的线程安全
 * <p>
 * Created by wangchenlong on 15/12/31.
 */
```

```
public class SafeActivity extends RxAppCompatActivity {
    private static final String TAG = "DEBUG-WCL: " + SafeActivity.class.getSimpleName();

    @Bind(R.id.simple_tv_text) TextView mTvText;

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple);
        ButterKnife.bind(this);

        Observable.interval(1, TimeUnit.SECONDS)
                .observeOn(AndroidSchedulers.mainThread())
                .compose(bindToLifecycle()) // 管理生命周期, 防止内存泄露
                .subscribe(this::showTime);
    }

    private void showTime(Long time) {
        mTvText.setText(String.valueOf("时间计数: " + time));
        Log.d(TAG, "时间计数器: " + time);
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.w(TAG, "页面关闭!");
    }
}
```

继承 `RxAppCompatActivity` , 添加 `bindToLifecycle` 方法管理生命周期. 当页面 `onPause` 时, 会自动结束循环线程. 如果注释这句代码, 则会导致内存泄露.

6. RxBinding

RxBinding是Rx中处理控件异步调用的方式, 也是由Square公司开发, Jake负责编写. 通过绑定组件, 异步获取事件, 并进行处理. 编码风格非常优雅.

除了RxJava, 再添加RxBinding的依赖.

```
    // RxBinding
    compile 'com.jakewharton.rxbinding:rxbinding:0.3.0'
    compile 'com.jakewharton.rxbinding:rxbinding-appcompat-v7:0.3.0'
    compile 'com.jakewharton.rxbinding:rxbinding-design:0.3.0'
```

Toolbar和Fab, 两个较新的控件.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:orientation="vertical"
    tools:context=".BindingActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
```

```
            android:id="@+id/rxbinding_t_toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:popupTheme="@style/AppTheme.PopupOverlay"
            tools:targetApi="21"/>

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_rxbinding"/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/rxbinding_fab_fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@android:drawable/ic_dialog_email"/>

</android.support.design.widget.CoordinatorLayout>
```

两个EditText控件, 对比传统方法和RxBinding方法.

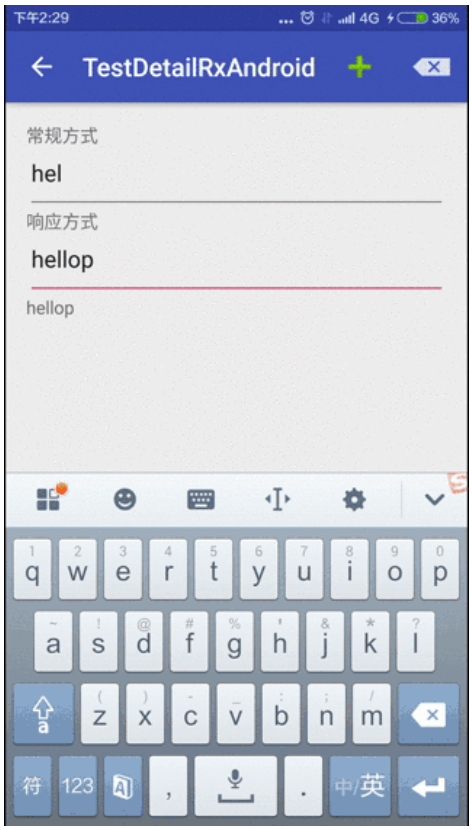
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:app="http://schemas.android.com/apk/res-auto"
              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:padding="@dimen/activity_margin"
              app:layout_behavior="@string/appbar_scrolling_view_behavior"
              tools:context=".BindingActivity"
              tools:showIn="@layout/activity_binding">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/usual_approach"/>

    <EditText
        android:id="@+id/rxbinding_et_usual_approach"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:hint="@null"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/reactive_approach"/>

    <EditText
        android:id="@+id/rxbinding_et_reactive_approach"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:hint="@null"/>

    <TextView
        android:id="@+id/rxbinding_tv_show"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

使用ButterKnife注入控件, 使用RxBinding方式绑定控件, 异步监听事件.

```
/**
 * Rx绑定
 * <p>
 * Created by wangchenlong on 16/1/25.
 */
public class BindingActivity extends AppCompatActivity {

    @Bind(R.id.rxbinding_t_toolbar) Toolbar mTToolbar;
    @Bind(R.id.rxbinding_et_usual_approach) EditText mEtUsualApproach;
    @Bind(R.id.rxbinding_et_reactive_approach) EditText mEtReactiveApproach;
    @Bind(R.id.rxbinding_tv_show) TextView mTvShow;
    @Bind(R.id.rxbinding_fab_fab) FloatingActionButton mFabFab;

    @Override protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_binding);
        ButterKnife.bind(this);

        initToolbar(); // 初始化Toolbar
        initFabButton(); // 初始化Fab按钮
        initEditText(); // 初始化编辑文本
    }

    // 初始化Toolbar
    private void initToolbar() {
        // 添加菜单按钮
        setSupportActionBar(mTToolbar);
        ActionBar actionBar = getSupportActionBar();
        // 添加浏览按钮
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
        }

        RxToolbar.itemClicks(mTToolbar).subscribe(this::onToolbarItemClicked);

        RxToolbar.navigationClicks(mTToolbar).subscribe(this::onToolbarNavigationClicked);
    }

    // 点击Toolbar的项
    private void onToolbarItemClicked(MenuItem menuItem) {
        String m = "点击\"" + menuItem.getTitle() + "\"";
        Toast.makeText(this, m, Toast.LENGTH_SHORT).show();
    }

    // 浏览点击
    private void onToolbarNavigationClicked(Void v) {
        Toast.makeText(this, "浏览点击", Toast.LENGTH_SHORT).show();
    }

    @Override public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_rxbinding, menu);
        return super.onCreateOptionsMenu(menu);
    }

    // 初始化Fab按钮
    private void initFabButton() {
        RxView.clicks(mFabFab).subscribe(this::onFabClicked);
    }

    // 点击Fab按钮
    private void onFabClicked(Void v) {
        Snackbar snackbar = Snackbar.make(findViewById(android.R.id.content), "点击Snackbar", Snackbar.LENGT
        snackbar.show();
        RxSnackbar.dismisses(snackbar).subscribe(this::onSnackbarDismissed);
    }
```

```
        // 销毁Snackbar, event参考{Snackbar}
        private void onSnackbarDismissed(int event) {
            String text = "Snackbar消失代码:" + event;
            Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
        }

        // 初始化编辑文本
        private void initEditText() {
            // 正常方式
            mEtUsualApproach.addTextChangedListener(new TextWatcher() {
                @Override
                public void beforeTextChanged(CharSequence s, int start, int count, int after) {

                }

                @Override public void onTextChanged(CharSequence s, int start, int before, int count) {
                    mTvShow.setText(s);
                }

                @Override public void afterTextChanged(Editable s) {

                }

            });

            // Rx方式
            RxTextView.textChanges(mEtReactiveApproach).subscribe(mTvShow::setText);
        }
    }
```

> Toolbar使用RxToolbar监听点击事件; Snackbar使用RxSnackbar监听; EditText使用RxTextView监听; 其余使用RxView监听.



OK, That's all! Enjoy It!