Fork me on GitHub

Open Android

開源開發

開源

Fresco

RxJava

espresso

UI TESTING FOR ANDROID

Proguard Annotation

目錄

Android App 開源開發

- gitbook: http://yongjhih.gitbooks.io/feed/

- github: https://github.com/yongjhih/android-gitbook/

開放書籍，歡迎共筆修改，以 gitbook 方式瀏覽時，可點擊頁面上方的編輯頁面連結 "EDIT THIS PAGE" 即可共筆。

善用開源來開發 Android App。

前端：

- RxJava, RxAndroid
- retrolambda (java7 + java8)
- retrofit, NotRetrofit
- okhttp
- DI: Dagger2
- AutoParcel, AutoValue, etc.
- ImageLoader: fresco, AUIL, picasso, glide, etc.
- json2pojo: jackson, gson, logansquare, etc.
- Orm: DBFlow, etc.
- SimpleFacebook, RetroFacebook
- EventBus/Otto
- mockito
- espresso
- robolectric
- assertj(fest), truth
- icepick
- auto-parse
- RxParse
- RxFacebook
- proguard annotations
- proguard snippets
- json2notification
- Kotlin
- anko
- ReactNative
- jitpack
- bintray

後端：

- Parse BAAS

開發環境：

- Android Studio
- gradle

專案環境：

- gitlab
- phabricator

持續整合環境：

- jenkins

版權

RxJava

#Promise #Reactive #Functional #Monad #Stream

前言

RxJava, Reactive Java, 一個 Java FRP (Functional Reactive Programming) 的實現。

可以接龍的 callbacks。

類似於 command line 的 `|` 的概念。

```
ls | wc -l
```

RxJava 大約從 2013 年中一場研討會之後漸漸嶄露頭角。大約一年後 2014 年中後開始許多文章出現(英文)。2014 年底 trello 甚至應聘熟悉 RxJava 的開發者。

有誰在用？先從有貢獻來看：

- Square
- SoundCloud
- NetFlix

短期效用：有效避免巢狀 callback 增加可讀性以及減少 `List<item>` 的轉換成本。

鑑於 RxJava 太少範例可以參考，才開始撰寫這篇文章偏向範例而不是從學術性角度介紹起，而這篇文章的技術來源大多是閱讀 RxJava 源碼、測試程式源碼以及其 github Wiki 而來。如果有謬誤之處，歡迎用各種方式聯絡我。

1. 注意：這邊直接使用 *lambda λ* 的表達式，如果你還不清楚，請跳轉到 *Lambda*
2. *java8.stream* 也實現了 *Reactive*。

先看一些範例對照後，了解樣貌之後，我們再來討論 RxJava 基本使用概念與方法。

有效解決重複的 loop 增進效能，維持同個 loop

假設我們有一萬名使用者 `List<User> users`，其中有五千名女性使用者。

列出使用者年齡：

Before:

```
List<Integer> getAgeList(List<User> users) {
    List<Integer> ageList = new ArrayList<>();

    for (User user : users) {
        ageList.add(user.getAge());
    }

    return ageList;
}
```

After:

```
Observable<Integer> getAgeObs(List<User> users) {
    return Observable.from(users).map(user -> user.getAge());
}

// 如果你堅持一定要傳遞 List
List<Integer> getAgeList(List<User> users) {
    return getAgeObs(users).toList().toBlocking().single();
}
```

列出女性使用者：

Before:

```
List<User> getFemaleList(/* @Writable */List<User> users) {
    List<User> femaleList = users;
    Iterator<User> it = users.iterator();

    while (it.hasNext()) {
        if (it.getGender() != User.FEMALE) it.remove();
    }

    return femaleList;
}
```

After:

```
Observable<User> getFemaleObs(List<User> users) {
    return Observable.from(users).filter(user -> user.getGender() =
}

// 如果你堅持一定要傳遞 List
List<User> getFemaleList(List<User> users) {
    return getFemaleObs(users).toList().toBlocking().single();
}
```

組合一下就可以列出女性使用者年齡：

Before:

```
List<Integer> getFemaleAgeList(List<User> users) {
    getAgeList(getFemaleList(users)); // 如果不改變寫法，會整整跑完兩個
}
```

你可以發現原本的寫法有個瑕疵，就是會先繞完一萬使用者，找出來五千名女性後，為了詢問年紀，只好再繞一次這五千名女性。

可以第一次找出女性使用者時，就順便問一下年紀嗎？

為了避免重複迴圈，你可改變寫法，以沿用 loop ：

```
List<Integer> getFemaleAgeList(List<User> users) {
    //getAgeList(getFemaleList(users));
    List<Integer> ageList = new ArrayList<>();

    for (User user : users) {
        if (user.getGender() == User.FEMALE) {
            ageList.add(user.getAge());
        }
    }

    return ageList;
}
```

而 Observable 不用刻意改變寫法，直接組起來就好：

After:

```
Observable<Integer> getFemaleAgeObs(List<User> users) {
    return getFemaleObs(users).map(user -> user.getAge());
}
```

你可以發現維持一樣的寫法，它會同時做兩件事情：找出女性順便詢問年紀(過濾與轉換)，避免重複的迴圈。

提前打斷迴圈的能力，避免不必要的過濾與轉換

列出一百名女性使用者：

Before:

```
List<User> getFemaleList(List<User> users, int limit) {
    return getFemaleList(users).subList(0, limit);
}

// 這裡會完整繞完萬名使用者，找出千名女性使用者後，才抽出百名。
getFemaleList(users, 100);
```

After:

```
Observable<User> getFemaleObs(List<User> users, int limit) {
    return getFemaleObs(users).take(limit);
}

// 這裡會蒐集到百名女性使用者後，即停止。
getFemaleObs(users, 100);
```

我們可以從這裡看到差異，儘管你只要找出百名女性使用者，原本的寫法卻會繞完萬名使用者，找出所有女性使用者，再分割前一百名。 而 Observable 會聰明的找到第一百名女性使用者就馬上停止。原本的寫法要做到提前停止，就必須改寫：

```
List<User> getFemaleList(/* @Writable */List<User> users) { ... }

List<User> getFemaleList(List<User> users, int limit) {
    //return getFemaleList(users).subList(0, limit);
    List<User> femaleList = new ArrayList<>();

    int i = 0;
    for (User user : users) {
        if (i >= limit) break;
        if (user.getGender() == User.FEMALE) femaleList.add(user);
        i++;
    }

    return femaleList;
}

// 比較靈活一點的寫法，提供 predicate function
List<User> getFemaleList(List<User> users, Func2<Boolean, User, Int
    List<User> femaleList = new ArrayList<>();

    int i = 0;
    for (User user : users) {
        if (!predicate.call(user, i)) break;
        if (user.getGender() == User.FEMALE) femaleList.add(user);
        i++;
    }

    return femaleList;
}

getFemaleList(users, 100);
getFemaleList(users, (user, i) -> i <= 100); // predicate Func2
```

這種靈活的方法套用在各個資料流身上，也就是 RxJava 所提供的 operators 。

接下來，開始一點組合應用：

列出前百名女性使用者年齡：

Before:

```
List<Integer> getFemaleAgeList(List<User> users, int limit) {
    return getAgeList(getFemaleList(users, limit));
}
```

After:

```
List<Integer> getFemaleAgeList(List<User> users, int limit) {
    return Observable.from(users)
        .filter(user -> user.getGender() == User.FEMALE)
        .take(limit)
        .map(user -> user.getAge())
        .toList().toBlocking().single(); // 如果你堅持一定要傳遞 List
}
```

你可以之後才決定選幾筆，Observable 選幾筆才作幾筆過濾與轉換，有效避免無謂的全數過濾與轉換。

你可以把界面維持 Observable 傳遞，維持一樣的撰寫方法，直接組裝起來就好：

```
Observable<User> getFemaleObs(Observable<User> userObs) {
    return userObs.filter(user -> user.getGender() == User.FEMALE);
}

Observable<Integer> getAgeObs(Observable<User> userObs) {
    return userObs.map(user -> p.getAge());
}

Observable<Integer> getFemaleAgeObs(List<User> users) {
    return getAgeObs(getFemaleObs(Observable.from(users)));
}
```

只做 100 筆過濾與轉換(女性、年齡)：

```
getFemaleAgeObs(users)
    .take(100) // 拿個 100 筆
    .toList().toBlocking().single();
```

拉平巢狀 **callback** 增加易讀性

Before:

```
void login(Activity activity, LoginListenr loginListener) {
    loginFacebook(activity, fbUser -> {
        getFbProfile(fbUser, fbProfile -> {
            loginParse(fbProfile, parseUser -> {
                getParseProfile(fbProfile, parseProfile -> {
                    loginListener.onLogin(parseProfile);
                });
            });
        });
    });
}
```

After:

```
void login(Activity activity, LoginListener loginListener) {
    Observable.just(activity)
        .flatMap(activity -> loginFacebook(activity))
        .flatMap(fbUser -> getFbProfile(fbUser))
        .flatMap(fbProfile -> loginParse(fbProfile))
        .flatMap(parseUser -> getParseProfile(parseUser))
        .subscribe(parseProfile -> loginListener.onLogin(parseProfi
}

// wrap callback functions in Observable<?>
Observable<FbUser> loginFacebook(Activity activity) {
    return Observable.create(sub -> {
        loginFacebook(activity, fbUser -> {
            sub.onNext(fbUser);
            sub.onCompleted();
        });
    });
}

Observable<FbProfile> getFbProfile(FbUser fbUser) { ... }
Observable<ParseUser> loginParse(FbProfile fbProfile) { ... }
Observable<ParseProfile> getParseProfile(ParseUser parseUser) { ...
```

如何導入套用與改變撰寫

既有長時間存取的函式改成 **Observable**

...

```
File download(String url) { ... return file; }

Observable<File> downloadObs(String url) {
    return Observable.defer(() -> Observable.just(download(url)));
}
```

...

```
downloadObs(url).subscribeOn(Schedulers.io()).subscribe(file -> {
  System.out.println(file);
});
```

既有的 callback 改成 Observable

...

```
Observable<ParseUser> loginParseWithFacebook(Activity activity) {
    return Observable.create(sub -> { // Observable.OnSubscriber
        ParseFacebookUtils.logIn(Arrays.asList("public_profile", "e
            @Override
            public void done(final ParseUser parseUser, ParseExcept
                if (err != null) {
                    sub.onError(err);
                } else {
                    sub.onNext(parseUser);
                    sub.onCompleted();
                }
            }
        });
    })
}
```

...

```
loginParseWithFacebook(activity).subscribe(parseUser -> {
  System.out.println(parseUser);
});
```

轉換 map()

我們經常把 `List<T>` 轉成 `List<R>`，如： `List<TextView>` 轉成 `List<String>`，你可能會把整個 textViews 一一取出 `toString()` 然後抄一份：

```
List<String> strings = new ArrayList<>();

for (TextView textView : textViews) {
   strings.add(textView.getText().toString());
}
```

萬一 textViews 有一萬筆，最終你其實在存取 strings 通常不會全部都用到，這樣就太浪費了。所以我們拿出牛仔精神 `Cow - Copy-On-Write` (Lazy/CallByNeed)，先寫好轉換程式，當拿到那筆再去轉換，當然缺點是 textViews 要一直拿著，要稍微留意一下。我們先想像一下，寫一個名稱叫做 MapList 的類別，先把 textViews 拿著，在取出的時候 (@Override public E get(int index))，再去跑轉換程式。這裡我們開放一個 `map(Mappable)` 好把轉換程式交給我們 (Mappable)。

```
MapList<T, R> extends ArrayList<R> { // @Unmodifitable
    List<T> list;
    Mappable<T, R> mapper;

    public MapList(List<T> list) {
        super();
        this.list = list;
    }

    @Override public R get(int i) {
        return mapper.map(list.get(i));
    }

    public MapList<T, R> map(Mappable<T, R> mapper) {
        this.mapper = mapper;
        return this;
    }

    public interface Mappable<T, R> {
        R map(T t);
    }
}
```

```
List<String> strings = new MapList<TextView, String>(textViews).map
```

這是我們自己寫一個 MapList 類別來達成，而現在有了 RxJava：

```
List<String> strings = new IteratorOnlyList(Observable.from(textVie
    .map(textView -> textView.getText().toString())
    .toBlocking()
    .getIterator());
```

如果你想維持 List 界面，為了維持 lazy，又 RxJava 這邊只有提供到 Iterator，所以我們沒有使用 `toList().toBlocking().single()`，你可以寫一個 IteratorOnlyList 把這個 iterator 包起來，方便傳遞，雖然很多操作都殘缺。

盡可能還是改用 Observable 作為界面吧。

去除重複資料 distinct()

列出有發文的使用者：

Before:

```
List<User> getPostedUsers(List<Post> posts) {
    Map<String, User> users = new HashMap<>();
    for (Post post : posts) {
        User user = post.getUser();
        users.put(user.getObjectId(), user);
    }
    return new ArrayList<>(users.values());
}
```

After:

```
Observable<User> getPostedUsers(List<Post> posts) {
    return Observable.from(posts).map(post -> post.getUser())
        .distinct(user -> user.getObjectId());
}
```

多方合併 merge(), concatWith()

列出貼文以及留言的使用者：

Before:

```
List<User> getActivityUsers(Collection<Post> posts, Collection<Comm
    Map<String, User> users = new HashMap<>();
    for (Post post : posts) {
        User user = post.getUser();
        users.put(user.getObjectId(), user);
    }
    for (Comment comment : comments) {
        User user = comment.getUser();
        users.put(user.getObjectId(), user);
    }
    return new ArrayList<>(users.values());
}
```

After:

```
Observable<User> getActivityUsers(Observable<Post> posts, Observabl
    return Observable.merge(posts.map(post -> post.getUser()),
        comments.map(comment -> comment.getUser()))
        .distinct(user -> user.getObjectId());
}
```

`merge()` 與 `concatWith()` 最大的差異是， `merge()` 是併發同時進貨，所以會交錯，而 `concatWith` 則是排隊等到前面進貨完才換下一位。

優先列出貼文的使用者後，才列出留言的使用者：

```
Observable<User> getActivityUsers(Observable<Post> posts, Observabl
    return posts.map(post -> post.getUser())
        .concatWith(comments.map(comment -> comment.getUser()))
        .distinct(user -> user.getObjectId());
}
```

如何使用

在我們看過一些對照組之後，大致上瞭解未來在使用上會呈現什麼樣貌。

我們開始回頭學學 What/Why/How，什麼是 RxJava 、為什麼要 RxJava 、如何開始使用 RxJava 。

首先你要認識基本操作元件：Observable， 如果你知道 AsyncTask 或者 FutureTask 了話，基本上差不多。先看這個例子：

```
// AsyncTask 版本
AsyncTask<Void, Void, String> helloAsync = new AsyncTask<>() {
    @Override public String doInBackground(Void... voids) {
        return "Hello, world!";
    }
}
```

```
// FutureTask 版本
FutureTask<String> helloFuture =
    new FutureTask<>(new Callable<String>() {
        @Override public String call() {
            return "Hello, world!";
        }
    });
```

```
// Observable 版本
Observable<String> helloObs = Observable.create(
    new Observable.OnSubscribe<String>() {
        @Override
        public void call(Subscriber<? super String> sub) {
            sub.onNext("Hello, world!");
            sub.onCompleted(); // 因為 Observable 支援複數的關係，所以
        }
    }
);

// Observable lambda 版本：
Observable<String> helloObs = Observable.create(sub -> {
        sub.onNext("Hello, world!");
        sub.onCompleted();
    });
```

這些 AsyncTask, FutureTask, Observable 都是生產者，定義出資料的產生，接下來，當產品生出來的時候通知你。所以我們補上 listener， RxJava 稱之為 Subscriber：

```
Subscriber<String> helloSubscriber = new Subscriber<>() {
    @Override public void onNext(String string) { System.out.printl
    @Override public void onCompleted() { }
    @Override public void onError(Throwable e) { }
};

helloObs.subscribe(helloSubscriber); // 你可以下訂(subscribe()) ，產品
```

通常我們會用 `subscribe(Action1<? super T> onNext, Action1<Throwable> onError), Action1<? super T> onCompleted)` 來搭配 lambda， 寫起來會比較簡便：

```
helloObs.subscribe(string -> System.out.println(string));
helloObs.subscribe(string -> System.out.println(string), e -> e.pri
helloObs.subscribe(string -> System.out.println(string), e -> e.pri
```

我們再稍微回到 Observable.create() , 如果你的原料早就準備好了， 我們可以寫成：

```
Observable.just("Hello, world!").subscribe(string -> System.out.pri
```

接下來是簡單的加工：

```
Observable.just("Hello, world!")
    .map(string -> "andrew: " + string) // "andrew: Hello, world!"
    .map(string -> string.length())
    .subscribe(length -> System.out.println(length)); // 21, 請不要
```

所以從原物料的進貨作業(create(), just())到加工(map())到下訂(subscribe()) 有了流程上的基本認識。我們可以討論幾個站點的作用與概念:

首先，生產過程(進貨與加工)會被認定不可預期的處理時間，很有可能會很久的意思。

另一個是，JIT(Just In Time)，零庫存概念，也就是說你可以定義很多道加工手續，但是再沒有下訂之前，通通都不會開跑的，包括進貨作業。

接下來，換一個比較實際的例子：

```
Observable.just("http://yongjhih.gitbooks.io/feed/content/RxJava.ht
    .map(url -> download(url))
    .subscribeOn(Schedulers.io()) // 把生產加工過程丟到背景去做
    .subscribe(file -> System.out.println(file));
```

如果你有很多網址要下載，你可能會想到：

```
Observable<List<String>> urlList = Observable.just(Arrays.asList("h
```

你可以留意到這時 Observable 產線上是 `List<String>` 了。所以你可能會這樣做：

```
    urlList.map(list -> {
        List<File> files = new ArrayList<>();
        for (String url : urls) files.add(download(url));
        return files;
    })
    .subscribeOn(Schedulers.io())
    .subscribe(files -> {
        for (File file : files) System.out.println(file);
    });
```

但是這種情況，你應該使用 Observable.from() ：

```
Observable<String> urls = Observable.from(Arrays.asList("http://yor
    "http://yongjhih.gitbooks.io/feed/content/README.html"));
```

```
urls.map(url -> download(url))
    .subscribeOn(Schedulers.io())
    .subscribe(file -> System.out.println(file));
```

或者

```
Observable.just("http://yongjhih.gitbooks.io/feed/content/RxJava.ht
    "http://yongjhih.gitbooks.io/feed/content/README.html"));
```

如果原料是 List，但是加工時，想要一個一個處理，用 Observable.from() 會比較好操作，如果你用 Observable.just() 那就會拿到一個 List 。其實有個方法可以展開： `.flatMap(list -> Observable.from(list))` ：

```
Observable.just(Arrays.asList("http://yongjhih.gitbooks.io/feed/cor
    "http://yongjhih.gitbooks.io/feed/content/README.html"))
    .map(urls -> {
        List<File> files = new ArrayList<>();
        for (String url : urls) files.add(download(url));
        return files;
    })
    .flatMap(files -> Observable.from(files)) // 可以這裡才展開
    .subscribeOn(Schedulers.io())
    .subscribe(file -> System.out.println(file));
```

`map()` 是轉換產線上的物件， flatMap() 是轉換 `Observable<Object>` 的方法，舉個例子：

```
Observable.just("http://yongjhih.gitbooks.io/feed/content/RxJava.ht
    .flatMap(url -> downloadObs(url))
    .map(file -> file.size())
    .susbscirbe(size -> Systen.out.println(size));
```

```
Observable<File> downloadObs(String url) { ... }
```

產生器與流量控制

原料無中生有。

印出 1 到 10:

Before:

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

After:

```
Observable.range(1, 10).forEach(i -> System.out.println(i));
```

這邊第一次使用 `forEach()` ，僅差 `subscribe()` 會回傳 `Subscription` ， `Subscription` 是用來停止生產的。

秒針：

```
// 計秒器
Subscription subscription = Observable.interval(1, TimeUnit.SECONDS

Observable.just(sub).delay(10, TimeUnit.SECONDS).subscribe(s -> s.u
```

片語 toBlocking().single()

這行會卡住，直到拿到一個為止。

single(), take(1), first()

single() 與 take(1) 最大的不同是，single() 只能使用在單數的 Observable 上，否則會噴 exception 。

例如：

```
List<Integer> integers = Observable.range(1, 2).toList().single();
```

```
Integer i = Observable.range(1, 2).toBlocking().take(1); // pass
```

```
Observable<Integer> singleInteger = Observable.range(1, 1).single()
```

```
Observable<Integer> singleInteger = Observable.range(1, 2).single()
```

名詞解釋

Observable 觀測所 Subscriber 訂閱者 Observer 觀察員

描述這是怎樣的工作，以及中間需要的製程，希望產生出什麼產品。

Observable 一份工作 task 一個未來 future , T 產品. 相當於 AsyncTask, Future

Subscriber/Observer onEvent, Listener. 提貨券.

subscribe 下訂。

Subscription 訂單, subscribe 下訂之後產生出來的訂單，這個訂單可以用來取消訂單來中止生產。

Eager vs. Lazy

動手玩

```
git clone https://github.com/yongjhih/RxJava-retrofit-github-sample
cd RxJava-retrofit-github-sample
./gradlew execute
```

修改 src/main/java/com/github/yongjhih/Main.java 內容就可以自己玩了。

組合資料 **zip()**

```
Observable<User> getUser(Activity activity) {
    Observable.zip(getFbUser(activity), getParseUser(activity),
        (fbUser, parseUser) -> getUser(fbUser, parseUser));
}
```

Subject

相當於 Event Bus，多方進貨，多方出貨。開放式，俗稱 Hot ，對應之前封閉式的 Observable 為 Cold 。

多方進貨，相當於把 Observable.OnSubscriber ， 讓其他人可以塞資料。

Before:

```
Observable.just("hello, world!").subscribe(System.out::println);
```

After:

```
Subject<String, String> subject = PublishSubject.create();
subject.asObservable().subscribe(System.out::println);

subject.onNext("hello, world!");
```

找一個實際點的例子：

```
Subject<View> mLikeCountSubject = PublishSubject.create();

@OnClick(R.id.like_button)
public void onLikeClick(View view) {
    mLikeCountSubject.onNext(view);
}

@Override
public void onResume() {
    super.onResume();

    mLikeCountSubject.asObservable().map(view -> 1)
        .scan((count, i) -> count + i)
        .subscribe(count -> likeText.setText(count.toString()));
}
```

http://reactivex.io/documentation/subject.html

Exception 處理

如果發生 exception 重試，會重新進貨，從頭再跑一輪。

最常用的是 `retry(Func2<Integer, Throwable, Boolean> predicate)`

如果是 NullPointerException 才重試:

```
retry((c, e)) -> e instanceof NullPointerException);
```

一直重試：

```
retry()
```

重試 3 次：

```
retry(3)
```

使用 Handler `retryWhen(final Func1<? super Observable<? extends Throwable>, ? extends Observable<?>> notificationHandler)`，例如，隨著重試次數延後重試時間:

```
Observable.create((Subscriber<? super String> s) -> {
    s.onError(new RuntimeException("always fails"));
}).retryWhen(attempt -> {
    return attempt.zipWith(Observable.range(1, 3), (n, i) -> i).fla
        System.out.println("delay retry by " + i + " second(s)");
        return Observable.timer(i, TimeUnit.SECONDS);
    });
}).toBlocking().subscribe(System.out::println);
```

忽略 exception：

```
.onErrorResumeNext(e -> Observable.empty()); // onCompleted()
```

或傳個替代資料：

```
.flatMap(parseUser -> {
    return Observable.zip(createChannel(parseUser), createFriendRol
        .onErrorResumeNext(e -> Observable.just(parseUser));
})
```

flatMap()

p.s. 似乎很多讀者對於 `flatMap()` 有理解的困難，所以這裡特別解釋一下 `flatMap`

...

concatMap() 與 flatMap()

```
Observable<File> downloadObs(String url) { ... }
```

```
Observable.just("https://raw.githubusercontent.com/yongjhih/androic
    .flatMap(s -> downloadObs(s).subscribeOn(Schedulers.io()))
    .subscribe(f -> System.out.println("downloaded: " + f));
```

這裡會同時開兩個 threads 在下載。避免這種情況，希望一個一個跑：

```
Observable.just("https://raw.githubusercontent.com/yongjhih/androic
    .concatMap(s -> downloadObs(s).subscribeOn(Schedulers.io()))
    .subscribe(f -> System.out.println("downloaded: " + f));
```

排序 toSortedList()

分組 groupBy()

分段 **window()**

快取 **cache()**

debounce()

逾時 **timeout()**

利用 **compose(Transformer)** 重用常用的流程組合

Compose/Transformer 0.20 版本(2014.8)

先丟背景等等回來前景:

Before:

```
.observeOn(AndroidScheduler.mainThread())
.subscribeOn(Schedulers.io))
```

After:

```
.compose(mainAsync())
```

```
<T> Transformer<T, T> mainAsync() {
    return obs -> obs.subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread());
}
```

計數器:

Before:

```
.map(o -> 1)
.scan((count, i) -> count + i)
```

After:

```
.compose(amount())
```

```
<T> Transformer<T, T> amount() {
    return obs -> obs.map(o -> 1)
        .scan((count, i) -> count + i);
}
```

```
public class Transformers {
    @SuppressWarnings("unchecked")
    private static final Transformer MAIN_ASYNC = obs -> obs.subscr
        .observeOn(AndroidSchedulers.mainThread());

    @SuppressWarnings("unchecked")
    public static final <T> Transformer<T, T> mainAsync() {
        return MAIN_ASYNC;
    }

    @SuppressWarnings("unchecked")
    private static final Transformer AMOUNT = obs -> obs.map(o -> 1
        .scan((count, i) -> count + i);

    @SuppressWarnings("unchecked")
    public static final <T> Transformer<T, T> amount() {
        return AMOUNT;
    }
}
```

附錄：Android View Observable 範例

寫一個讚計數器:

```
ViewObservable.clicks(findViewById(R.id.like_button))
    .map(clickEvent -> 1)
    .scan((count, i) -> count + i)
    .subscribe(likes -> {
        TextView likesView = (TextView) findViewById(R.id.likes_vie
        textView.setText(likes.toString());
    });
```

透過 Observable 實現 Java8 Optional

在沒有 Optional 的狀況下：

Before:

```
String displayName(ParseUser parseUser) {
    String displayName = parseUser.getString("displayName");
    if (displayName == null) {
        displayName = "Unnamed";
    }
    return displayName;
}
```

After:

```
String displayName(ParseUser parseUser) {
    return Optional.ofNullable(parseUser.getString("displayName"))
}
```

```
// Optional 實現:

public class Optional<T> {
    Observable<T> obs;

    public Optional(Observable<T> obs) {
        this.obs = obs;
    }

    public static <T> Optional<T> of(T value) {
        if (value == null) {
            throw new NullPointerException();
        } else {
            return new Optional<T>(Observable.just(value));
        }
    }

    public static <T> Optional<T> ofNullable(T value) {
        if (value == null) {
            return new Optional<T>(Observable.empty());
        } else {
            return new Optional<T>(Observable.just(value));
        }
    }

    public T get() {
        return obs.toBlocking().single();
    }

    public T orElse(T defaultValue) {
        return obs.defaultIfEmpty(defaultValue).toBlocking().single
    }
}
```

Oracle 官方比較複雜的 Optional 範例，當各類別傳遞已作成 Optional 界面：

Before:

```
String getVersion(Computer computer) {
  String version = "UNKNOWN";
  if (computer != null) {
    Soundcard soundcard = computer.getSoundcard();
    if (soundcard != null) {
      USB usb = soundcard.getUsb();
      if (usb != null) {
        version = usb.getVersion();
      }
    }
  }
  return version;
}
```

After:

```
// Optional 版本
String getVersion(Computer computer) {
  return computer.flatMap(Computer::soundcard) // soundcard()
    .flatMap(Soundcard::usb) // usb()
    .map(Usb::getVersion)
    .orElse("UNKNOWN");
}

public class Computer {
  private Soundcard soundcard;
  public Optional<Soundcard> soundcard() { return Optional.ofNullab
  public Soundcard getSoundcard() { return soundcard; }
}

public class Soundcard {
  private Usb usb;
  public Optional<Usb> usb() { return Optional.ofNullable(usb); }
  public Usb getUsb() { return usb; }
}

public class Usb {
  String version;
  public String getVersion() { return version; }
}
```

P.S. Groovy 語言: `String version = computer?.getSoundcard()?.getUSB()?.getVersion();`

如果真的要用 RxJava Optional 請用專門的專案
https://github.com/eccyan/RxJava-Optional

這邊僅是教學需要。 (https://gist.github.com/yongjhih/25017ac41efb4634c2ab)

Threading, Scheduler 執行序控制

```
Observable.just("http://yongjhih.gitbooks.io/feed/content/RxJava.ht
.subscribeOn(Schedulers.io()) // 在 io Thread() 跑 (60 thread pool,
.observeOn(AndroidScheduler.mainThread()) // 在 mainThread 回來印
.subscirbe(System.out::println);
```

改善 Cache 流程

```
Observable.merge(ParseObservable.getPostsFromLocalDatabase(), Parse
```

或者用 `amb()`, `first()` 等方式作

Notification

Backpressure

Assertions

Using assertj-rx:

```
assertThat(observable.toBlocking())
    .completes()
    .emitsSingleValue("hello");
```

後記

這裡鮮少討論一般常見的理論：無論是 FRP, monad, push/pull, cold/hot 。主要因為先以範例，也就是實際看得到的改善是什麼來作介紹。

http://yarikx.github.io/NotRxJava/ 也是從這種方式帶領你看發展進程，希望大家會喜歡

See Also

- 章節：輕量資料流處理

文獻:

- https://speakerdeck.com/jakewharton/2014-1?slide=13
- http://slides.com/yaroslavheriatovych/frponandroid
- http://en.wikipedia.org/wiki/Null_coalescing_operator
- http://www.oracle.com/technetwork/articles/java/java8-optional-2175753.html
- http://java.dzone.com/articles/java-8-elvis-operator
- http://blog.danlew.net/2015/06/22/loading-data-from-multiple-sources-with-rxjava/
- https://blog.growth.supply/party-tricks-with-rxjava-rxandroid-retrolambda-1b06ed7cd29c

一些 Rx 開源：

- https://github.com/yongjhih/RxParse
- https://github.com/ogaclejapan/RxBinding
- https://github.com/eccyan/RxJava-Optional
- https://github.com/square/sqlbrite
- https://github.com/pushtorefresh/storio
- https://github.com/mcharmas/Android-ReactiveLocation
- https://github.com/pardom/Ollie
- https://github.com/frankiesardo/ReactiveContent
- https://github.com/evant/rxloader
- https://github.com/vyshane/rex-weather
- https://github.com/ReactiveX/RxNetty
- https://github.com/vbauer/caesar
- https://gist.github.com/ivacf/874dcb476bfc97f4d555
- https://github.com/ribot/assertj-rx

- 小抄：https://gist.github.com/yongjhih/bbe3b528873c7eb671c6

[1] `rx.android.observables.AndroidObservable` has changed to `rx.android.app.AppObservable` (Version 0.24 – January 3rd 2015)

RxAndroid

https://github.com/ReactiveX/RxAndroid

生命週期的連動。

AppObservable

```
AppObservable.bindActivity()
AppObservable.bindFragment()
```

主要檢查 `Fragment.isAdded()`, `Activity.isFinishing()` 。

註：筆者不是很清楚，為什麼不用 *overloading:*
`AppObservable.bind(Activity/Frgment/v4.Fragment)` 來取代
`AppObservable.bindFragment(Fragment)`,
`AppObservable.bindFragment(v4.Fragment)`,
`AppObservable.bindActivity(Activity)`

LifecycleObservable

```
LifecycleObservable.bindActivityLifecycle()
LifecycleObservable.bindFragmentLifecycle()
```

當 Activty/Fragment 對應的生命週期結束時，自動 `unsubscribe()` 。

`LifecycleObservable` 哪時候訂閱哪時候取消對照表：

```
CREATE -> LifecycleEvent.DESTROY;
START -> LifecycleEvent.STOP;
RESUME -> LifecycleEvent.PAUSE;
PAUSE -> LifecycleEvent.STOP;
STOP -> LifecycleEvent.DESTROY;
```

手動自己 `unsubscribe()`， 如果 Activity 要結束，把一些 subscriptions 取消：

```
class SimpleActivity extends Activity {
    CompsotionSubscription mSubscriptions = new CompositeSubscripti

    @Override
    protected void onResume() {
        super.onResume();

        bind(Observable.just("Hello, world"), s -> textView.setText
    }

    protected <T> void bind(Observable<T> obs, Action1<T> onNext) {
        mCompositeSubscription.add(AppObservable.bindActivity(this,
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        mCompositeSubscription.unsubscribe();
    }
}
```

`LifecycleObservable` + `RxActivity`:

```
class SimpleActivity extends RxActivity {

    @Override
    public void onResume() {
        LifecycleObservable.bindActivityLifecycle(lifecycle(),
            AppObservable.bindActivity(this, Observable.just("Hello
        ).subscribe(s -> textView.setText(s));
    }
}
```

ViewObservable, WidgetObservable

View 的連動. 當 View 顯示時 `subscribe()` 離開時 `unsubscribe()`

```
ViewObservable.bindView()
```

Event 的連動.

```
ViewObservable.clicks()
```

ogaclejapan/RxBinding

https://github.com/ogaclejapan/RxBinding

類似於 ViewObservable。主要以 MVVM 雙向連動作努力。

Before：

```
class HogeActivity extends Activity {

    @InjectView(R.id.text)
    private TextView mTextView;

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

        AppObservable.bindActivity(this, Observable.just("hoge"))
            .subscribeOn(Schedulers.io())
            .subscribe(setTextAction());
    }

    Action1<String> setTextAction() {
        return text -> mTextView.setText(text);
    }
}
```

After:

```
class HogeActivity extends Activity {
    // ...

    private Rx<TextView> mRxTextView

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

        mRxTextView = RxView.of(mTextView);
        Subscription s = mRxTextView.bind(Observable.just("hoge"),
    }
}
```

RxLifecycle

See Also

- RxActivity
- RxFragmentActivity
- RxFragment
- ogaclejapan/RxBinding
- JakeWharton/RxBinding
- https://github.com/trello/RxLifecycle
- https://github.com/ReactiveX/RxAndroid/issues/172

註：並沒有 *RxAppCompatActivity*

RxBinding(JakeWharton/RxBinding)

依據 Android 各項元件出發，重新披上 RxJava 的皮。

RxAndroid Before:

```
ViewObservable.clicks(textView).subscribe(ev -> {});
```

RxBinding After:

```
RxTextView.textChanged(textView).subscribe(ev -> {});
```

基本上與 ViewObservable, ogaclejapan/RxBinding 都是處理 View 相關的連動。

See Also

- Jack Wharton 的 RxJava 電台訪談：
  http://fragmentedpodcast.com/episodes/7/
- https://github.com/JakeWharton/RxBinding
- https://github.com/ogaclejapan/RxBinding

資料流替代方案

RxJava 已經介紹了資料流的概念。如果你不需要 RxJava 提供的執行緒管理、錯誤處理等又覺得 RxJava 不夠輕量。或許你可以採用輕量，只針對資料流的函式庫。

- https://github.com/kgmyshin/Marray
- https://github.com/konmik/solid

特點：

- Marray 可修改。
- SolidList 主要以 ImuttableList 出發，所以沒實現修改能力。
- SolidList 支援 Parcelable 傳遞。

另外還有其他 Promise 選用:

- Bolt-Android

另一個知名 https://github.com/goldmansachs/gs-collections 不過似乎不夠輕。

http://www.goldmansachs.com/gs-collections/documents/GS%20Collections%20Training%20Session%20and%20Kata%205.0.0.pdf

Lambda λ

一種簡化寫 callback 的表達式。

Before:

```
view.setOnClickListener(new View.OnClickListener() {
  @Override public void onClick(View v) {
    println("yo");
  }
});
```

After:

```
view.setOnClickListener(v -> System.out.println(v));
```

or

```
view.setOnClickListener(System.out::println);
```

- 當 interface 只有一個 method 需要實作時，就不需要特別再說是哪個 method name 了。包含 interface name 就可以整個省略。只剩下參數名稱要寫而已，如果怕參數型別有混淆之虞可寫上型別：

```
view.setOnClickListener((View v) -> println(v));
```

- 如果沒有參數：

Before:

```
Runnable = new Runnable() {
  @Override public void run() {
    println("yo");
  }
};
```

After:

```
Runnable = () -> println("yo");
```

如果你要在 Android 上使用，請參考 gradle-retrolambda 。

進階：如果 abstract class 只有一個 abstract method 是否也能適用 lambda？

筆者測試過，目前是不行的。

這裡筆者有想到一招替代方案，寫個可以吃 lambda 的 abstract class creator 來幫忙生，例如：

Before:

```
void save(ParseUser user) {
    // SaveCallback 是 abstract class 且只有一個 abstract method: "vo
    user.save(new SaveCallback() {
        @Override public void done(ParseException e) {
            e.printStackTrace();
        }
    });
}
```

After:

```
void save(ParseUser user) {
    user.save(Callbacks.save(e -> e.printStackTrace()));
}
```

寫一個可以吃 lambda 的 Callbacks.save() 來幫忙生 abstract SaveCallback：

```
public class Callbacks {
    public interface ISaveCallback {
        void done(ParseException e);
    }

    public static SaveCallback save(ISaveCallback callback) {
        return new SaveCallback() {
            @Override public void done(ParseException e) {
                callback.done(e);
            }
        };
    }
}
```

Bolts-Android

如果你已經在用 RxJava，那這應該僅供參考。

Bolts 是一款 promise 的實現。由 Parse.com 主持。Facebook 在收購 Parse.com 後，也積極整合 bolts 。

`Bolts.Task<T>` 相當於 `Observable<T>`

`Bolts.Task.continueWith()` 相當於 `Observable.map()`

```
Task<ParseObject> saveAsync;
...
// .map(o -> null);
saveAsync(obj).continueWith(new Continuation<ParseObject, Void>() {
  public Void then(Task<ParseObject> task) throws Exception {
    if (task.isCancelled()) {
      // 取消
    } else if (task.isFaulted()) {
      // 失敗
      Exception error = task.getError();
    } else {
      // 成功
      ParseObject object = task.getResult();
    }
    return null;
  }
});
```

`Bolts.Task.continueWithTask()` 相當於 `Observable.flatMap()`

```
// .flatMap(o -> saveObs(o)).map(o -> null);
query.findInBackground().continueWithTask(new Continuation<List<Par
  public Task<ParseObject> then(Task<List<ParseObject>> task) throw
    if (task.isFaulted()) {
      return null;
    }

    List<ParseObject> students = task.getResult();
    students.get(1).put("salutatorian", true);
    return saveAsync(students.get(1));
  }
}).onSuccess(new Continuation<ParseObject, Void>() {
  public Void then(Task<ParseObject> task) throws Exception {
    return null;
  }
});
```

`Task.forResult()` 相當於 `Observable.just()`

```
Task<String> successful = Task.forResult("The good result.");
```

`Task.create()` 相當於 `Observable.create()`

```
public Task<ParseObject> fetchAsync(ParseObject obj) {
  final Task<ParseObject>.TaskCompletionSource tcs = Task.create();
  obj.fetchInBackground(new GetCallback() {
    public void done(ParseObject object, ParseException e) {
     if (e == null) {
       tcs.setResult(object);
     } else {
       tcs.setError(e);
     }
   }
  });
  return tcs.getTask();
}
```

`Task.callInBackground()` 相當於 `Observable.defer()`:

```
 Task<Void> Task.callInBackground(new Callable<Void>() {
  public Void call() {
    // Do a bunch of stuff.
  }
});
```

`Task.waitForCompletion()` 相當於 `Observable.toBlocking()`:

Tasks.java:

```
public static <T> T wait(Task<T> task) {
  try {
    task.waitForCompletion();
    if (task.isFaulted()) {
      Exception error = task.getError();
      if (error instanceof RuntimeException) {
        throw (RuntimeException) error;
      }
      throw new RuntimeException(error);
    } else if (task.isCancelled()) {
      throw new RuntimeException(new CancellationException());
    }
    return task.getResult();
  } catch (InterruptedException e) {
    throw new RuntimeException(e);
  }
}
```

RxBolts

ref. yongjhih/RxBolts/.../TaskObservable.java

```
    public static <R> Observable<R> just(Task<R> task) {
        return Observable.create(sub -> {
            task.continueWith(t -> {
                if (t.isCancelled()) {
                    sub.unsubscribe(); //sub.onCompleted();?
                } else if (t.isFaulted()) {
                    sub.onError(t.getError());
                } else {
                    R r = t.getResult();
                    if (r != null) sub.onNext(r);
                    sub.onCompleted();
                }
                return null;
            });
        });
    }
```

See Also

- https://github.com/BoltsFramework/Bolts-Android

AutoValue

一個透過 Annotation Programming 簡化撰寫 identity model 的編譯時期函式庫。

Before:

```
public class User {
  public String name;
  public int id;

  public String name() {
    return name;
  }

  public int id() {
    return id;
  }

  public User(String name, int id) {
    this.name = name;
    this.id = id;
  }

  @Override
  public String toString() {
    return "User{"
      + "name=" + name
      + ", id=" + id
      + "}";
  }

  @Override
  public boolean equals(Object o) {
    if (o == this) {
      return true;
    }
    if (o instanceof User) {
      User that = (User) o;
```

```
      return (this.name.equals(that.name()))
        && (this.id == that.id());
    }
    return false;
  }

  @Override int hashCode() {
    return Objects.hashCode(name, id);
  }
}
```

```
User andrew = new User("Andrew", 1);
User andrew1 = new User("Andrew", 1);
User andrew2 = new User("Andrew", 2);
System.out.println(andrew.equals(andrew1));
System.out.println(andrew.equals(andrew2));
```

After:

```
import com.google.auto.value.AutoValue;

@AutoValue
public abstract class User {
    public abstract String name();
    public abstract int id();

    public static User builder() {
        return new AutoValue_User.Builder();
    }

    @AutoValue.Builder
    interface Builder {
        Builder name(String s);
        Builder id(int n);
        User build();
    }
}
```

```
User andrew = User.builder().name("Andrew").id(1).build();
User andrew1 = User.builder().name("Andrew").id(1).build();
User andrew2 = User.builder().name("Andrew").id(2).build();
System.out.println(andrew.equals(andrew1));
System.out.println(andrew.equals(andrew2));
```

Android Parcelable

frankiesardo/auto-parcel:

```
@AutoParcel
abstract class SomeModel implements Parcelable {
    abstract String name();
    abstract List<SomeSubModel> subModels();
    abstract Map<String, OtherSubModel> modelsMap();

    public static Builder builder() {
        return new AutoParcel_SomeModel.Builder();
    }

    @AutoParcel.Builder
    public interface Builder {
        public Builder name(String x);
        public Builder subModels(List<SomeSubModel> x);
        public Builder modelsMap(Map<String, OtherSubModel> x);
        public SomeModel build();
    }
}
```

或者 https://github.com/johncarl81/parceler

```
@AutoValue
@Parcel
public abstract class AutoValueParcel {

    @ParcelProperty("value") public abstract String value();

    @ParcelFactory
    public static AutoValueParcel create(String value) {
        return new AutoValue_AutoValueParcel(value);
    }
}
```

parceler 的產生器, 有點獨特..

See Also

- https://github.com/vbauer/jackdaw
- https://github.com/frankiesardo/auto-parcel
- https://github.com/johncarl81/parceler
- https://github.com/rharter/auto-value-parcel

Dagger2

DI 工具

什麼是 DI

DI, Dependency Injection (相依性注入)， Anti-DIY, Auto-DIY 自動組裝，行前準備/著裝/組件/要件

例如 ButterKnife 就是一種 DI，只是針對 findView 來做的 Injection 。

在特定的生命週期，幫你生成所需的物件。Dagger 是讓你創造自己的 Injection。

另外一個重點是，重用元件。

著名的咖啡機

沖泡出一杯風味十足的咖啡之前，你需要一台濾泡式咖啡機。

需要的元件有：

- 加熱器：把水加熱
- 幫浦

在沒有的 DI 概念下：

Before:

```
CoffeMaker maker = new CoffeeMaker();
maker.brew(); // 沖泡
maker.brew(); // 沖泡
```

```
class CoffeeMaker { // 咖啡機
    private final Heater heater; // 加熱器
    private final Pump pump; // 幫浦

    CoffeeMaker() {
        this.heater = new ElectricHeater(); // 電熱器
        this.pump = new Thermosiphon(heater); // 虹吸幫浦(虹吸，所以也
    }

    public void brew() { /* ... */ }
}
```

缺點是每杯咖啡生產的整台咖啡機、電熱器與幫浦，這些組件都無法延用到其他裝置身上，太浪費了，一點都不環保。

手動 DI:

After:

```
Heater heater = new ElectricHeater();
Pump pump = new Thermosiphon(heater);
CoffeeMaker maker = new CoffeeMaker(heater, pump);
CoffeeMaker maker2 = new CoffeeMaker(heater, pump);
maker.brew();
maker2.brew();
```

```
class CoffeeMaker {
    private final Heater heater;
    private final Pump pump;

    CoffeeMaker(Heater heater, Pump pump) {
        this.heater = heater;
        this.pump = pump;
    }

    public void brew() { /* ... */ }
}
```

這樣至少電熱器與幫浦都可以有機會重覆拿給別人使用了。

但是這樣我們要泡咖啡前，都要自己準備電熱器與幫浦然後組裝成咖啡機後才開始泡咖啡。

我們希望寫好咖啡機所需要的組件，請一個組裝工人幫我們組，以後只要說我現在要用咖啡機就馬上組裝好了，我們都不用自己 DIY。其他裝置也是由這個工人負責組裝，這個工人可以沿用相同零組件來生產。

利用 Dagger2 自動 DI 來組裝那些要件，只要描述好要件相依後，就可以一直泡一直泡：

```
Coffee coffee = Dagger_CoffeeApp_Coffee.create();
Coffee coffee2 = Dagger_CoffeeApp_Coffee.create();
coffee.maker().brew(); // 一直泡
coffee2.maker().brew(); // 一直泡
```

咖啡機要加熱加壓沖泡，相依要件關係圖：

```
CoffeeMaker -> DripCoffeeModule ----------------------> Heater
                  \-> PumpModule -> ThermosiphonPump -/
```

濾泡式咖啡機需要把水加熱、加壓後沖泡：

```
class CoffeeMaker {
  private final Lazy<Heater> heater; // 加熱器
  private final Pump pump;

  // 準備幫浦與加熱器
  @Inject CoffeeMaker(Lazy<Heater> heater, Pump pump) {
    this.heater = heater;
    this.pump = pump;
  }

  // 沖泡
  public void brew() {
    heater.get().on(); // 加熱
    pump.pump(); // 加壓
    System.out.println(" [_]P coffee! [_]P "); // 熱騰騰的咖啡出爐囉！
    heater.get().off(); // 隨手關加熱器
  }
}
```

開始寫組裝說明書：

```
@Singleton // 共用咖啡機
@Component(modules = DripCoffeeModule.class) // 濾泡裝置(安裝著高壓熱水
public interface Coffee {
    CoffeeMaker maker();
}
```

濾泡裝置需要幫浦加壓器具：

```
@Module(includes = PumpModule.class) // 一同準備加壓器具
class DripCoffeeModule { // 濾泡裝置
  @Provides @Singleton Heater provideHeater() { // 提供共用的加熱器具
    return new ElectricHeater(); // 電熱器具
  }
}
```

幫浦：

```
@Module(complete = false, library = true) // complete = false 需要借
class PumpModule { // 幫浦加壓器具
  @Provides Pump providePump(Thermosiphon pump) { // 利用熱虹吸管來提
    return pump;
  }
}
```

熱虹吸管幫浦：

```
class Thermosiphon implements Pump {
  private final Heater heater;

  @Inject
  Thermosiphon(Heater heater) { // 索取加熱器來加壓
    this.heater = heater;
  }

  @Override public void pump() {
    if (heater.isHot()) {
      System.out.println("=> => pumping => =>");
    }
  }
}
```

動手玩

```
git clone https://github.com/yongjhih/dagger2-sample
cd dagger2-sample
./gradlew execute
```

範例

Before:

```
OkHttpClient client = new OkHttpClient();
TwitterApi api = new TwitterApi(client);

String user = "Andrew Chen";

Tweeter tweeter = new Tweeter(api, user);
tweeter.tweet("Hello, world!");

Timeline timeline = new Timeline(api, user);
for (Tweet tweet : timeline.get()) {
    System.out.println(tweet);
}

String user2 = "Andrew Chen2";

Tweeter tweeter2 = new Tweeter(api, user2);
tweeter.tweet("Hello, world!");

Timeline timeline2 = new Timeline(api, user2);
for (Tweet tweet : timeline.get()) {
    System.out.println(tweet);
}
```

隱藏相依前置作業，透過 builder 來獲得末端物件。

After:

```
ApiComponent apiComponent = Dagger_ApiComponent.create();

TwitterComponent twitterComponent = Dagger_TwitterComponent.builder
    .apiComponent(apiComponent)
    .twitterModule(new TwitterModule("Andrew Chen"))
    .build();

Tweeter tweeter = twitterComponent.tweeter();
Timeliner timeline = twitterComponent.timeline();

for (Tweet tweet : timeline.get()) {
    System.out.println(tweet);
}

TwitterComponent component2 = Dagger_TwitterComponent.builder()
    .apiComponent(apiComponent)
    .twitterModule(new TwitterModule("Andrew Chen2"))
    .build();

Tweeter tweeter2 = component2.tweeter();
Timeliner timeline2 = component2.timeline();

for (Tweet tweet : timeline.get()) {
    System.out.println(tweet);
}
```

連帶修改:

```
@Module
public class NetworkModule {
    @Provides @Singleton // @Singleto 註明沿用同一個, @Provides 註明可
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient();
    }
}
```

```
@Singleton
public class TwitterApi {
    private final OkHttpClient client;

    @Inject // 註明由相依提供 OkHttpClient
    public TwitterApi(OkHttpClient client) {
        this.client = client;
    }
}
```

```
@Singleton
@Component(modules = NetworkModule.class) // 由哪些 modules 組成
public interface ApiComponent {
    TwitterApi api();
}
```

```
@Module
public class TwitterModule {
    private final String user;

    public TwitterModule(String user) {
        this.user = user;
    }

    @Provides
    Tweeter provideTweeter(TwitterApi api) {
        return new Tweeter(api, user);
    }

    @Provides
    Timeline provideTimeline(TweeterApi api) {
        return new Timeline(api, user);
    }
}
```

```
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
    Tweeter tweeter();
    Timeline timeline();
}
```

See Also

- https://speakerdeck.com/jakewharton/dependency-injection-with-dagger-2-devoxx-2014
- http://google.github.io/dagger/
- https://www.youtube.com/watch?v=oK_XtfXPkqw
- https://github.com/JorgeCastilloPrz/Dagger2Scopes
- http://fernandocejas.com/2015/04/11/tasting-dagger-2-on-android/

Annotation Programming 簡介

meta programming

- 編譯時期 Processor
- 執行時期 Reflection, InvocationHandler

這邊主要討論編譯時期 JSR 269 javax.annotation.processing.AbstractProcessor

從常見的 ORM annotations 、json2pojo(Gson, Jackson) 、Dagger1/2、Mockito、Retrofit、AutoValue、AutoParcel、ButterKnife、AndroidAnnotations、RoboGuice 等函式庫，大量利用 annotations 來精簡聚焦、解決煩冗的例行性撰寫程序。

像是 AutoValue 就節省了煩冗的 getter 、 setter 、builder 等例行撰寫程序。

除了解析 annotations 還包括解析 abstract methods, methods, interfaces, fields 等，緊接著是產生 java file 的 template language 以及物件導向的產生器。

解析 Annotation

```
@Example
public class ExampleClass implements Runnable {
    public final String name;
    public ExampleClass(String name) {
        this.name = name;
    }
    @Override
    public void run() {
        System.out.println(name);
    }
}
```

```
public ExampleProcessor extends AbstractProcessor {
    // ...
    @Override
    public boolean process(Set<? extends TypeElement> annotations,
        Set<? extends Element> elements = env.getElementAnnotatedWi
        for (Element e : elements) { // 可列出 @Example
            System.out.println(e);
        }
        return false;
    }
}
```

產生器的撰寫方法

依據解析結果產生 java file。

Template Language:

Apache Velocity Template Language, *.vm

For AVTL example:

```
#if (!$pkg.empty)
package $pkg;
#end

#foreach ($i in $imports)
import $i;
#end

@${generated}("com.google.auto.value.processor.AutoAnnotationProces
final class $className implements $annotationName {

## Fields

#foreach ($m in $members)
  #if ($params.containsKey($m.toString()))

  private final $m.type $m;

  #else

  private static final $m.type $m = $m.defaultValue;

  #end
#end
```

Square JavaPoet: https://github.com/square/javapoet

For JavaPoet example:

```
package com.example.helloworld;

public final class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, JavaPoet!");
  }
}
```

```
MethodSpec main = MethodSpec.methodBuilder("main")
    .addModifiers(Modifier.PUBLIC, Modifier.STATIC)
    .returns(void.class)
    .addParameter(String[].class, "args")
    .addStatement("$T.out.println($S)", System.class, "Hello, JavaP
    .build();

TypeSpec helloWorld = TypeSpec.classBuilder("HelloWorld")
    .addModifiers(Modifier.PUBLIC, Modifier.FINAL)
    .addMethod(main)
    .build();

JavaFile javaFile = JavaFile.builder("com.example.helloworld", hell
    .build();

javaFile.emit(System.out);
```

TypeMirror -> Annoation

```
Annotation annotation = ((DeclaredType) typeMirror).asElement().get
```

TODO

https://github.com/yongjhih/JavaPoetic:

```
JavaFile javaFile = JavaFile.package("com.example.helloworld").clas
    JavaClass.public().final().name("HelloWorld").method(
            JavaMethod.public().static().void().name("main").parame
                JavaStatement("$T.out.println($S)", System.class, '
                JavaStatement("$T.out.println($S)", System.class, '
            )
        )
    )
);
```

名詞解釋

APT, Annotation-Processing Tool

See also

- https://speakerdeck.com/jakewharton/annotation-processing-boilerplate-destruction-square-waterloo-2014
- https://github.com/8tory/simple-parse (runtime)
- https://github.com/8tory/auto-parse (source)
- http://velocity.apache.org/engine/devel/vtl-reference-guide.html
- https://github.com/yongjhih/JavaPoetic
- https://github.com/yongjhih/RetroFacebook
- http://blog.retep.org/2009/02/13/getting-class-values-from-annotations-in-an-annotationprocessor/

RecyclerView 簡介

基本使用方法:

Before:

```
@InjectView(R.id.icons)
RecyclerView icons;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    final List<String> list = new ArrayList<>(Arrays.asList("http:/

    RecyclerAdapter<IconViewHolder> listAdapter = new RecyclerAdapt
        @Override
        public IconViewHolder onCreateViewHolder(ViewGroup parent,
            return new IconViewHolder(LayoutInflater.from(context)
        }

        @Override
        public void onBindViewHolder(IconViewHolder viewHolder, int
            viewHolder.icon.setImageURI(Uri.parse(list.get(positior
        }
    }

    icons.setLayoutManager(new LinearLayoutManager(activity));
    icons.setAdapter(listAdapter);

    list.add("http://example.com/b.png");
    listAdapter.notifyDataSetChanged();
}
```

```
public class IconViewHolder extends BindViewHolder<String> {
    @InjectView(R.id.icon)
    public SimpleDraweeView icon;

    public IconViewHolder(View itemView) {
        super(itemView);
        ButterKnife.inject(this, itemView);
    }
}
```

有資料外漏的情形。

改用筆者簡單寫的 `ListRecyclerAdapter<T, VH>`
(https://github.com/yongjhih/recyclerlist)

https://gist.github.com/yongjhih/d8db8c69190293098eec

After:

```
@InjectView(R.id.icons)
public RecyclerView icons;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ListRecyclerAdapter<String, IconViewHolder> listAdapter = ListR
    listAdapter.getList().add("http://example.com/a.png");

    listAdapter.createViewHolder((parent, viewType) -> new IconView

    icons.setLayoutManager(new LinearLayoutManager(activity));
    icons.setAdapter(listAdapter);

    listAdapter.getList().add("http://example.com/b.png");
    listAdapter.notifyDataSetChanged(); // TODO hook List.add(), Li
}
```

IconViewHolder:

```
public class IconViewHolder extends BindViewHolder<String> {
    @InjectView(R.id.icon)
    public SimpleDraweeView icon;

    public IconViewHolder(View itemView) {
        super(itemView);
        ButterKnife.inject(this, itemView);
    }

    @Override
    public void onBind(int position, String item) {
        icon.setImageURI(Uri.parse(item));
    }
}
```

以 Avatar 頭像為例：

```
// 1. listAdapter.createViewHolder(): 決定使用哪種 layout + viewHolde
// 2. ViewModel.builder(): 以及如何把原始資料降階至流通性資料格式
// 不需要瞭解具備 View 相關設定知識，如圖片顯示現在是用 SimpleDraweeView.s
// 如果畫面要微調，直接複製 layout 調整風格。
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    //ListRecyclerAdapter<String, IconViewHolder> listAdapter = Lis
    //listAdapter.getList().add("http://example.com/a.png");
    ListRecyclerAdapter<AvatarViewModel, AvatarViewHolder> listAdap

    listAdapter.createViewHolder((parent, viewType) -> new AvatarVi

    listAdapter.getList().add(AvatarViewModel.builder().icon("http
    for (User user : getUsers()) { // 新增其他使用者
        listAdapter.getList().add(AvatarViewModel.builder().icon(us
    }
    // via RxJava
    // listAdapter.getList().addAll(Observable.from(getUsers()).map

    ...
}
```

```
// 只需具備 View 相關知識，設定流通性資料
// 不需要瞭解原始資料來源格式(不管是從 Facebook/Parse 還是 local sqlite 或
// 其中一個優點是當來源資料改變成其他格式，你依然可以沿用此 ViewHolder
public class AvatarViewHolder extends BindViewHolder<AvatarViewMode
    @InjectView(R.id.icon)
    public SimpleDraweeView icon;
    @InjectView(R.id.text1)
    public TextView name;
    @InjectView(R.id.updatedAt)
    public RelativeTimeTextView updatedAt;

    public AvatarViewHolder(View view) {
        super(view);
        ButterKnife.inject(this, view);
    }

    @Override
    public void onBind(int position, AvatarViewModel item) {
        //icon.setImageURI(Uri.parse(item));
        icon.setImageURI(Uri.parse(item.icon()));
        name.setText(item.name());
        updatedAt.setReferenceTime(item.updatedAt());
    }
}
```

```
// 提供流通性基本資料的介面
@AutoParcel
public abstract class AvatarViewModel implements Parcelable {
    public abstract String icon();
    public abstract String name();
    public abstract Long updatedAt();

    public static Builder builder() {
        return new AutoParcel_AvatarViewModel.Builder();
    }

    @AutoParcel.Builder
    public interface Builder {
        public Builder icon(String x);
        public Builder name(String x);
        public Builder updatedAt(Long x);
        public AvatarViewModel build();
    }
}
```

- http://stackoverflow.com/questions/26649406/nested-recycler-view-height-doesnt-wrap-its-content/28510031

Annotations

常用的 Annotations 介紹.

google guava, jsr, apache common, google common, android support

```
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
...

    /**
     * Add support for inflating the <fragment> tag.
     */
    @Nullable
    @Override
    public View onCreateView(String name, @NonNull Context context,
        ...
```

```
import android.support.annotation.StringRes;
...
    public abstract void setTitle(@StringRes int resId);
```

```
import android.support.annotation.IntDef;
...
public abstract class ActionBar {
    ...
    @IntDef({NAVIGATION_MODE_STANDARD, NAVIGATION_MODE_LIST, NAVIGA
    @Retention(RetentionPolicy.SOURCE)
    public @interface NavigationMode {}

    public static final int NAVIGATION_MODE_STANDARD = 0;
    public static final int NAVIGATION_MODE_LIST = 1;
    public static final int NAVIGATION_MODE_TABS = 2;

    @NavigationMode
    public abstract int getNavigationMode();

    public abstract void setNavigationMode(@NavigationMode int mode
```

```
    @IntDef(flag=true, value={
            DISPLAY_USE_LOGO,
            DISPLAY_SHOW_HOME,
            DISPLAY_HOME_AS_UP,
            DISPLAY_SHOW_TITLE,
            DISPLAY_SHOW_CUSTOM
    })
    @Retention(RetentionPolicy.SOURCE)
    public @interface DisplayOptions {}
```

@VisibleForTesting

```
@VisibleForTesting
public void setLogger(ILogger logger) {...}
```

@Keep

> @Keep
>
> We've also added @Keep to the support annotations. Note however that > that annotation hasn't been hooked up to the Gradle plugin yet (though > it's in progress.) When finished this will let you annotate methods and > > classes that should be retained when minimizing the app.

看起來還在進行中，可以先用筆者的專案：https://github.com/yongjhih/proguard-annotations

See Also

- ref. http://tools.android.com/tech-docs/support-annotations
- https://plus.google.com/+StephanLinzner/posts/GBdq6NsRy6S

Image Loader - Android-Universal-Image-Loader, picasso, glide, fresco

例如圖片網址:

```
http://upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/125px-Flag_of_the_Republic_of_China.svg.png
```

Before:

```
new AsyncTask<String, Void, Bitmap> {
    @Override
    protected Bitmap doInBackground(String... urls) {
        String url = urls[0];
        Bitmap bitmap = null;
        try {
            InputStream in = new java.net.URL(url).openStream();
            bitmap = BitmapFactory.decodeStream(in);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return bitmap;
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        if (result != null) mImageView.setImageBitmap(result);
    }
}.execute("http://upload.wikimedia.org/wikipedia/commons/thumb/7/72
```

After:

AUIL(Android-Universal-Image-Loader):

```
ImageLoader.getInstance().displayImage("http://upload.wikimedia.org
```

Picasso:

```
Picasso.with(context).load("http://upload.wikimedia.org/wikipedia/c
```

Glide:

```
Glide.with(context).load("http://upload.wikimedia.org/wikipedia/com
```

Fresco:

```
mImageView.setImageURI(Uri.parse("http://upload.wikimedia.org/wikip
```

除了寫法的簡便之外，為什麼要 Image Loader，什麼是 Image Loader ？

ImageLoader 會很聰明的知道 ImageView 在可視範圍內才去做下載與解壓縮 bitmap 一旦離開就停止一切作業。而且為了二次快速顯示，依據 ImageView 的可視長寬作參考最適 cache 。

1. 顯示快取 - 首先，為了再次顯示一樣的圖片時，可以快速顯示，所以我們會把 bitmap 暫存在記憶體 memory cache 。那在有限的 memory cache ， 我們要如何管理？常見的是 LRU cache 策略，memory cache 有限，最近看過的優先留下來。以及依據畫布大小的快取。
2. 儲存快取 - 再來，網路來的圖片、網址圖片，下載儲存快取管理 - Disk Cache。
3. 連線快取 - okhttp SPDY

在 fresco 出現之前，會比較推崇 AUIL(Android Universal Image Loader)，接著 glide 、picasso 。

追求小 code size 可以選 picasso ，輕簡。

AUIL 是因為在記憶體、儲存空間的快取策略還有其它有的沒有的都可以訂製。彈性比較大一點(所以 code size 大一點)。

只有 fresco 需要更換 layout class 原因是因為它為了效能，操作較低階的畫布。(Layout Class 都換了，所以順便支援 Gif 、影片? 誤)

對於 Image Loader 常見的需求：

- 更換 OkHttp 下載器
- 潤角、圓圖、顯示動畫
- 支援 Exif 轉正，支援影片快照轉正

Fresco - facebook

- ImagePipeLine - 更換 OkHttp 下載器 (有 bug，已解 PR#21)

picasso - square

- Factory -> 更換 OkHttp 下載器
- Transformer -> 潤角、圓圖、顯示動畫

Android-Universal-Image-Loader

- ImageDownloader -> 更換 OkHttp 下載器
- ImageDisplayer -> 潤角、圓圖、顯示動畫
- ImageDecoder -> 支援 Exif 轉正，支援影片轉正

glide

glide 許多 Google 官方 samples 都可看到它的蹤影，所以基本上 Google 有挺。

附錄 - 虛擬網址 - Facebook 真實圖片轉址

如果你的 ImageLoader 針對網路圖片，不支援轉址，又或者網址藏在 json 裡。ImageLoader 大多支援 ContentProvider 網址，`content://` 所以我們可以利用它來做虛擬網址轉址。

p.s. `https://graph.facebook.com/{id}/picture` 雖然本身有轉址能力，不過這裡為了教學所需，還是寫了一個 *FacebookPictureProvider* 作本地轉址範例。

例如:

GET `https://graph.facebook.com/601234567/picture?redirect=0`：

```
{
  "data": {
    "is_silhouette": false,
    "url": "https://fbcdn-profile-a.akamaihd.net/hprofile-ak-xpf1/v
  }
}
```

data.url 才是真實的圖片網址。

所以我們註冊一個內部網域：

`content://com.facebook.content.PictureProvider/601234567`

提供轉址到真實圖片網址：

`https://fbcdn-profile-a.akamaihd.net/hprofile-ak-xpf1/v/t1.0-1/p50x50/1234567`

```
public class FacebookPictureProvider extends NetworkPipeContentProv
    public static final String AUTHORITY = "com.facebook.provider.F
    public static final Uri CONTENT_URI = Uri.parse("content://" +
    private Facebook facebook;

    // content://com.facebook.provider.PictureProvider/601234567
    @Override
    public String getUri(Uri uri) { // 轉址 - 改寫網址
       Picture picture = facebook.picture(uri.getPathSegments().get

       return picture.data.url;
    }

    interface Facebook {
        // https://graph.facebook.com/{id}/picture
        // https://graph.facebook.com/601234567/picture
        @GET("/{id}/picture")
        Observable<Picture> picture(@Path("id") String id);
    }

    static class Picture {
        Data data;
    }

    static class Data {
        // "https://fbcdn-profile-a.akamaihd.net/hprofile-ak-xpf1/v
        String url;
    }

    @Override
    public boolean onCreate() {
        RestAdapter restAdapter = new RestAdapter.Builder()
            .setEndpoint("https://graph.facebook.com")
            .build();
        facebook = restAdapter.create(Facebook.class);
        return true;
    }
}
```

- https://github.com/yongjhih/facebook-content-provider

See Also

- Using okhttp backed for Volley - https://gist.github.com/bryanstern/4e8f1cb5a8e14c202750
- https://github.com/facebook/fresco/pull/21

json to POJO

- Gson
- Jackson
- Instagram/ig-json-parser
- LoganSquare

基本上 ig-json-parser、LoganSquare 是基於 Jackson 衍伸的，編譯時期的欄位處理，搭配 Jackson stream parsing 達成。

Gson 而是執行時期作欄位處理，自然有改善的空間。

LoganSquare 啟發於 Instagram/ig-json-parser 而有較好的包裝。

LoganSquare

```
// String jsonString:
//
// {
//   name: "Andrew",
//   id: 1
// }
User user = LoganSquare.parse(jsonString, User.class);
System.out.println("name: " + user.name);
System.out.println("id: " + user.id);
```

```
@JsonObject
public class User {
    @JsonField
    public String name;

    @JsonField
    public int id;
}
```

先撇除物件化描述，可以先探究，為什麼 JSONObject for loop 去爬會比較慢呢？

一個初步的原因是因為先要把 json string 塞成 JSONObject，表示已經繞完過一輪了。然後你再次繞一輪 JSONObject 就已經是兩輪了。

而 Jackson 提供的 streaming 就是正在繞的當下就會呼叫 callback 了。

See Also

- http://instagram-engineering.tumblr.com/post/97147584853/json-parsing

Clean architecture

先將程式碼擺放包裝劃分清楚再來針對動線做規劃。

Clean architecture 主要探討的是系統分層以及互動模型

常見的互動模型：

- MVC
- MVP (P, Presenter)
- MVVM (VM, ViewModel)
- Flux

在 Android 開發上，常見的是 MVP 與 MVVM 一般開發上，並不一定都只有一套互動模型在運行。

MVP 其實大家應該十分熟悉，只是沒有意識到而已。那就是 ListView/RecyclerView 就有用到了。

我們知道 MVP 三層，所以只要介紹中間那層 P 會比較直接，這邊稍微簡化舉例：

```
class UserCardPresenter extends Presenter<User> {
    public ViewHolder onCreateViewHolder(ViewGroup parent) {
        return new UserCardViewHolder(parent, R.layout.item_icon);
    }

    public void onBindViewHolder(/* View */UserCardViewHolder userC
        // userCardView.textView1.setText(item.name);
    }
}
```

MVVM 基本上，可當作是 MVP 的子集合，注重雙向連動，以利個別測試，你可以虛擬化其中一方做測試。

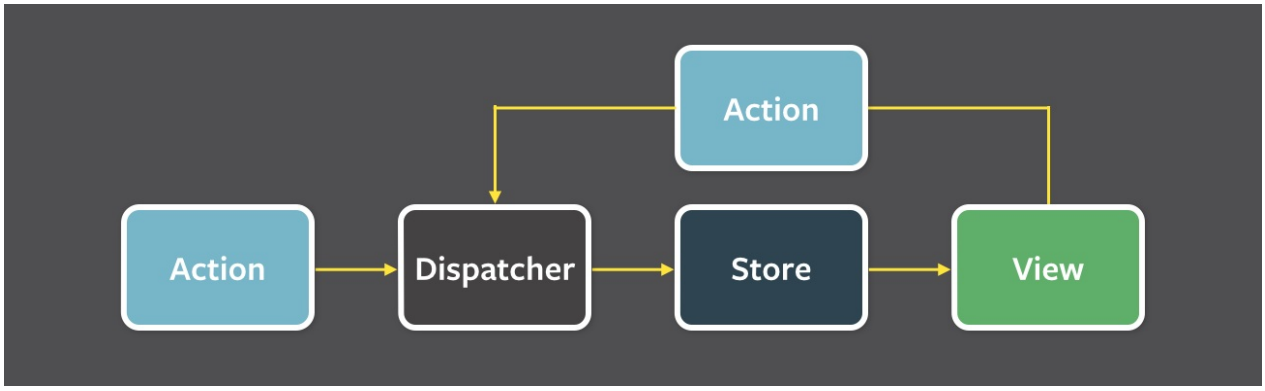
ogaclejapan/RxBinding MVVM

利用 RxJava 實現了 MVVM 的 two-way binding (雙向連動)

Data-Binding support v22

讓連動更簡便撰寫 VM 應該就更薄了。

Flux



不直接雙向回 model/store， 透過一個 dispatcher 來做管理。

See Also

- http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/
- http://fernandocejas.com/2015/07/18/architecting-android-the-evolution/
- http://manuel.kiessling.net/2012/09/28/applying-the-clean-architecture-to-go-applications/
- https://www.groupbuddies.com/posts/20-clean-architecture
- http://konmik.github.io/introduction-to-model-view-presenter-on-android.html
- https://medium.com/mobiwise-blog/android-basic-project-architecture-for-mvp-72f4b33252d0
- http://armueller.github.io/android/2015/03/29/flux-and-android.html
- http://lgvalle.xyz/2015/08/04/flux-architecture/

- https://github.com/skimarxall/RxFlux

Sample:

- https://github.com/android10/Android-CleanArchitecture
- https://github.com/sockeqwe/mosby
- https://github.com/JorgeCastilloPrz/Dagger2Scopes
- https://github.com/PaNaVTEC/Clean-Contacts

- https://github.com/techery/presenta
- https://github.com/antoniolg/androidmvp
- https://github.com/wongcain/MVP-Simple-Demo
- https://github.com/pedrovgs/EffectiveAndroidUI
- https://github.com/glomadrian/MvpCleanArchitecture
- https://github.com/spengilley/ActivityFragmentMVP
- https://github.com/jlmd/UpcomingMoviesMVP
- https://github.com/JorgeCastilloPrz/EasyMVP
- https://github.com/richardradics/MVPAndroidBootstrap
- https://github.com/richardradics/RxAndroidBootstrap
- https://github.com/inloop/AndroidViewModel
- https://github.com/lgvalle/android-flux-todo-app

Retrofit 1

rest api 直接對應成 java interface， 且搭配 Json Mapper 直接轉換。

一個類似於 JAX-RS/JSR-311) 的實現，基於一些理由並沒有完全按照 JAX-RS 實現。

Interface:

```
interface GitHub {
    @GET("/repos/{owner}/{repo}/contributors")
    Observable<List<Contributor>> contributors(
        @Path("owner") String owner,
        @Path("repo") String repo);

    @GET("/users/{user}")
    Observable<User> user(
        @Path("user") String user);

    @GET("/users/{user}/starred")
    Observable<List<Repo>> starred(
        @Path("user") String user);
}
```

Usage:

```
        RestAdapter restAdapter = new RestAdapter.Builder()
            .setEndpoint("https://api.github.com")
            .build();

        GitHub github = restAdapter.create(GitHub.class);

        github.contributors("ReactiveX", "RxJava")
            .flatMap(list -> Observable.from(list))
            .forEach(c -> System.out.println(c.login + "\t" + c.cor
```

Models:

```
    static class Contributor {
        String login;
        int contributions;
    }

    static class User {
        String name;
    }

    static class Repo {
        String full_name;
    }
```

Retrofit 1 網路處理

retrofit 內部運作也是使用 `Request` 概念來運行。

Retrofit 很顯然的，希望讓開發者專注 Restful 介面，e.g. `List<Repo> repos = github.repos();`

處理的介面不再是網路相關的 Request，反倒是刻意隱含了 Request，導致網路細部操作看似無法處理，所以提供了錯誤處理註冊，`Client` 配置等接口。

1. 常見的網路處理：
2. Cache
3. Retry
4. Time Out
5. Cancel
6. Priority

使用 OkHttpClient 大多可以處理。(AOSP 4.4 之後其實也內建，不確定有沒有包進 SDK)

如果沒有用 RxJava Observable 了話，確實 Retry policy 的部份確實有點殘念，也不是無解，只是要自己辛苦點寫 ErrorHandler 重發。

使用 RxJava `retry()` 達到重試次數效果：

```
int retryLimit = 3;
Observable<Repo> repos = github.repos()
    .retry(3);
```

使用 RxJava `retryWhen()` 達到 backoff 效果：

```
int retryLimit = 3;
float backoff = 1.3f

Observable<Repo> repos = github.repos()
    .retryWhen(attempt -> attempt.zipWith(Observable.range(1, retry
        .flatMap(i -> Observable.timer(i * backoff, TimeUnit.SECOND
```

Cancel 也是，如果沒有 RxJava 你只能寫 ExecutorService 處理或者操作 OkHttpClient。

使用 Rxjava `unsubcribe()` 取消：

```
Subscription reposSubscription = repos.subscribe();

reposSubscription.unsubscribe(); // 取消
```

其中比較困難的是 Priority，需要處理 OkHttpClient executor 。

基本上，「RxJava + Retrofit + OkHttpClient(supports SPDY)」一起用，應該是沒什麼太大的問題。

NotRetrofit

由於 retrofit 是執行時期處理 annotations 效能有改善的空間。NotRetrofit 是改用編譯時期處理。如同 google fork square/dagger 專案的理由一樣: google/dagger2。

https://github.com/yongjhih/NotRetrofit

Retrofit 2

```
interface GitHub {
    @GET("/repos/{owner}/{repo}/contributors")
    Call<List<Contributor>> contributors(
        @Path("owner") String owner,
        @Path("repo") String repo);
}
```

解決 Retrofit 1 長久以來存在的一些問題。不直接回傳物件，而是透過一個中介 `Call` / `Response` 來包裝，使得可以保留網路資訊 Headers 。

如何取得分頁？

```
interface GitHub {
    // ...
    @GET
    Call<List<Contributor>> contributorsPaginate(@Url String url);
}
```

```
Call<List<Contributor>> contributorsCall = github.contributors("squ
Response<List<Contributor>> contributorsResponse = contributorsCall
String nextLink = contributorsResponse.headers().get("Link");
Call<List<Contributor>> contributorsNextCall = github.contributorsP
Response<List<Contributor>> contributorsNextResponse = contributors
```

Retrofit 1 分析 (1.9.0)

```
private class RestHandler implements InvocationHandler {
  // ...
  public Object invoke(Object proxy, Method method, final Object[]
    // sync
    return invokeRequest(requestInterceptor, methodInfo, args);
    // Rx
    return rxSupport.createRequestObservable({
      (ResponseWrapper) invokeRequest(requestInterceptor, methodInf
    });
    // async
    Callback<?> callback = (Callback<?>) args[args.length - 1]; //
    httpExecutor.execute({
      (ResponseWrapper) invokeRequest(interceptorTape, methodInfo,
    });
  }
}
```

See Also

- https://github.com/yongjhih/RxJava-retrofit-github-sample
- https://github.com/yongjhih/NotRetrofit
- https://github.com/orhanobut/wasp

Orm - ActiveAndroid, DBFlow, Ollie

ORM, Object Relational Mapping, 物件關聯映射. 筆者將之譯為「關聯資料物件化」。

```
CREATE TABLE User (id INTEGER PRIMARY KEY AUTOINCREMENT, name  TEXT
```

```
class User {
    public Long id;
    public String name;
}

// set
User.create().name("Andrew Chen").save();

// get
Cursor cursor = MyDb.query("Select * from User where name = ?", "An
List<User> users = Orm.loadFromCursor(User.class, cursor);
User andrew = users.get(0);
```

以往的 Orm library 其實有些瓶頸， 例如：query 回來的都是 `List<Model>` ，這象徵著 query 萬一很多筆，Orm 必須全部繞完把每個都組成 Model ，真正有用過會發現會非常的緩慢，就算用 Transaction 包起來加速，也無法有效改善。這部份我們團隊寫了一個 CursorList 來作 Lazy 是等用到才去組，效果很好。

這個概念 sprinkles Orm library 也有發現，所以做了跟我們團隊一樣的事情。(DBFlow 有採納)

sprinkles 還有作自動升級程式也就是在 Model 新增欄位不用自己寫 alter table 。這點 DBFlow 應該也有採納。

目前我們是 ActivieAndroid + CursorList + EventBus 來作 Observing model，是由於開發時沒有 sprinkles 與 DBFlow，到現在已經沒法方便轉過去了。

另外 DBFlow 很巧妙的使用 `@interface Table RetentionPolicy.SOURCE` 編譯時期產生一個承載欄位名稱的子類別, 所以可以用 `ProfileModel$Table.DISPLAYNAME` 取出 "displayname" 欄位名稱，來有效減少 hard code 。

```
ProfileModel extends BaseModel {
    @Column String displayname;
}
```

緩慢確實有一部份是 reflection 所造成，而 reflection 緩慢的主因在於解析 annotation 。所以通常我們會把 NoteModel.class 所有的 column attribute 都存到一個 annotations cache map 避免重複解析 annotation 。所以只會慢在建立 annotations cache map 那第一次而已，Ollie ModelAdapter 主要解決的首重，就是這個廣泛存在於 Orm lib 初始化過慢問題。

```
List<Note> notes = new Select().from(Note.class).execute(); // huge
```

在萬筆資料的存取時，造成的緩慢是在於建立萬筆 Model 的累計時間，而不在於 reflection annotations (already in cache)。

1. 試想為什麼 SQLiteDatabase 存取時，回傳的不是 List<?> 而是 Cursor。
2. 試想 cursor 所有資料已經都從 DB 拉出來了？還是當跑 cursor.next(), cursor.get*() 才去拉資料？(DB 應該只是快速扼要的把數量與索引準備好，就馬上拋回 cursor ， 當 cursor.next() 才把該筆資料找出來 ， cursor.getString() 才動用 io 去拉資料。)
3. 再試想另一個：為什麼需要 CursorAdapter 而不是 loop cursor 塞進 ArrayAdapter 就好。 `// while (cursor.moveToNext()) notes.add(Note.load(cursor)); new ArrayAdapter(notes);`

問題都指向同一個 -> 因為大量的 io (loop cursor)以及 建立大量的物件 (Note.load(cursor)) 累計起來過慢。這些資料通常運算的部份已經在 query 條件過程中結束，僅為了拿出來顯示，但是這麼大量的資料全都要顯示出來的機會，其實很低，就這個狀況，可以 lazy 概念緩解，指到哪拿到哪。SQLiteCursor 基本上就是一種 lazy 的存在。

整理問題與解法：

1. 解析 annotation -> cache
2. loop cursor -> lazy
3. load from cursor -> lazy

量測方法：

1. 把 `com.activeandroid.util.SQLiteUtils.processCursor() do {} while (cursor.moveToNext())` 做空，單純測量 loop cursor 的耗時
2. 量測 `Model.loadFromCursor()`

名詞解釋

DTO, Data Transfer Object, 資料傳輸物件。「參數物件化」

flow + mortar

從 2014/01 Square 部落格初次登板 mortart and flow 的開頭可以得知， 一開始要解決的問題是 Fragment 換頁問題。

從 Android 4.0 之後大量改成 Fragment 導致一個 Activity 要掌管 Fragment transaction 換頁。

在一般用途上，筆者偷懶直接用 ViewPager + FragmentStatePagerAdapter + ViewPager.Transformer + 特製的 PagerIndicator， 基本上堪用。

FragmentMaster

另一種 ViewPager + FragmentStatePagerAdapter + ViewPager.Transformer 的整合方案。

See Also

- https://github.com/square/flow/
- https://corner.squareup.com/2014/01/mortar-and-flow.html
- https://www.bignerdranch.com/blog/an-investigation-into-flow-and-mortar/
- https://github.com/WeMakeBetterApps/MortarLib

EventBus: greenrobot/EventBus, Square/otto

通常用來處理跨 Activity, View, Thread 等溝通問題。

1. 通常遇到這種問題，一種是建立溝通管道，例如傳遞 callback 的方式。
2. 利用廣播方式，intent broadcast
3. 懶得傳遞 callback 就依靠全域變數註冊 callback.

greenrobot/EventBus 就是走第三種方案。

greenrobot/EventBus vs. Otto

網路上有很多比較，可以參考。這裡先給個簡單的參考： EventBus 比較快、 Otto 比較小。

補充 - 回復性、耦合性

儲存問題, 可回復性. 如果透過 saveInstanceState 就可以交由系統負責儲存。一種是儲存在 disk 如: sharedPreferences. 再來是 DB 。

也就是 http://endlesswhileloop.com/blog/2015/06/11/stop-using-event-buses/ 提及的：

> - Requires the data as a dependency.
> - Handle the state where the data is not available.

不過 http://endlesswhileloop.com/blog/2015/06/11/stop-using-event-buses/ 標題過激，事實上，只要不要濫用，能夠正常建立 callback 溝通管道的就建立，不得已我們再來考慮 EventBus 。

See Also

有 memory leak 的問題:

- #57 Weak reference to the subscriber, 解法: yongjhih/EventBus/commit/3d3c1ca6
- https://github.com/greenrobot/EventBus
- https://github.com/square/otto
- http://endlesswhileloop.com/blog/2015/06/11/stop-using-event-buses/

開源套件設置

要把編譯好的 jar/aar 套件，上傳到套件管理中心，方便別人下載使用：

```
dependencies {
    compile 'com.github.yongjhih:RetroFacebook:1.0.0'
}
```

那麼你要就上到一個 maven 套件伺服器。目前知名的伺服器中心有三家：

- mavenCentral
- jcenter
- jitpack

mavenCentral

由 sonatype 組織所維護, 最早的一家。 必須要開源才能被收錄。最嚴苛的一家。

申請方式，首先，你必須要有一個開源專案。

註冊 sonatype 帳號後，發 ticket，在 ticket 內容寫你的開源網址，通常是 github repository 網址，如：https://github.com/yongjhih/RetroFacebook，這樣你就會拿到 `com.github.yongjhih` 的 group。以後你都可以往這個群組裡面丟 jar/aar。

如果是公司行號要申請，就必須在 ticket 內容寫明你是域名的擁有者，例如：`com.infstory`，這樣你才能拿到 `com.infstory` Group。

很多上傳的方法，這裡就先不介紹。

上傳後，會先進到 stage 暫存區，做掃描行為後，你進到後台確定發布。不過通常半個到一個工作天才能夠真正進到 mavenCentral。

See Also:

- OSSRH

jcenter

由 bintray 公司所維護。原則上也是需要開源。jcenter 會鏡像 mavenCentral 。而目前 jcenter 是 android studio 預設的套件中心。加上目前由於網頁設計的比較便民，所以大多已經改由 jcenter 發布了。當然很多其他系統還是只有 mavenCentral 所以 jcenter 也提供同步到 mavenCentral，不過你要給它 OSSRH 的帳密就是了，當然 jcenter 也表明不會儲存你的帳密。

jitpack

大約是 2015 年初的時候成立的。最大的特點是，直接支援 github/bitbucket repository 。所以不用特別申請套件中心帳號。因為不是你去放套件，而是它自己來拉你的源碼來編譯 jar/aar。(託運算與儲存的廉價)

jitpack 使用方法

引用方 build.gradle:

```
repositories {
    // ...
    maven { url "https://jitpack.io" }
}

dependencies {
    compile 'com.github.{user}:{repo}:{version}'
}
```

提供方：

在 module 內 build.gradle 加入：

```
buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        // ...
        classpath 'com.github.dcendents:android-maven-gradle-plugin
    }
}

apply plugin: 'com.github.dcendents.android-maven'
apply from: 'android-javadoc.gradle'
```

以及新增 android-javadoc.gradle:

```
task sourcesJar(type: Jar) {
    from android.sourceSets.main.java.srcDirs
    classifier = 'sources'
}

task javadoc(type: Javadoc) {
    failOnError  false
    source = android.sourceSets.main.java.sourceFiles
    classpath += project.files(android.getBootClasspath().join(File
}

task javadocJar(type: Jar, dependsOn: javadoc) {
    classifier = 'javadoc'
    from javadoc.destinationDir
}

artifacts {
    archives sourcesJar
    archives javadocJar
}
```

如果你引用的了太多 jitpack 上面的套件，會需要很長的時間等待 jitpack 的編譯時間。故筆者作為一個函式庫貢獻者，盡可能還是採取 jitpack 與 jcenter 並行。甚者，如有閒暇會讓 jcenter 同步到上游 mavenCentral 。

p.s. jitpack 未來或許營利除了私有套件之外，可能可以提供付費服務，如報表等。javadoc 生成塞廣告。結合 travis-ci, cycle-ci 。同步 mavenCentral/jcenter

線上 javadoc

- javadoc.io for maven central, `http://javadoc.io/doc/{GROUP}{.SUBGROUP}/{ARTIFACT}/{VERSION}`
- jitpack, `https://jitpack.io/{GROUP}{/SUBGROUP}/{ARTIFACT}/{VERSION}/javadoc/`

ref.

- https://github.com/chrisbanes/gradle-mvn-push
- https://github.com/novoda/bintray-release

Assert 斷言 - assertj, truth

assertj-android

- *註: 2011 以 fest-android 釋出，後來加入 AssertJ 家族，更名為 assertj-android`

針對 Android 的類別作語法撿結語錯誤訊息的強化。

語法的簡潔：

Before(JUnit):

```
assertEquals(View.GONE, view.getVisibility());
```

After(AssertJ):

```
assertThat(view).isGone();
```

錯誤訊息的強化：

Before(JUnit):

```
Expected:<[8]> but was:<[4]>
```

After:

```
Expected visibility <gone> but was <invisible>
```

truth

- 註*:* 筆者是在 *2015/2* 留意到它
- 註*: 2014/12* 由 *google testing blog* 消息釋出

在沒有特定的類別下，提供一個置換錯誤訊息的能力。

Before(JUnit):

```
boolean buttonEnabled = false;
assertTrue(buttonEnabled);
```

After(truth):

```
ASSERT.that(buttonEnabled).named("buttonEnabled").isTrue();
```

錯誤訊息的強化：

Before(JUnit):

```
<false> was expected be true, but was false
```

After(truth):

```
"buttonEnabled" was expected to be true, but was false
```

assertj-rx

```
assertThat(observable.toBlocking()).completes();
```

```
assertThat(observable.toBlocking())
    .completes()
    .emitsSingleValue("hello");
```

```
assertThat(observable.toBlocking()).fails();
```

See Also

- https://github.com/square/assertj-android

- https://github.com/google/truth
- http://joel-costigliola.github.io/assertj
- https://github.com/google/truth/issues/43
- http://googletesting.blogspot.tw/2014/12/testing-on-toilet-truth-fluent.html
- https://github.com/ribot/assertj-rx

Mockito

虛擬物件

用來驗證是否如預期的互動行為，例如:

```
void add(List<String> list, String item) {
   list.add(item);
}

void testAddList() {
    List<String> mockedList = mock(List.class);
    add(mockedList, "one");
    verify(mockedList).add("one");
}
```

也可以偽裝行為，例如：

```
LinkedList mockedList = mock(LinkedList.class);

// 如果有人呼叫 get(0) ，就回傳 "first"
when(mockedList.get(0)).thenReturn("first");

assertEquals("first", mockedList.get(0));
```

RxParse 測項範例

測試 `ParseObservable.all(ParseQuery).subscribe(onNext, onError, onCompleted);` ，不應該先呼叫 onCompleted 才呼叫 onNext。

```
  /**
   * <code>ParseObservable.all(ParseQuery).subscribe(onNext, onErro
   */
  @Test
  public void testParseObservableAllNextAfterCompleted() {
```

```
    // 建立三個偽裝 ParseUser
    ParseUser user = mock(ParseUser.class);
    ParseUser user2 = mock(ParseUser.class);
    ParseUser user3 = mock(ParseUser.class);
    List<ParseUser> users = new ArrayList<>();
    when(user.getObjectId()).thenReturn("" + user.hashCode()); // 
    users.add(user);
    when(user2.getObjectId()).thenReturn("" + user2.hashCode());
    users.add(user2);
    when(user3.getObjectId()).thenReturn("" + user3.hashCode());
    users.add(user3);
    // 偽裝 ParseQueryController , 讓 ParseQuery.findInBackground(),
    ParseQueryController queryController = mock(ParseQueryControlle
    ParseCorePlugins.getInstance().registerQueryController(queryCon

    Task<List<ParseUser>> task = Task.forResult(users);
    // 偽裝呼叫 findAsync(ParseQuery.State, ParseUser, Task) 時, 回傳
    when(queryController.findAsync(
            any(ParseQuery.State.class),
            any(ParseUser.class),
            any(Task.class))
    ).thenReturn(task);

    // 偽裝呼叫 countAsync(ParseQuery.State, ParseUser, Task) 時, 回傳
    when(queryController.countAsync(
      any(ParseQuery.State.class),
      any(ParseUser.class),
      any(Task.class))).thenReturn(Task.<Integer>forResult(users.si

    ParseQuery<ParseUser> query = ParseQuery.getQuery(ParseUser.cla
    query.setUser(new ParseUser());

    final AtomicBoolean completed = new AtomicBoolean(false);
    rx.parse.ParseObservable.all(query)
        //.observeOn(Schedulers.newThread())
        //.subscribeOn(AndroidSchedulers.mainThread())
        .subscribe(new Action1<ParseObject>() {
        @Override public void call(ParseObject it) {
            System.out.println("onNext: " + it.getObjectId());
            if (completed.get()) { // 竟然發現 onCompleted 已經被呼叫
```

```
                fail("Should've onNext after onCompleted.");
            }
        }
    }, new Action1<Throwable>() {
        @Override public void call(Throwable e) {
            System.out.println("onError: " + e);
        }
    }, new Action0() {
        @Override public void call() {
            System.out.println("onCompleted");
            completed.set(true);
        }
    });

    try {
        ParseTaskUtils.wait(task); // 因為是非同步執行，需要等待 task 
    } catch (Exception e) {
        // do nothing
    }
}
```

安裝

```
dependencies { testCompile "org.mockito:mockito-core:1.+" }
```

See Also

- Mockito - https://github.com/mockito/mockito http://mockito.org/
- EasyMock - https://github.com/easymock/easymock http://easymock.org/
- PowerMock - https://github.com/jayway/powermock

Test - espresso, rxpresso

Ui 操作測試函式庫

RxPresso

```
rxPresso.given(mockedRepo.getUser("id"))
            .withEventsFrom(Observable.just(new User("some name")))
            .expect(any(User.class))
            .thenOnView(withText("some name"))
            .perform(click());
```

See Also

- https://github.com/novoda/rxpresso

Debug 除錯 - stetho, leakcanary

facebook/stetho

square/leakcanary

Test - robolectric

robolectric 提供 Android 相關的實體，例如： `Activity` , `Context` 以便離線測試。

```
@RunWith(RobolectricTestRunner.class)
public class MyActivityTest {

  @Test
  public void clickingButton_shouldChangeResultsViewText() throws E
    MyActivity activity = Robolectric.setupActivity(MyActivity.clas

    Button button = (Button) activity.findViewById(R.id.button);
    TextView results = (TextView) activity.findViewById(R.id.result

    button.performClick();
    assertThat(results.getText().toString()).isEqualTo("Robolectric
  }
}
```

Kotlin

卡特琳

特點：

- null-safety / Optional 最佳解決方案。 `?:`
- 語法簡潔 `for ((key, value) in map)`
- AutoValue?
- lambdas `setOnClickListener({ finish() })`
- Jake Wharton 加持 (誤

對於 android 來說，Kotlin 開始知名的時候，大概可以追溯到 2014 年中旬登上 Android 開發週報的：http://blog.gouline.net/2014/08/31/kotlin-the-swift-of-android/ ，剛開始看到是覺得確實很敏捷，但是對於成熟度抱著遲疑得態度。

在這之後，筆者是在 2015 年一月份 Jake Wharton 在 G+ 發表了一篇貼文之後，確實很多人跟筆者一樣，較為積極的看待這個語言。

除了這些特性之外，對於 android 來說，滿大的優勢在於 symbol size 以及 code size 相較於其他語言，十分羽量。(kotlin: 6k~, scala: 50k~)

POJO

Before:

```
@AutoValue
public abstract class Money {
  public abstract String currency();
  public abstract int amount();

  public static Money of(String currency, int amount) {
    return new AutoValue_Money(currency, amount);
  }
}
```

After:

```
data class Money(val currency: String, val amount: Int)
```

Lambda

Before:

```
thing.setListener(new Listener() {
  @Override public void onThing() {
    System.out.println("Thing!");
  }
});
```

After:

```
t.setListener(Listener { println("Thing!") })
```

Multi-assignment for loop

Before:

```
for (Map.Entry<String, String> entry : map.entrySet()) {
  System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

After:

```
for ((key, value) in map) {
  println(key + ": " + value)
}
```

簡便 **getter** 與 **setter**

```
public var context: Context? = null
  get
  set (value) {
    $context = value
  }
```

Null Safety

Nullable (類 @Nullable) :

```
var a: String? = "bar"
// `a = null` is ok
// `a.length()` throws exception
// `a?.length()` is ok
// `a!!.length()` // throws NPE if a is null
```

NonNull (類 @NonNull):

```
var a: String = "bar"
// `a = null` throws NPE
// `a.length()` is ok
```

Elvis operator, before:

```
val l = if (a != null) a.length() else -1
```

After:

```
val l = a?.length() ?: -1
```

集合運算子

any(其中)

其中一個項目成立，就回傳真。

語法：

```
list.any(() -> Boolean)
```

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 任何一個數字能整除 2
assertTrue(list.any { it % 2 == 0 })
// 任何一個數字大於 10
assertFalse(list.any { it > 10 })
```

all(都)

其中一個項目不成立，就回傳假。(所有的項目都成立，就回傳真。)

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 都小於 10
assertTrue(list.all { it < 10 })
// 都整除 2
assertFalse(list.all { it % 2 == 0 })
```

count(共有幾個)

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 整除 2 的項目共有 3 個
assertEquals(3, list.count { it % 2 == 0 })
```

reduce

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertEquals(21, list.reduce { total, next -> total + next })
```

reduceRight (倒著 reduce)

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertEquals(21, list.reduceRight { total, next -> total + next })
```

fold (類似有初始值的 reduce)

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 4 + 1+2+3+4+5+6 = 25
assertEquals(25, list.fold(4) { total, next -> total + next })
```

foldRight (同 fold，只是倒著)

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 4 + 6+5+4+3+2+1 = 25
assertEquals(25, list.foldRight(4) { total, next -> total + next }
```

forEach

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
list forEach { println(it) }
```

forEachIndexed

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
list forEachIndexed { index, value
       -> println("$index : $value") }
```

max

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertEquals(6, list.max())
```

maxBy

在算最大之前，先運算一次。

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 所有數值負數後，最大的那個是 1
assertEquals(1, list.maxBy { -it })
```

min

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertEquals(1, list.min())
```

minBy

不解釋

```
val list = listOf(1, 2, 3, 4, 5, 6)
assertEquals(6, list.minBy { -it })
```

none (沒有一個)

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 沒有一個整除 7
assertTrue(list.none { it % 7 == 0 })
```

sumBy

```
val list = listOf(1, 2, 3, 4, 5, 6)
// 2 餘數的總和
assertEquals(3, list.sumBy { it % 2 })
```

導入方法

- 可利用 Android Studio kotlin plugin 轉換程式碼 (轉完不一定可動，大多稍微改一下就好了)
- buildscript.dependencies: `classpath "org.jetbrains.kotlin:kotlin-gradle-plugin`
- `apply plugin: 'kotlin-android'`
- dependencies: `compile 'org.jetbrains.kotlin:kotlin-stdlib:0.12.200'`

FAQ

- 如果多方繼承(`class` / `interface`)時， `super.XXX()` 就會不清楚你要呼叫哪位 parent，所以改成 `super<>.XXX` 即可。

RxKotlin

```
observable<String> { subscriber ->
        subscriber.onNext("H")
        subscriber.onNext("e")
        subscriber.onNext("l")
        subscriber.onNext("")
        subscriber.onNext("l")
        subscriber.onNext("o")
        subscriber.onCompleted()
    }.filter { it.isNotEmpty() }.
    fold (StringBuilder()) { sb, e -> sb.append(e) }.
    map { it.toString() }.
    subscribe { result ->
      a.received(result)
    }

    verify(a, times(1)).received("Hello")
```

Anko

捨棄 xml 直接用 kotlin 語言來配置 UI。 立意良好(data-binding, react, angular alternative)。

- Anko Preview Plugin for idea

Functional constructs and patterns - funKtionale

2015/3 mid

1

```
val sum2ints = { x: Int, y: Int -> x + y }
val curried: (Int) -> (Int) -> Int = sum2ints.curried()

assertEquals(curried(2)(4), 6)

val add5 = curried(3)
assertEquals(add5(5), 8)
```

```
val format = { prefix: String, x: String, postfix: String ->
    "${prefix}${x}${postfix}"
}

val prefixAndBang = format(p3 = "!")

val hello = prefixAndBang(p1 = "Hello, ")

println(hello("world"))
```

對照表

(origin from Using Project Kotlin for Android)

Library	Jar Size	Dex Size	Method Count	Field Count
kotlin-runtime-0.10.195	354 KB	282 KB	1071	391
kotlin-stdlib-0.10.195	541 KB	835 KB	5508	458

Library	Jar Size	Dex Size	Method Count	Field Count
rxjava-1.0.4	678 KB	513 KB	3557	1668
support-v4-21.0.3	745 KB	688 KB	6721	1886
play-services-base-6.5.87	773 KB	994 KB	5212	2252
okio-1.2.0	54 KB	55 KB	508	76
okhttp-2.2.0	304 KB	279 KB	1957	882
retrofit-1.9.0	119 KB	93 KB	766	228
picasso-2.4.0	112 KB	97 KB	805	342
dagger-1.2.2	59 KB	54 KB	400	119
butterknife-6.0.0	48 KB	50 KB	307	73
wire-runtime-1.6.1	71 KB	71 KB	471	147
gson-2.3.1	206 KB	170 KB	1231	390
Total	2963 KB	2894 KB	21935	8063

Library	Jar Size	Dex Size	Method Count	Field Count
scala-library-2.11.5	5.3 MB	4.9 MB	50801	5820
groovy-2.4.0-grooid	4.5 MB	4.5 MB	29636	8069
guava-18.0	2.2 MB	1.8 MB	14833	3343

See Also

- https://github.com/ReactiveX/RxKotlin
- Using Project Kotlin for Android @JackWharton
- https://github.com/JetBrains/anko
- http://kotlinlang.org/docs/reference/
- http://try.kotlinlang.org/
- https://medium.com/@octskyward/kotlin-fp-3bf63a17d64a
- http://blog.zuehlke.com/en/android-kotlin/
- http://antonioleiva.com/collection-operations-kotlin/
- https://docs.google.com/presentation/d/1XTm-9WnwoiYhyHGamt-dHJBKmkEr3WajDCOGxgfIRsc
- https://github.com/importre/popular

Aspect

#AspectJ #AOP

AOP 切面導向設計。筆者認為也是一種 Meta Programming 的範疇。

我們還是以範例來作說明。在 Android 比較知名的代表作是 JackWharton/hugo 。

```
@DebugLog
public String getName(String first, String last) {
  SystemClock.sleep(15);
  return first + " " + last;
}
```

```
getName("Andrew", "Chen");
```

```
V/Example: ⇢ getName(first="Andrew", last="Chen")
V/Example: ⇠ getName [16ms] = "Andrew Chen"
```

```
@Aspect
public class Hugo {
  @Pointcut("within(@hugo.weaving.DebugLog *)") // 產生一個叫 within
  public void withinAnnotatedClass() {}

  @Pointcut("execution(* *(..)) && withinAnnotatedClass()")
  public void methodInsideAnnotatedType() {} // 攔方法參數型別

  @Pointcut("execution(*.new(..)) && withinAnnotatedClass()")
  public void constructorInsideAnnotatedType() {} // 攔建構子參數型別

  @Pointcut("execution(@hugo.weaving.DebugLog * *(..)) || methodIns
  public void method() {} // 攔方法

  @Pointcut("execution(@hugo.weaving.DebugLog *.new(..)) || constru
  public void constructor() {} // 攔建構子

  @Around("method() || constructor()") // 如果欄到就
  public Object logAndExecute(ProceedingJoinPoint joinPoint) throws
    long startNanos = System.nanoTime();
    Object result = joinPoint.proceed(); // 代為執行攔截區間
    long stopNanos = System.nanoTime();
    // 算一下時間、印一下。
  }
}
```

See Also

- https://github.com/JakeWharton/hugo

AccountManager

使用者帳號服務。

你可以以你的 app 名義透過 AccountManager 建立屬於你的使用者。

建立帳號(Top Down)：

```
accountManager.addAccountExplicitly((Account) account, "password",

Account account = new Account(username, accountType);

String accountType = "com.infstory"; // 帳號識別類型
String username = "yongjhih@example.com"; // 帳號名稱
```

相當然爾，你可以提供登入畫面，透過網路服務確定登入成功後，再執行上方程式來建立本地使用者。個資以及帳密都會存在系統裡，僅有 app 擁有者本身才可以存取密碼。(當然已經鮮少 *app* 會頻繁的使用密碼，一般後端的使用者授權服務，大多登入成功後，會提供一組暫時授權證書，大多操作皆使用此授權證書)

從系統中取得帳號資料(userdata, 帳密與授權證書需要 app 擁有者才可以存取)：

```
Account[] accounts = accountManager.getAccountsByType(accountType);
Account account = accounts.length != 0 ? accounts[0];
```

設定授權證書(accessToken/authToken):

```
accountManager.setAuthToken(account, authTokenType, authToken);
```

取得授權證書(accessToken/authToken):

```
String authToken = accountManager.peekAuthToken(account, authToken
```

GitHub App

我們可以寫一個 GitHub App，GitHub 網頁登入後，建立一個 GitHub 帳號，把 access token 存進去。

```
public class LoginActivity extends Activity {
  @Override public void onResume() {
    super.onResume();

    loginButton.onClick(() -> {
      final GitHub github = GitHub.create();

      // Populate username, password, clientId, clientSecret from U
      // ..

      AppObservable.bindActivity(this, github.getAccessToken(userna
        .subscribeOn(Scheduler.io())
        .subscribe(accessToken -> {
          String accountType = "com.github";
          Account account = new Account(username, "com.github");
          AccountManager accountManager = AccountManager.get(contex
          accountManager.addAccountExplicitly(account, password, us
          // 一般會寫 "user,user:email" 等, 在 oauth 稱為 scope, 不過帳
          String authType = "password"; // authType/scope/permissio
          String authToken = accessToken.accessToken;
          accountManager.setAuthToken(account, authType, authToken)

          finish();
        });
    });
  }
}
```

```
@Retrofit("https://api.github.com")
public abstract class GitHub {
  @FormUrlEncoded
  @POST("/oauth/token?grant_type=password") Observable<AccessToken>
      @Field("username") String email,
      @Field("password") String password,
      @Field("client_id") String clientId,
      @Field("client_secret") String clientSecret);
  public static GitHub create() { return new Retrofit_GitHub(); }
}
```

提供 3rd-party app 授權能力，充當行動裝置版本的 oauth provider 授權服務

3rd-party app 只要依據你帳號管理中心的 accountType 作為使用者中心/授權服務的識別名稱，呼叫即可透過系統叫起你的登入畫面，以提供授權證書。

```
AccountManagerFuture<Bundle> authTokenFutureBundle = accountManager
String authToken = authTokenFutureBundle.getString(AccountManager.K
```

GitHub App 就要比較辛苦的扮演使用者中心：

- 提供帳號管理中心
- 以及其登入畫面
- 向 AccountManager 建立使用者

先從剛熟悉的登入畫面開始，不只是單純的自己登入就好，要想辦法把 authToken 傳給委託人。

改繼承 AccountAuthenticatorActivity 讓它幫你把一些你不想知道的東西塞好，如果你是用 AppCompatActivity 之類的，請參考 AccountAuthenticatorActivity.java 源碼，自己塞。

```
public class LoginActivity extends AccountAuthenticatorActivity {
  @Override public void onResume() {
    super.onResume();

    loginButton.onClick(() -> {
      final GitHub github = GitHub.create();

      // ..

      AppObservable.bindActivity(this, github.getAccessToken(usern
        .subscribeOn(Scheduler.io())
        .subscribe(accessToken -> {
          String accountType = "com.github";
          Account account = new Account(username, "com.github");
          AccountManager accountManager = AccountManager.get(contex
          accountManager.addAccountExplicitly(account, password, us
          String authType = "password";
          String authToken = accessToken.accessToken;
          accountManager.setAuthToken(account, authType, authToken)

          // 2.
          Intent intent = new Intent();
          intent.putExtra(AccountManager.KEY_ACCOUNT_NAME, username
          intent.putExtra(AccountManager.KEY_ACCOUNT_TYPE, accountT
          intent.putExtra(AccountManager.KEY_AUTHTOKEN, authToken);
          ((AccountAuthenticatorActivity) this).setAccountAuthentic
          setResult(RESULT_OK, intent);

          finish();
        });
    });
  }
}
```

委託登入授權服務(Authenticator) 實現所有 AccountManager 相關的代理操作。

3rd-party 對 AccountManager.getAuthToken() 其實會委託給我們的登入授權服務 Authenticator 受理， 這部份我們再叫出剛配置好的登入畫面:

```
// 登入授權服務
public class GitHubAuthenticator extends AbstractAccountAuthenticat
  // 取得授權
  @Override
  public Bundle getAuthToken(AccountAuthenticatorResponse response,
        String authTokenType, Bundle options) throws NetworkErrorEx
    // 1. 你可以無條件提供授權
    // 2. 或者你可以提供一個畫面提醒使用者，是否同意提供授權

    // 如果還沒登入請使用者登入完再跳回來，繼續授權。

    AccountManager accountManager = AccountManager.get(context);
    // 拿拿看既有 GitHub Service 的權限授權
    String authToken = accountManager.peekAuthToken(account, authTo

    String accountType = "com.github";
    if (authToken != null) {
      Bundle bundle = new Bundle();
      bundle.putString(AccountManager.KEY_ACCOUNT_NAME, account.nam
      bundle.putString(AccountManager.KEY_ACCOUNT_TYPE, accountType
      bundle.putString(AccountManager.KEY_AUTHTOKEN, authToken);
      return bundle;
    }
    String authType = "password";

    // 拿不到 authToken 就當作登入吧 (由於目前我們的政策是，登入後，授權已經
    return addAccount(response, accountType, authType, (String[]) n

  }

  // 建立帳號
  @Override
  public Bundle addAccount(AccountAuthenticatorResponse response, S
        String authTokenType, String[] requiredFeatures, Bundle opt
        throws NetworkErrorException {

      // 登入畫面
      Intent intent = new Intent(context, LoginActivity.class);
      intent.putExtra(AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESP
```

```
        final Bundle bundle = new Bundle();
        bundle.putParcelable(AccountManager.KEY_INTENT, intent);
        return bundle;
    }
}
```

公開授權服務 AndroidManifest.xml:

```
<service
    android:name=".GitHubAuthenticatorService"
    android:exported="false">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator
    </intent-filter>
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/authenticator" />
</service>

<activity
  android:name=".LoginActivity"
  android:exported="true"
  android:label="@string/app_name" />
```

res/xml/authenticator.xml:

```
<account-authenticator xmlns:android="http://schemas.android.com/ap
    android:accountType="com.github"
    android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher"
    android:label="@string/account_label" />
```

為了聽 AccountManagerService 發出來的委託 Service:

```
public class GitHubAuthenticatorService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
        return new GitHubAuthenticator(this).getIBinder();
    }
}
```

搭配 NotRetrofit 自動授權

```
GitHub github = GitHub.create(context);
github.repos().subscribe(System.out::println);
```

```
public interface GitHub {
    @RequestInterceptor(GitHubAuthInterceptor.class)
    @GET("/{owner}/repos")
    Observable<Repo> repos(@Path String owner);
}
```

在呼叫 `GitHub.repos()` 時，會先呼叫 `accountManager.getAuthToken()`，拿到 token 後，才憑證發出 `GET /{owner}/repos` request。

```
@Singleton
public class GitHubAuthInterceptor extends AuthenticationIntercepto

    @Override
    public String accountType() {
        return "com.github";
    }

    @Override
    public String authTokenType() {
        return "com.github";
    }

    @Override
    public void intercept(String token, RequestFacade request) {
        if (token != null) request.addHeader("Authorization", "Bear
    }

}
```

搭配 retroauth 自動授權

```
@Authentication(accountType = R.string.auth_account_type, tokenType
public interface GitHub {
    @Authenticated
    @GET("/{owner}/repos")
    Observable<Repo> repos(@Path String owner);
}
```

retroauth 分析

AuthInvoker.java#L59

```
getAccountName()
    .flatMap(this::getAccount)
    .flatMap(this::getAuthToken) // 取得授權證
    .flatMap(this::authenticate) // 設定憑證
    .flatMap(o -> request)
    .retry((c, e) -> retryRule.retry(c, e));
```

Notification

使用 `v4.NotificationCompat.Builder` 。

See Also

- https://github.com/8tory/json2notification

開源八卦

Jake Wharton

是個看起來不用睡覺的人

隸屬:

- Square

著作:

- ActionBarSherlock
- AndroidViewPagerIndicator
- ButterKnife
- NotRxAndroid
- RxAndroid
- Dagger

square/moshi

moshi 是一個 json to POJO (Ojm, object json mapping) 的函式庫. 類似 Gson, jackson. 但是這個名字竟然是 Jake Wharton 他家的狗名...

Chris Banes

隸屬:

- Google

著作:

- PullToRefresh -> support.v4.SwipeRefreshLayout
- gradle-mvn-push

Square

Lucas Rocha, lucasr

- London, UK

隸屬:

- -Mozilla-
- Facebook
- TwoWayView

Jean-Baptiste Queru, JBQ

也是個看起來不用睡覺的人

- AOSP 技術首席

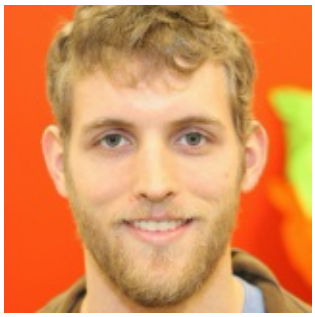
隸屬:

- Yahoo

Xavier Ducrohet

Android Build Tool team leader

Gregory Kick



隸屬:

- Google

居住：

- Chicago, IL

參與：

- Dagger2

Devoxx

經典問題 - 指引你「這該怎麼弄？」

這裡是從問題指引你到哪裡可以找到答案。

gradle 如何只測試單項?

```
./gradlew testDebug --tests='*.<testname>'
```

利用回傳空列表 `Collections.emptyList()` 來減少不必要的 null `List` 檢查

- 不再 `return new ArrayList<>();` 改用 `Collections.emptyList()`
- 不再 `return new HashSet<>();` 改用 `Collections.emptySet()`

不用檢查：

```
List<ResolveInfo> infos = packageManager.queryIntentActivities(inte
// infos == null? 像是官方文件就寫清楚肯定會回傳一個空列表，所以我們可以不用
for (info : infos) {
  System.out.println(info);
}
```

要怎麼寫非同步？

- Thread
- HandlerThread
- AsyncTask

要怎麼避免 Callback Hell

使用 RxJava 或者 Bolts 等具 promise 架構來整平

按鈕不想讓別人連按，要怎麼寫 debounce？

Restful client 要怎麼寫？用哪套？

- Retrofit
- Volley

See Also: ...

ImageLoader 要用哪套？

每套有其特性，一般短小精幹可用 picasso.

- 大量客製調整 AUIL
- Glide 效能考量

See Also: ...

Json 轉物件 要用哪套？

- LoganSquare 目前是非 reflection 方式，所以應有較高的效能表現。
- Jackson
- Gson

See Also: ...

我該從哪得知 Android App 開發的資訊？

前端 UI 、美工 Art 、平面字體挑選、換場動畫設計。

App 工程開發。

函式庫資訊

沒有回傳值的 Callback 用 RxJava 要怎麼寫？

> 想要問一下，我有一個 init() function 可能會做很久，所以我裡面開了一個 thread，然後 ui 傳入一個 callback 當我 init 完成之後我 callback 他。
>
> 如果想要改成 RxJava 的架構，所以我 init 回傳了 Observable，這裡 Boolean 其實有點多餘，因為我只是要單純的通知 ui 說我做完了，感覺有點硬是要用 Observable 來做。

如果你用 `AsyncTask<I, N, O>` 也會有類似的問題。

```
Observable<Void> initAsyncObs = Observable.defer(() -> Observable.j
  .subscribeOn(Schedulers.io())
  .observeOn(AndroidSchedulers.mainThread());

Void initSync() {
  // ...
  return null;
}
```

Background Thread API 用 RxJava `.observeOn(AndroidSchedulers.mainThread());` 好嗎？

> api 回傳 Observable， 然後由 ui subscribe 來取得資料更新畫面 而 Observable 先在 api 層設定好要在 background thread 執行，讓 ui 不用去管這段 不知道這樣的觀念是否正確，或是有更好的架構？

Android Threading/Scheduling 很多不只是 background 的問題，還有 lifecycle 問題，LoaderManager 其實也有在處理 lifecycle 問題。

所以 Ui 還是要自己善用 `AppObservable.bindActivity(activity, obs)`，`AppObservable.bindFragment(fragment, obs)`，`ViewObservable.bindView(view, obs)` 這些就是負責處理在 Android 上的 LifeCycle 以及 Threading 問題。 (rxandroid)

Ui Adapter 要用的，RxJava 回傳 `Observable<List<E>>` 還是 `Observable<E>`

> 感覺回傳 List 就很鳥，但是如果 ui 就是要收集到全部的資料在一次刷新，回傳 list 對 ui 來說好像比較方便

為了 UI adapater 方便，是否傳遞 `Observable<List<E>>` ？

首先在函式庫角度來看，它應該只做到單純性、重用性、通用性、彈性、操作性的最大化。

- 傳遞 `Observable<List<E>>` 的劣勢，很明顯的是大部分的 Rx Operator 都無法直接使用，也就是我說的操作性很低。所以一般狀況下，不會傳遞無操作性的介面。

在這個前提下，自然會導出 -> 函式庫不提供 `Observable<Collection/Iteratable>` 。

接著是下一層的問題， 那 Ui 該怎麼辦？很簡單，請服用： `Observable.toList()` ， `Observable.buffer()` , etc.

- See Also: https://kaif.io/z/compiling/debates/drg0Cf6oGj/dsCeOiO6Gz

作者群介紹

這本書本來目標就是共筆書籍，所以有第 N 作者。

第一作者

yongjhih, Andrew Chen

> 為什麼你會寫本書？

其實因緣際會，本來只想寫一篇而已。

一開始主要工作上的需要，採用了 RxJava 的函式庫，為了讓 讓夥伴們可以上手，但是它的「上手」文章略嫌不足，所以才開始撰寫了第一個章節 RxJava，接著想說順便把之前的一些開發經驗讓夥伴們可以瞭解，陸陸續續的就越寫越多章節，就變成這樣了。

> 你是哪時候開始接觸 Android App 開發？

嚴格講起來是從 2013 年，九月份左右才開始。到現在 2015 年約兩年多。

因為其實一開始是在手機系統廠工作，做了三年，從 2010 年中開始的到 2013 年中。那時也雖然也有改 Android App 不過就是 AOSP 系統內建的那些 App 稍微修修 Bug 調整一下 UI 之類的。獨立開發 App 到還不曾有過。App 實際開發起來，其實是截然不同的。

> 你現在在哪工作？

新創公司幫人寫寫 App 以及建立開發環境與佈署環境。

2013, SCM, ITS, Code Review, CI / gitlab, phabricator, jenkins. 內部系統都放置於 docker container.

2014, Backend Couchabase. Analytics System(with docker): Webdis, Redis, Logstash, ElasticSearch, Kibana.

2015, Parse.

> Github?

https://github.com/yongjhih

> LinkedIn?

https://tw.linkedin.com/in/yongjhih

第二作者

負責章節：

編輯

Docker

常用指令

啟動:

```
docker run -it ubuntu /bin/bash
```

or

```
docker run -it ubuntu:14.04 /bin/bash
```

進入 container

```
docker exec -it c007a10e4 bash
```

```
sudo usermod -aG docker andrew
```

編註

- 這是一篇與 Android 開發較微無關的章節。屬於後端平台性的章節。
- 筆者是在 2013/11, 0.6.7 之後的版本接觸。

資源

Development

- http://androidweekly.net/
- http://android-arsenal.com/ (http://android-arsenal.herokuapp.com/)
- https://github.com/trending?l=java
- https://github.com/wasabeef/awesome-android-ui / https://github.com/wasabeef/awesome-android-libraries
- http://developer.android.com/develop
- http://androidlibs.org/
- https://android-libs.com/

UI/UX

- https://dribbble.com/
- http://androidniceties.tumblr.com/
- https://www.materialup.com/
- http://developer.android.com/design/

Forum

- https://www.facebook.com/groups/270034869726161/
- https://www.facebook.com/groups/AKDGroup/

Documentation

- https://github.com/codepath/android_guides/wiki

Tools

- http://romannurik.github.io/AndroidAssetStudio/ (forks:

http://jgilfelt.github.io/AndroidAssetStudio)
- http://petrnohejl.github.io/Android-Cheatsheet-For-Graphic-Designers/
- http://angrytools.com/android/button
- http://inloop.github.io/shadow4android/

apk deploy for live demo

- appetize.io

Parse

一種 BaaS 。

RxParse

- https://github.com/yongjhih/RxParse

RxParse 測試

```
@Test
public void testParseObservableAllNextAfterCompleted() {
  ParseUser user = mock(ParseUser.class);
  ParseUser user2 = mock(ParseUser.class);
  ParseUser user3 = mock(ParseUser.class);
  List<ParseUser> users = new ArrayList<>();
  when(user.getObjectId()).thenReturn("" + user.hashCode());
  users.add(user);
  when(user2.getObjectId()).thenReturn("" + user2.hashCode());
  users.add(user2);
  when(user3.getObjectId()).thenReturn("" + user3.hashCode());
  users.add(user3);
  ParseQueryController queryController = mock(ParseQueryControlle
  ParseCorePlugins.getInstance().registerQueryController(queryCor

  Task<List<ParseUser>> task = Task.forResult(users);
  when(queryController.findAsync(
          any(ParseQuery.State.class),
          any(ParseUser.class),
          any(Task.class))
  ).thenReturn(task);
  when(queryController.countAsync(
    any(ParseQuery.State.class),
    any(ParseUser.class),
    any(Task.class))).thenReturn(Task.<Integer>forResult(users.si
```

```
    ParseQuery<ParseUser> query = ParseQuery.getQuery(ParseUser.cla
    query.setUser(new ParseUser());

    final AtomicBoolean completed = new AtomicBoolean(false);
    rx.parse.ParseObservable.all(query)
        //.observeOn(Schedulers.newThread())
        //.subscribeOn(AndroidSchedulers.mainThread())
        .subscribe(new Action1<ParseObject>() {
        @Override public void call(ParseObject it) {
            System.out.println("onNext: " + it.getObjectId());
            if (completed.get()) {
                fail("Should've onNext after completed.");
            }
        }
    }, new Action1<Throwable>() {
        @Override public void call(Throwable e) {
            System.out.println("onError: " + e);
        }
    }, new Action0() {
        @Override public void call() {
            System.out.println("onCompleted");
            completed.set(true);
        }
    });

    try {
        ParseTaskUtils.wait(task);
    } catch (Exception e) {
        // do nothing
    }
  }

  @Test
  public void testParseObservableFindNextAfterCompleted() {
    ParseUser user = mock(ParseUser.class);
    ParseUser user2 = mock(ParseUser.class);
    ParseUser user3 = mock(ParseUser.class);
    List<ParseUser> users = new ArrayList<>();
    users.add(user);
    users.add(user2);
```

```
    users.add(user3);
    ParseQueryController queryController = mock(ParseQueryControlle
    ParseCorePlugins.getInstance().registerQueryController(queryCon

    Task<List<ParseUser>> task = Task.forResult(users);
    when(queryController.findAsync(
            any(ParseQuery.State.class),
            any(ParseUser.class),
            any(Task.class))
    ).thenReturn(task);

    ParseQuery<ParseUser> query = ParseQuery.getQuery(ParseUser.cla
    query.setUser(new ParseUser());

    final AtomicBoolean completed = new AtomicBoolean(false);
    rx.parse.ParseObservable.find(query)
        //.observeOn(Schedulers.newThread())
        //.subscribeOn(AndroidSchedulers.mainThread())
        .subscribe(new Action1<ParseObject>() {
        @Override public void call(ParseObject it) {
            System.out.println("onNext: " + it);
            if (completed.get()) {
                fail("Should've onNext after completed.");
            }
        }
    }, new Action1<Throwable>() {
        @Override public void call(Throwable e) {
            System.out.println("onError: " + e);
        }
    }, new Action0() {
        @Override public void call() {
            System.out.println("onCompleted");
            completed.set(true);
        }
    });

    try {
        ParseTaskUtils.wait(task);
    } catch (Exception e) {
        // do nothing
```

```
    }
  }
```

- https://github.com/yongjhih/RxParse/blob/master/rxparse/src/test/java/com/parse/ParseObservableTest.java

Cloud Coding

Before, cloud code 原本的寫法：

```
Parse.Cloud.define("signInWithWeibo", function (request, response)
  //console.log(request.user + request.params.accessToken); // 取參
  // if (where) response.success(obj); // 回傳資料
  // else response.error(error); // 回報錯誤
}
```

After, 1. 改善註冊 RPC 的方法：

```
defineCloud(signInWithWeibo);

function signInWithWeibo(request, response) {
  // ...
}

function defineCloud(func) {
  Parse.Cloud.define(func.name, func); // func.name 可以取得 func 的
}
```

After, 2. 將 response 機制隱藏，轉成對應的 Promise ：

```
function promiseResponse(promise, response) {
  promise.then(function (o) {
    response.success(o);
  }, function (error) {
    response.error(error);
  })
}

/**
 * Returns the session token of available parse user via weibo acce
 *
 * @param {Object} request Require request.params.accessToken
 * @param {Object} response
 * @returns {String} sessionToken
 */
function signInWithWeibo(request, response) {
  promiseResponse(signInWithWeiboPromise(request.user, request.para
}

/**
 * Returns the session token of available parse user via weibo acce
 *
 * @param {Parse.User} user
 * @param {String} accessToken
 * @param {Number} expiresTime
 * @returns {Promise<String>} sessionToken
 */
function signInWithWeiboPromise(user, accessToken, expiresTime) {
  // ...
}
```

Parse.Cloud.httpRequest

回傳 `{Promise<HTTPResponse>}` ，所可以接龍：

```
/** @returns {Promise<String>} email */
function getEmail(accessToken) {
  // GET https://api.weibo.com/2/account/profile/email.json?access_
  // 這裡嚴格分離的 params 方式, 好處是未來改成 POST 也統一寫法
  return Parse.Cloud.httpRequest({
    url: "https://api.weibo.com/2/account/profile/email.json",
    params: {
      access_token: accessToken
    }
  }).then(function (httpResponse) {
    return JSON.parse(httpResponse.text)[0].email; // [ { email: "a
  });
}
```

Promise

- `Parse.Promise.as("Hello")`

```
Parse.Promise.as("Hello").then(function (hello) {
  console.log(hello);
});
```

- `Parse.Promise.when(helloPromise, worldPromise)`

```
var helloPromise = Parse.Promise.as("Hello");
var worldPromise = Parse.Promise.as(", world!");
Parse.Promise.when(helloPromise, worldPromise).then(function (hello
  console.log(hello + world);
});
```

flat and zip:

```
var helloPromise = Parse.Promise.as("Hello");
var worldPromise = Parse.Promise.as(", world!");
helloPromise.then(function (hello) {
  return Parse.Promise.when(Parse.Promise.as(hello), worldPromise);
}).then(function (hello, world) {
  console.log(hello + world);
});
```

Error handling:

```
/**
 * Returns email.
 *
 * @param {String} accessToken
 * @returns {Promise<String>} email
 */
function getEmailAlternative(accessToken) {
    return getEmail(accessToken).then(function (email) {
        if (!email) return Parse.Promise.error("Invalid email");

        return Parse.Promise.as(email);
    }, function (error) {
        return getUid(accessToken).then(function (uid) {
            return Parse.Promise.as(uid + "@weibo.com");
        });
    });
}

/**
 * Returns email
 *
 * @param {String} accessToken
 * @returns {Promise<String>} email
 */
function getEmail(accessToken) {
    return Parse.Cloud.httpRequest({
        url: "https://api.weibo.com/2/account/profile/email.json",
           params: {
```

```
                access_token: accessToken
            }
    }).then(function (httpResponse) {
        return JSON.parse(httpResponse.text)[0].email; // [ { emai
    });
};

/**
 * Returns uid.
 *
 * @param {String} accessToken
 * @returns {Promise<String>} uid
 */
function getUid(accessToken) {
    return Parse.Cloud.httpRequest({
        url: "https://api.weibo.com/2/account/get_uid.json",
           params: {
                access_token: accessToken
           }
    }).then(function (httpResponse) {
        return JSON.parse(httpResponse.text).uid; // { uid: 564744
    });
}
```

ref.

- https://gist.github.com/yongjhih/e196e01fc7da9c03ce7e

gradle

大多數的樣貌 build.gradle:

```
buildscript { // 建置設定區 - 引入建置相關插件庫
    repositories {
        jcenter() // 建置套件庫
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.2.3' // 插件庫
    }
}

apply plugin: 'com.android.application'

repositories {
    jcenter() // 函式套件庫
}

dependencies {
    //compile '{group}:{artifact}:{version}'
    //compile project('{module}')
}

android {} // com.android.application 插件設定區
```

多模組的目錄結構：

```
-a-project
|--build.gradle // 一般空檔, 除非需要子模組共用的設定, 可以在這裡設定
|--a-module/build.gradle
|--b-module/build.gradle
```

設定預設編譯哪些 module：

settings.gradle:

```
include ':a-module'
include ':b-module'
// or include ':a-module', 'b-module'
```

設定外部路徑：

```
// ...
include ':b-c-module'
project(':b-c-module').projectDir = new File(settingsDir, '../b-pro
```

build.gradle:

```
// ...
dependencies {
    // ...
    compile project(':b-c-module')
}
// ...
```

設定快取有效時間

預設 24 小時，每天一開始的編譯都會比較久。為了避免這種情形，可以拉長時間，如有必要再透過強制刷新來解決。

寫到專案設定：

```
configurations.all {
  resolutionStrategy {
    cacheDynamicVersionsFor 30, 'days'
    cacheChangingModulesFor 30, 'days'
  }
}
```

強制刷新套件

如果有些套件像是 SNAPSHOT.jar 剛更新，可透過 `--refresh-dependencies` 來刷到新的版本：

```
./gradlew --refresh-dependencies assembleDebug
```

顯示詳細的測試項目通過與失敗

```
tasks.withType(Test) {
  testLogging {
    exceptionFormat "full"
    events "passed", "skipped", "failed", "standardOut", "standardE
    showStandardStreams = true
  }
}
```

一般測試

```
./gradlew testDebug
```

測試單項

```
./gradlew testDebug --tests='*.<testname>'
```

顯示更多 lint 警告

```
tasks.withType(JavaCompile) {
  options.compilerArgs << "-Xlint:deprecation" << "-Xlint:unchecked
}
```

安裝 gradle wrapper

```
task wrapper(Wrapper) {
  gradleVersion = "2.4"
}
```

```
gradle wrapper
```

升級 gradle wrapper

修改 gradle/wrapper/gradle-wrapper.properties:

```
...
distributionUrl=https\://services.gradle.org/distributions/gradle-2
```

ref.

- https://docs.gradle.org/current/dsl/org.gradle.api.artifacts.ResolutionStrategy.html

React Native

- https://github.com/8tory/json2notification
- https://github.com/Avocarrot/json2view
- http://nativescript.org

目標都是 remote 顯示 native 畫面。

React 基本上是一種 View 的存在。

先無論描述 View 的語言簡潔與否，以遠端更新 View 的能力加上強健的生態，似乎不容小覷。

See also

- https://react.parts/native

Android SDK

列出套件

```
android list sdk
```

安裝 #1 套件

```
android update sdk --filter 1
```

移除套件

```
...
```

ref.

- http://tools.android.com/recent/updatingsdkfromcommand-line

adb

安裝 apk

```
adb install -r ${apk}
```

清除資料

```
adb shell pm clear `adb shell pm list packages ${部分名稱}`
```

移除 app

```
adb shell pm uninstall `adb shell pm list packages ${部分名稱}`
```

列出 apps

```
adb shell pm list packages ${部分名稱}
```

強制停止 app

```
adb shell am force-stop `adb shell pm list packages ${部分名稱}`
```

or

```
adb shell am kill `adb shell pm list packages ${部分名稱}`
```

喚醒手機

```
adb shell input keyevent 82 # KeyEvent.KEYCODE_MENU
```

佈景 Theme

常用的佈景架構：

res/values/theme.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 主色調 -->
    <color name="colorPrimary">@color/md_teal_400</color>
    <color name="colorPrimaryDark">@color/md_teal_500</color>

    <!-- 應用程式主要佈景決定使用明亮地暗底白字 Theme.AppCompat.Light.Dar
    <style name="Theme.App" parent="{Theme.AppCompat.Light.DarkActi
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</iter
    </style>

    <!-- 常用分頁佈景 -->
    <style name="Theme.App.ToolBar" parent="Theme.App.NoActionBar">
    </style>

    <style name="Theme.App.ToolBar.ActionBarOverlay" parent="Theme.
        <item name="windowActionBarOverlay">true</item>
        <item name="android:windowActionBarOverlay">true</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="windowActionModeOverlay">true</item>
        <item name="actionModeBackground">@color/colorPrimary</iter
    </style>

    <style name="Widget.App.ToolBar" parent="Theme.App">
        <item name="background">@color/colorPrimary</item>
        <item name="android:background">@color/colorPrimary</item>
        <item name="theme">@style/ThemeOverlay.AppCompat.Dark.Actio
    </style>

    <style name="Theme.App.ToolBar.ActionBarOverlay.NoDisplayOption
        <item name="displayOptions"></item>
```

```
        <item name="android:displayOptions"></item>
    </style>

    <!-- 如果 actionbar 需要壓過內容 -->
    <style name="Theme.App.ActionBarOverlay" parent="Theme.App">
        <item name="windowActionBarOverlay">true</item>
        <item name="android:windowActionBarOverlay">true</item>
        <item name="android:windowContentOverlay">@null</item>
    </style>

    <!-- 如果 actionbar 需要壓過內容且自訂 navigation bar -->
    <style name="Theme.App.ActionBarOverlay.FullScreen" parent="The
        <item name="android:windowFullscreen">true</item>
        <item name="android:windowContentOverlay">@null</item>
    </style>

    <!-- 全螢幕無系統列 -->
    <style name="Theme.App.NoActionBar.FullScreen" parent="Theme.Ap
        <item name="android:windowFullscreen">true</item>
        <item name="android:windowContentOverlay">@null</item>

        <item name="android:windowBackground">@color/colorPrimary</
        <item name="android:colorBackground">@color/colorPrimary</i
    </style>

    <style name="Theme.App.NoWindowContentOverlay" parent="Theme.Ap
        <item name="android:windowContentOverlay">@null</item>
    </style>

    <style name="Theme.App.NoActionBar" parent="Theme.App"> <!-- Co
        <item name="android:windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
    </style>
</resources>
```