# 7. Сеть и реактивность.

## Handler

В Android к **потоку** (thread) может быть привязана **очередь сообщений**. Мы можем помещать туда **сообщения**, а система будет за очередью следить и отправлять сообщения на **обработку**. При этом мы можем указать, чтобы сообщение ушло на обработку не сразу, а спустя определенное кол-во времени.

Handler - это механизм, который позволяет **работать** с очередью сообщений. Он привязан к конкретному потоку (thread) и работает с его очередью. Handler умеет **помещать** сообщения в очередь. При этом он ставит самого себя в качестве **получателя** этого сообщения. И когда приходит время, система достает сообщение из очереди и **отправляет** его адресату (т.е. в Handler) на обработку.

Показать пример в студии

*/\*\* Called when the activity is first created. \*/*

public void onCreate(Bundle *savedInstanceState*) {

super.onCreate(*savedInstanceState*);

setContentView(R.layout.*activity\_main*);

tvStatus = (TextView) findViewById(R.id.*tvStatus*);

pbConnect = (ProgressBar) findViewById(R.id.*pbConnect*);

btnConnect = (Button) findViewById(R.id.*btnConnect*);

h = new Handler() {

@Override

public void handleMessage(android.os.Message *msg*) {

switch (*msg*.what) {

case STATUS\_NONE:

btnConnect.setEnabled(true);

tvStatus.setText("Not connected");

break;

case STATUS\_CONNECTING:

btnConnect.setEnabled(false);

pbConnect.setVisibility(View.*VISIBLE*);

tvStatus.setText("Connecting");

break;

case STATUS\_CONNECTED:

pbConnect.setVisibility(View.*GONE*);

tvStatus.setText("Connected");

break;

}

}

;

};

h.sendEmptyMessage(STATUS\_NONE);

}

public void onclick(View *v*) {

Thread *t* = new Thread(new Runnable() {

public void run() {

try {

// устанавливаем подключение

h.sendEmptyMessage(STATUS\_CONNECTING);

TimeUnit.*SECONDS*.sleep(2);

// установлено

h.sendEmptyMessage(STATUS\_CONNECTED);

// выполняется какая-то работа

TimeUnit.*SECONDS*.sleep(3);

// разрываем подключение

h.sendEmptyMessage(STATUS\_NONE);

} catch (InterruptedException *e*) {

*e*.printStackTrace();

}

}

});

*t*.start();

}

## AsyncTask

[doInBackground](http://developer.android.com/reference/android/os/AsyncTask.html#doInBackground(Params...)) – будет выполнен в новом потоке, здесь решаем все свои тяжелые задачи. Т.к. поток не основной - не имеет доступа к UI.

[onPreExecute](http://developer.android.com/reference/android/os/AsyncTask.html#onPreExecute()) – выполняется перед doInBackground, имеет доступ к UI

[onPostExecute](http://developer.android.com/reference/android/os/AsyncTask.html#onPostExecute(Result)) – выполняется после doInBackground (не срабатывает в случае, если AsyncTask был отменен - об этом в следующих уроках), имеет доступ к UI

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {

protected Long doInBackground(URL... urls) {

int count = urls.length;

long totalSize = 0;

for (int i = 0; i < count; i++) {

totalSize += Downloader.downloadFile(urls[i]);

publishProgress((int) ((i / (float) count) \* 100));

// Escape early if cancel() is called

if (isCancelled()) break;

}

return totalSize;

}

protected void onProgressUpdate(Integer... progress) {

setProgressPercent(progress[0]);

}

protected void onPostExecute(Long result) {

showDialog("Downloaded " + result + " bytes");

}

}

new DownloadFilesTask().execute(url1, url2, url3);

## RxJava

До этог

|  |  |  |
| --- | --- | --- |
|  | **Реактивные потоки (С поддержкой backpressure)** | **Без поддержки backpressure** |
| 0…n элементов, complete | error | Flowable | Observable |
| item | complete | error |  | Maybe |
| item | error |  | Single |
| complete | error |  | Completable |

## Создание источников

Flowable.*just*("Hello world")

рассказать, почему ничего не происходит

Flowable.*just*("Hello world")

.subscribe(*str* -> {

Log.*d*("LES", *str*);

});

теперь в лог печатается строка

Пример через .create

Observable.*create*(new ObservableOnSubscribe<String>() {

@Override

public void subscribe(ObservableEmitter<String> *emitter*) throws Exception {

*emitter*.onNext("first");

*emitter*.onNext("second");

*emitter*.onComplete();

}

}).subscribe(*str* -> Log.*d*("LES", *str*));

Пример через .fromCallable()

Flowable.fromCallable(() -> {  
 return "Done";  
})

Пример блокирующего вызова:

Disposable *disposable* = Flowable.*fromCallable*(() -> {

Thread.*sleep*(5000); // imitate expensive computation

return "Done";

}).subscribe(*str* -> Toast.*makeText*(this, "done", Toast.*LENGTH\_SHORT*).show());

## Schedulers

* Schedulers.computation(): Run computation intensive work on a fixed number of dedicated threads in the background. Most asynchronous operator use this as their default Scheduler.
* Schedulers.io(): Run I/O-like or blocking operations on a dynamically changing set of threads.
* AndroidSchedulers.*mainThread*()

Пример неблокирующего вызова:

Disposable *disposable* = Flowable.*fromCallable*(() -> {

Thread.*sleep*(5000); // imitate expensive computation

return "Done";

})

.subscribeOn(Schedulers.*computation*())

.observeOn(AndroidSchedulers.*mainThread*())

.subscribe(*str* -> Toast.*makeText*(this, "done", Toast.*LENGTH\_SHORT*).show());

## Операторы и цепочки

private Observable<List<String>> getArrayList(String *str*) {

List<String> *result* = new ArrayList<>();

*result*.add(*str*);

*result*.add("first");

*result*.add("second");

*result*.add("third");

return Observable.*just*(*result*);

}

Disposable *disposable* = Observable.*fromCallable*(() -> {

Thread.*sleep*(5000); // imitate expensive computation

return "Done";

})

.map(*str* -> *str* + "123")

.flatMap(new Function<String, ObservableSource<List<String>>>() {

@Override

public ObservableSource<List<String>> apply(String *str*) throws Exception {

return getArrayList(*str*);

}

})

.flatMap(new Function<List<String>, ObservableSource<String>>() {

@Override

public ObservableSource<String> apply(List<String> *strings*) throws Exception {

return Observable.*fromIterable*(*strings*);

}

})

.map(*str* -> *str* + " one more")

.subscribeOn(Schedulers.*computation*())

.observeOn(AndroidSchedulers.*mainThread*())

.subscribe(*str* -> Log.*d*("LES", "finish: " + *str*));

Рассказать про отписку от disposable.

[http://rxmarbles.com/](http://rxmarbles.com/#zip) - нагядно про операторы

## Поддержка лябмд

compileOptions {

sourceCompatibility JavaVersion.*VERSION\_1\_8*

targetCompatibility JavaVersion.*VERSION\_1\_8*

}

## Загрузка картинок

Glide <https://github.com/bumptech/glide>

ImageView imageView = (ImageView) findViewById(R.id.my\_image\_view);  
GlideApp.with(this).load("<http://goo.gl/gEgYUd>").into(imageView);

## HTTP

**HTTP** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *HyperText Transfer Protocol* — «протокол передачи [гипертекста](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82)») — [протокол](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [прикладного уровня](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) передачи данных

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. Стартовая строка ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Starting line*) — определяет тип сообщения;
2. Заголовки ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Headers*) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Message Body*) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

URL: https://api.a101.handh.ru/v1/users/0

Method: GET

Protocol: http/1.1

Status: Complete

Response: 200 OK

SSL: Yes

Request time: Sun Jul 08 17:09:41 GMT+03:00 2018

Response time: Sun Jul 08 17:09:43 GMT+03:00 2018

Duration: 1691 ms

Request size: 0 B

Response size: 595 B

Total size: 595 B

---------- Request ----------

User-Agent: a101-android/0.3.1-debug

Content-Type: application/json

X-Auth-Token: OZEHwdj-sqfQMAGqN2JbVN2aaqpFwj4QbZkpWFUFdSfDMv8m\_V

---------- Response ----------

Server: nginx/1.4.6 (Ubuntu)

Date: Sun, 08 Jul 2018 14:09:35 GMT

Content-Type: application/json; charset=UTF-8

Transfer-Encoding: chunked

Connection: keep-alive

X-Rate-Limit-Limit: 200

X-Rate-Limit-Remaining: 199

X-Rate-Limit-Reset: 0

Access-Control-Expose-Headers:

Set-Cookie: \_csrf=zlSTpxHYbYWZ6F2iBWfK6JdMr7QK2fQO; path=/; HttpOnly

{

"code": 1,

"name": "",

"message": "",

"data": {

"user": {

"id": 3,

"firstName": "Алексей",

"lastName": "Головин",

"patronymicName": "Николаевич",

"avatar": "https://storage.a101.handh.ru/user\_avatar/3/large.476b047909c7b977b6cd5c30f59680e0.jpg",

"birthday": "1983-06-22",

"phone": {

"countryCode": "ru",

"internationalCode": "7",

"nationalNumber": "9199677102",

"mask": "000 000 00 00",

"image": "https://storage.a101.handh.ru/flag/ru.png"

},

"email": "hhh@hh.hh",

"apartments": [

13,

12,

2

],

"syncComplete": false,

"registration": "2018-04-05T16:14:01+0300",

"integration": "all"

}

},

"time": "2018-07-08T17:09:35+0300"

}

#### **GET**[[править](https://ru.wikipedia.org/w/index.php?title=HTTP&veaction=edit&section=10) | [править код](https://ru.wikipedia.org/w/index.php?title=HTTP&action=edit&section=10)]

Используется для запроса содержимого указанного ресурса.

#### **POST**[[править](https://ru.wikipedia.org/w/index.php?title=HTTP&veaction=edit&section=12) | [править код](https://ru.wikipedia.org/w/index.php?title=HTTP&action=edit&section=12)]

Применяется для передачи пользовательских данных заданному ресурсу.

и много других

## JSON XML

## 

## Retrofit

<http://square.github.io/retrofit/> Retrofit  
public interface GitHubService {  
 @GET("users/{user}/repos")  
 Call<List<Repo>> listRepos(@Path("user") String user);  
}

Retrofit retrofit = new Retrofit.Builder()  
 .baseUrl("<https://api.github.com/>")

.addConverterFactory(GsonConverterFactory.create(*gson*))

.setDateFormat(Constants.*DATE\_FORMAT*)

.build();  
  
GitHubService service = retrofit.create(GitHubService.class);

<https://github.com/google/gson> GSON

val *gson* = GsonBuilder()

.addConverterFactory(GsonConverterFactory.create(*gson*))

Связка rx и ретрофита

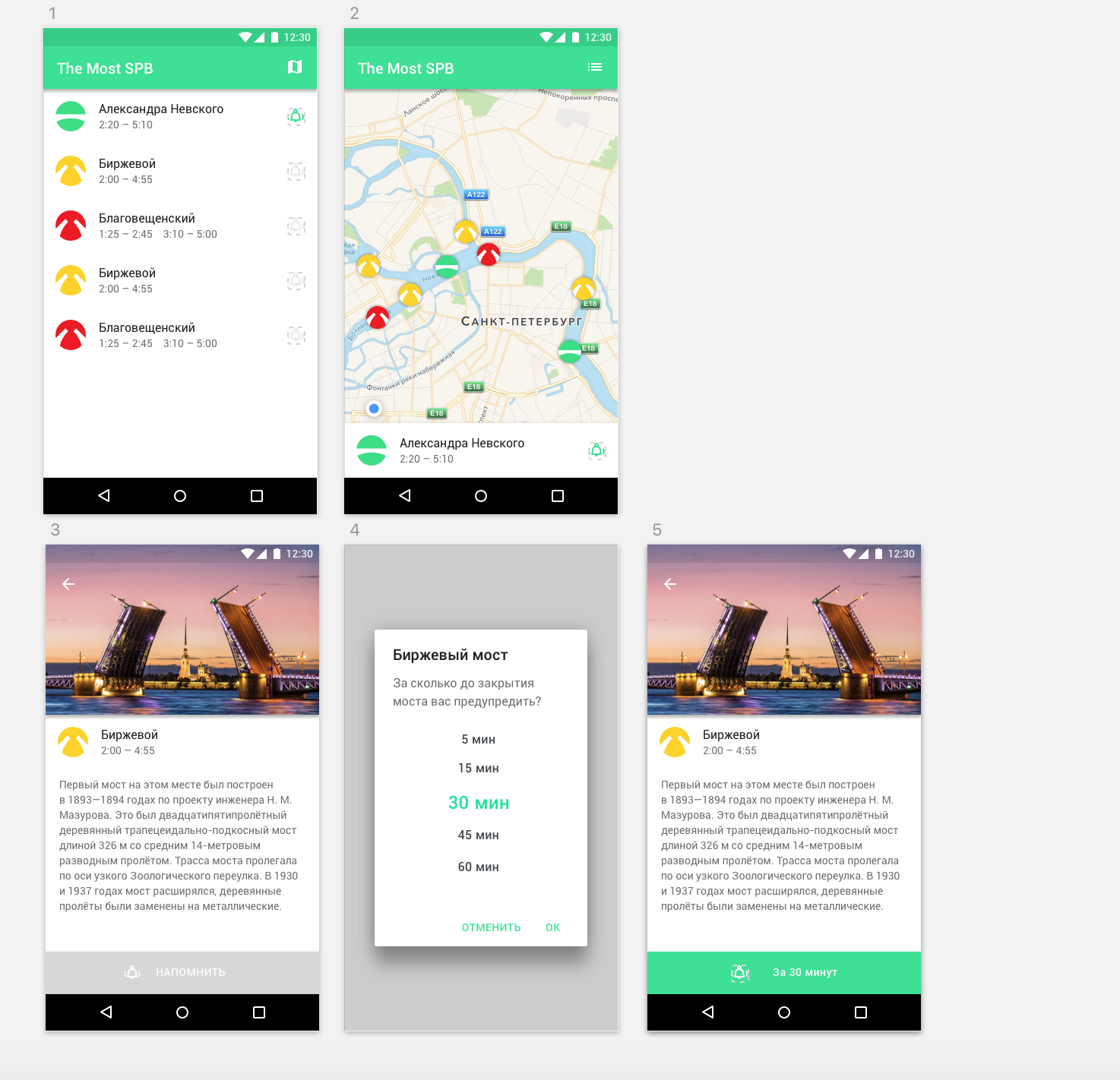
<https://github.com/square/retrofit/tree/master/retrofit-adapters/rxjava2>   
  
Retrofit retrofit = new Retrofit.Builder()  
 .baseUrl("https://example.com/")  
 .addCallAdapterFactory(RxJava2CallAdapterFactory.create())  
 .build();

interface MyService {  
 @GET("/user")  
 Single<User> getUser();  
}

#### Полезные ссылки

1. обязательная к прочтению статья <https://habr.com/company/badoo/blog/328434/> **Обязательна к прочтению**
2. серия статей про rx <https://habr.com/post/265269/> . рекомендуема к прочтению (немного устарела, т.к. есть вторая версия rx)
3. для особо интересующихся и желающих познать насколько глубока нора <https://habr.com/post/336268/>
4. Официальная документация: <https://github.com/ReactiveX/RxJava>
5. Handler - <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/143-urok-80-handler-nemnogo-teorii-nagljadnyj-primer-ispolzovanija.html> и последующие уроки
6. AsyncTask <https://developer.android.com/reference/android/os/AsyncTask#doInBackground(Params...)> <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/149-urok-86-asynctask-znakomstvo-neslozhnyj-primer.html>
7. <http://square.github.io/retrofit/> - Retrofit
8. <https://github.com/google/gson> GSON
9. <https://ru.wikipedia.org/wiki/HTTP> Http, <https://habr.com/post/215117/> - проще про http

#### Задание



Стек технологий для дз:

1. Retrofit
2. RxJava и RxAndroid
3. Gson

Добавить в проект rx -

implementation "io.reactivex.rxjava2:rxjava:2.x.y"

На момент написания, последняя версия - 2.1.14

Добавить

implementation 'io.reactivex.rxjava2:rxandroid:2.0.2'

Связка rx и ретрофита

<https://github.com/square/retrofit/tree/master/retrofit-adapters/rxjava2>

Есть GET запрос на список мостов <http://gdemost.handh.ru/api/v1/bridges/?format=json>

1. **Экран списка мостов**

Необходимо реализовать экран со списком мостов, который наполняется из запроса выше.  
Под названием отображается время, когда мост разведен.

Зеленый индикатор - мост сведен

Желтый - осталось меньше часа до развода моста

Красный - мост разведен

Колокольчик в списке означает, что пользователь установил напоминание о разводе данного моста.

**2. Экран информации о мосте.**

При нажатии на мост в списке - должен открываться экран детальной информации о мосте. Поведение шапки на этом экране реализовать на свое усмотрение. Можно использовать CollapsingToolbar для схлопывания картинки.

Кнопка снизу должна ставить напоминание о том, что мост разведут в такое-то время. Использовать <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/204-urok-119-pendingintent-flagi-requestcode-alarmmanager.html>

Данные на этот экран передаются со списка

**3. Экран карты (дополнительное задание)**

На экране со списком мостов есть переключатель сверху на вид карты.   
При нажатии на любой пин моста, снизу открывается краткая информация о мосте. При нажатии на нижний блок открывается детальная информация о мосте.   
Использовать Google Maps <https://developers.google.com/maps/documentation/android-sdk/intro>

о вы использовали императивный стиль программирования - когда команды выполняются одна за другой.

RxJava предлагает использовать реактивный стиль - когда есть источники данных и потребители, которые реагируют на изменения этих источников данных. т.е. все выполняется не последовательно, а асинхронно.

Нарисовать на доске пример, когда есть источник данных, и когда он что-то илучает что-то происходит.

Современные приложения так и работают. в фоне происходит много всего, при этом интерфейс не блокируется.

В RxJava 2 источники представлены двумя основными типами — Flowable и Observable. Рассказать, чем они отличаются.