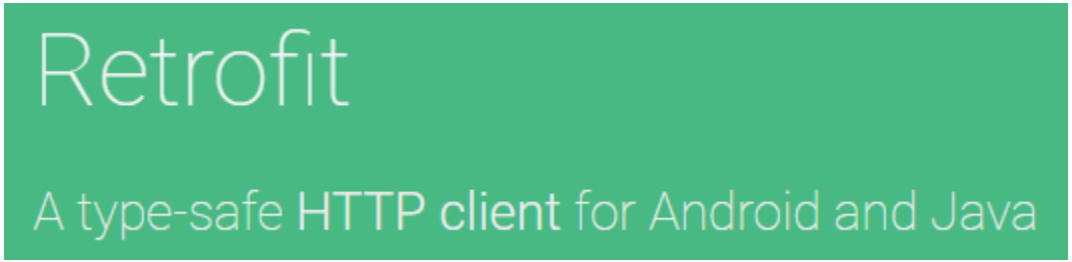
Android Retrofit2 Library

Jeon Yongtae
https://github.com/yongtaii/yongapps

1 Retrofit Library

Retrofit Library

Retrofit



Square에서 제공하는 Open Source Library (https://square.github.io/retrofit/)
RESTful 웹서비스를 안드로이드나 자바환경에서 쉽게 이용할 수 있게 해주는 라이브러리

Retrofit & OkHttp (from Squre)

OkHttp

- Lower-level HTTP Connection details을 다룬다
- Retrofit 작업을 위한 HTTP 통신 방법으로 OkHttp 라이브러리를 이용한다
- 따라서 Retrofit 라이브러리는 기본적으로 OkHttp 라이브러리를 포함한다

Retrofit

- 파라미터, 쿼리, 헤더 등 매핑작업 등을 도와준다 (Generating URL)
- 결과 처리작업 등을 편리하게 도와준다 (Convert를 통해 Response body를 Parsing)
- OkHttp의 윗단에서 사용될 수 있다

Retrofit을 사용하지 않고 OkHttp만을 이용해서도 작업이 가능하나 URL 매핑, 파라미터 매핑, 헤더세팅 등의 귀찮은 작업들이 많아지기 때문에 OkHttp와 Retrofit을 함께 사용한다

2 Why Retrofit ?

HttpClient Library 를 사용하면 ?

HTTP통신을 가장 간단히 사용한다면, HttpURLConnection을 많이 사용했을 것이다.
Java.net에 내장되어 있기 때문에 별도의 라이브러리가 필요 없다.
그럼, 이런 클래스를 놔두고 왜 굳이 Retrofit, Okhttp, volley 라이브러리들을 사용할까?

HTTP 통신개발의어려움

정말 간단히 사용하는 경우 그럴 수도 잇지만, 그렇지 않은 경우 고려할 것들이 많다.

1) 연결
2) 캐싱
3) 실패한 요청의 재시도
4) 스레딩
5) 응답 분석
6) 오류 처리
7) etc..

HTTP 요청을 위해 저많은 것들을 개발하다보면 배보다 배꼽이 커질 수 있다.

Volley vs Retrofit

AsyncTask vs Volley vs Retrofit

	One Discussion	Dashboard (7 requests)	25 Discussions
AsyncTask	941 ms	4,539 ms	13,957 ms
Volley	560 ms	2,202 ms	4,275 ms
Retrofit	312 ms	889 ms	1,059 ms

Retrofit이 가장 빠른 응답속도를 보여준다

Retrofit 장점

Retrofit

- 빠른 응답 속도
- 간단한 구현 방법
- CALL 요청 취소 가능
- 동기/비동기 선택 가능

  (AsyncTask를 사용하지않고도 Backgrond Thread에서 작업을 수행 한후 Callback 을 통해 MainThread에서 UI업데이트를 할 수 있다)

3 API Declaration

Setting Up Retrofit Interface

API interface

```
interface APIInterface {

    @GET("/api/unknown")
    Call<MultipleResource> doGetListResources();

    @POST("/api/users")
    Call<User> createUser(@Body User user);

    @GET("/api/users?")
    Call<UserList> doGetUserList(@Query("page") String page);

    @FormUrlEncoded
    @POST("/api/users?")
    Call<UserList> doCreateUserWithField(@Field("name") String name, @Field("job") String job);
}
```

- HTTP Requst를 요청하는 Method 들을 구현

Setting Up Retrofit Interface

REQUEST METHOD

- Retrofit은 각 HTTP 메서드에 대한 Annotation들을 제공한다
  (@GET , @POST, @PUT , @DELETE , @PATCH or @ HEAD)
- HTTP 요청을 수행하는 Method 들을 Annotation을 사용해서 정의한다
- doGetListResources(), createUesr() … : Method Name
- MultipleResource ,User … : 응답 매개 변수를 각 변수에 맵핑하는데 사용되는 응답 오브젝트 모델 POJO 클래스이다
- URL에 Query Parameter를 직접 명시할 수도 있다 `@GET("users/list?sort=desc")`

Setting Up Retrofit Interface

Url Manipulation

- Request URL은 메서드에서 파라미터,블럭 교체등을 통해 업데이트 될 수 있다. 블록은 {} 기호로 둘러싸여 있으며, 해당 문자열에 동일한 String으로 @Path Annotation 사용해야 한

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId);
```

Request Body

- @Body : POST로 요청할 때 Body에 Object를 넣어 보내는 경우 사용

  Object는 Retrofit Instance에 명시된 Converter에 의해 Convert 된다.

```
@POST("users/new")
Call<User> createUser(@Body User user);
```

Setting Up Retrofit Interface

Form Encoded And Multipart

- Method들은 전송을 위해 form-encoded 와 multipart data도 명시할 수 있다
  Method에 @FormUrlEncoded가 있으면 form-encoded 데이터가 전송된다
  각 key-value 쌍은 @Filed와 함께 쓰인다.

```
@FormUrlEncoded
@POST("user/edit")
Call<User> updateUser(@Field("first_name") String first, @Field("last_name") String last);
```

- Multipart 요청은 @Multipart와 @Part Annotation이 사용된다.

```
@Multipart
@PUT("user/photo")
Call<User> updateUser(@Part("photo") RequestBody photo, @Part("description") RequestBody description);
```

Setting Up Retrofit Interface

Header ManiPulation

- @Headers Annotation을 통해 Static header를 설정할 수 있다

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
Call<List<User>> widgetList();
```

- Request Header는 @Header Annotation을 통해 동적으로 업데이트가 가능하다

```
@GET("user")
Call<User> getUser(@Header("Authorization") String authorization);
```

4 Use Retrofit

Permission (AndroidManifest.xml)

build.gradle(app)

```
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
```

- Converter : 다양한 타입의 Response를 직렬화 시켜주는 도구.
  (우리는 Json 형태로 오는 Response를 Parsing할 것이므로 Gson을 이용)
- OkHttp 라이브러리는 Retorift 2 라이브러리에 포함된다

Dependencies (build.gradle)

AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

- INTERNET사용을 위해 INTERNET 권한을 추가한다

Use Retrofit

Retrofit

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("https://reqres.in")
            .addConverterFactory(GsonConverterFactory.create())
            .build();


    APIInterface apiInterface = retrofit.create(APIInterface.class);
    Call<MultipleResource> call = apiInterface.doGetListResources();
```

- Retrofit 객체 초기화 (BaseUrl 설정, Converter 설정)
- APIClient.getClient() 로 반환된 Retrofit Class는 APIInterface의 인터페이스를 구현한다
- apiInterface로 부터 만들어진 call 객체는 웹서버로 HTTP 요청을 만들 수 있다

Use Retrofit

Call Class

- Requst를 보내고 Response를 반환하는 Retrofit Method를 호출한다
- 각 CALL 객체는 HTTP 요청/응답 쌍을 만든다
- excute() : 동기 실행 / enqueue() 비동기 실행
- cancel()을 통해 호출을 취소할 수 있다
- https://square.github.io/retrofit/2.x/retrofit/retrofit2/Call.html

Use Retrofit

Asynchronous

```
call.enqueue(new Callback<MultipleResource>() {
    @Override
    public void onResponse(Call<MultipleResource> call, Response<MultipleResource> response) {
        Log.d("TAG",response.code()+"");
        String displayResponse = "";

        MultipleResource resource = response.body();
        Integer text = resource.page;
        Integer total = resource.total;
        Integer totalPages = resource.totalPages;
        List<MultipleResource.Datum> datumList = resource.data;
    }

    @Override
    public void onFailure(Call<MultipleResource> call, Throwable t) {
        call.cancel();
    }
});
```

- MutlipleResource resource = resonse.body() : 모델클래스를 Response에 맵핑

Use Retrofit

Callback Class

- Response를 전달한다. 주어진 Request에 대한 Respones로 하나의 Method를 호출한다
- Android : Callback 메서드는 메인(UI) 스레드에서 실행된다.
- JVM : 콜백은 요청을 수행 한후 백그라운드 스레드에서 실행된다
- onResponse() : HTTP Response를 받았을 때 호출된다. HTTP Response는 404나 505가 올 수 있으므로, Response.isSuccessful()을 통해 성공적인 응답임을 확인한다
- onFailure : 통신 중 Network Exception 발생 시, Request를 만들거나 Reponse를 처리할때 예상치 못한 Exception 발생시 호출된다.
- https://square.github.io/retrofit/2.x/retrofit/retrofit2/Callback.html

Use Retrofit

Synchronous

```
apiInterface = APIClient.getClient().create(APIInterface.class);
Call<MultipleResource> call = apiInterface.doGetListResources();
call.execute().body();
```

- 동기 호출

5 Converter

Converter

Converter

- 기본적으로 Retrofit은 HTTP Body를 OkHttp의 ResponseBody 타입으로 직렬화 해제 할 수 있으며, RequestBody 타입만 승인이 가능하다
- 다른 타입 지원을 위해 Converter 추가가 가능하다. 보편적으로 6개 라이브러리 모듈이 쓰인다

•Gson: com.squareup.retrofit2:converter-gson
•Jackson: com.squareup.retrofit2:converter-jackson
•Moshi: com.squareup.retrofit2:converter-moshi
•Protobuf: com.squareup.retrofit2:converter-protobuf
•Wire: com.squareup.retrofit2:converter-wire
•Simple XML: com.squareup.retrofit2:converter-simplexml
•JAXB: com.squareup.retrofit2:converter-jaxb
•Scalars (primitives, boxed, and String): com.squareup.retrofit2:converter-scalars

6 POJO CLASS

POJO Class

POJO CLASS

- Plain Old Java Object
- 일반적인 Java 객체
- POJO는 프로그램 가독성과 재사용성을 높이기 위해 사용됨
- POJO는 쓰고, 이해하기 쉽기 때문에 가장 많이 수용되었다
- Sun 마이크로시스템에 의해 EJB 3.0에 도입되었다
- 모든 JavaBean은 POJO 이지만, 모든 POJO가 JavaBean은 아니다. JavaBean을 구현하기 위해서는 POJO를 바탕으로 여러가지 제약이 필요하다.

```
public class Employee
{
    String name;

    public Employee(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}
```

POJO Class

MultipleResources.java

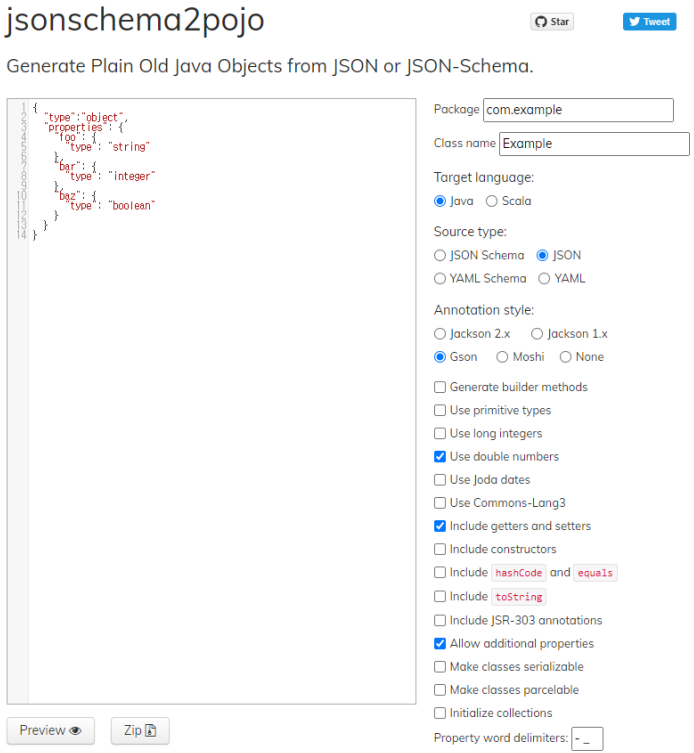
```
public class MultipleResource {

    @SerializedName("page")
    public Integer page;
    @SerializedName("per_page")
    public Integer perPage;
    @SerializedName("data")
    public List<Datum> data = new ArrayList<>();

    public class Datum {
        @SerializedName("id")
        public Integer id;
        @SerializedName("name")
        public String name;
    }
}
```

- @SerializedName Annotation은 JSON Response의 필드이름을 명시하는데 사용된다

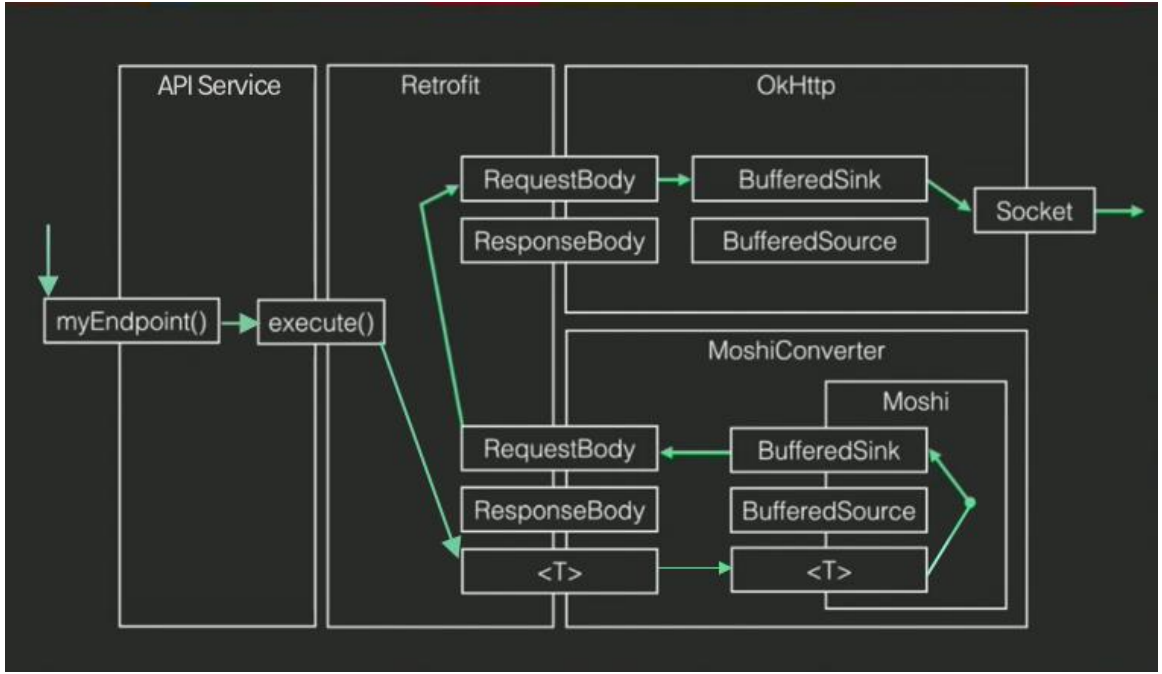
POJO Class



- http://www.jsonschema2pojo.org/
  Response에 대한 POJO 클래스를 만들기 위해 페이지를 이용할 수 있다
- JSONArray 는 List로 Serialised 된다

7 Data Flow

Data Flow : Request

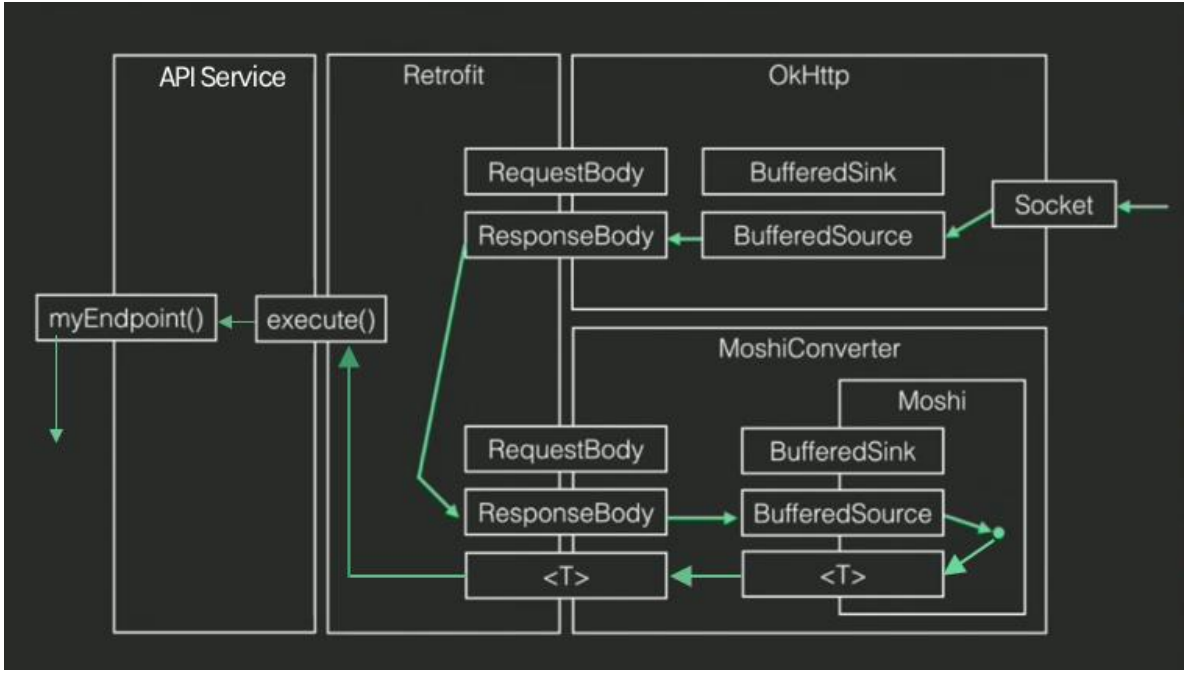


Data Flow

Mediator : Okio / Converter : Moshi

- OkHttp는 Retrofit 아래에 위치한다
- OkHttp는 HTTP Request를 위해 Socket에 연결한다
- BufferedSink / BufferedSource 는 OkHttp의 입출력 으로 보면 된다
- Okhttp/Retrofit 사이의 타입은 RequestBody/ResponseBody 다
- EndPoint 를 호출할 때 Retrofit은 Object를 Converter로 전달
- 해당 Object를 서버에 보내려고 할때, Moshi는 그것을 RequestBody에 랩핑될 BufferSink에 작성한다
- OKHttp는 Socket에 BufferSink로 전달한다

Data Flow : Response



Data Flow

Mediator : Okio / Converter : Moshi

- Socket으로부터 읽는 것도 비슷한 방식이다
- OkHttp는 응답을 Okio 타입으로 랩핑한다. 그리고 다시 ResponseBody로 랩핑한다
- Buffersouce 유형의 데이터를 가져온 다음 Converter가 데이터를 앱에 필요한 데이터 모델로 변환한다

참고

- https://galid1.tistory.com/617
- http://instructure.github.io/blog/2013/12/09/volley-vs-retrofit/
- https://github.com/HwangEunmi/Retrofit-Sample
- https://www.journaldev.com/
- https://medium.com/mindorks/understand-how-does-retrofit-work-c9e264131f4a
- https://square.github.io/retrofit/

Thanks!

Any questions ?

You can find me at

- **jeonyt89@gmail.com**