OkHttp e Retrofit

Alunos:
Adilson Angelo da Silva Junior e Lucas Vinicius da Costa Santana

Visão geral

OkHttp

- Cliente HTTP
- Suporte HTTP 2
- Criação de requisições HTTP otimizadas e robustas
- Simples e direta

Retrofit

- Parametrização de URL e queries
- Interface OO para requisições HTTP (type-safe)
- Roda em cima de OkHttp
- Diversos tipos de conversão de objetos (serialização)

OkHttp

- Cliente para requisições HTTP
- Principais features
  - Suporte a HTTP 2
  - Connection Pooling: cache de conexões para reduzir latência
  - Uso transparente de GZIP nos downloads
  - Cache de respostas para requisições repetitivas
  - Chamadas síncronas e assíncronas
  - Robustez frente a redes problemáticas
  - Interceptor

Requisitos OkHttp

- Android 5.0+ / Java 8+
- Okio (I/O e sockets)

```
compile 'com.squareup.okio:okio:2.4.1'
compile 'com.squareup.okhttp:okhttp:4.2.2'
```

Exemplo de uso do OkHttp

GET Request

Cliente

URL builder

Request builder

Chamada assíncrona de requisição

```java
public static void main(String[] args) {
    OkHttpClient client = new OkHttpClient();

    HttpUrl httpurl = new Builder()
            .scheme("http")
            .host("shibe.online")
            .addPathSegments("api/shibes")
            .addQueryParameter("count", "3")
            .addQueryParameter("httpsUrls", "true")
            .build();
    // URL: https://shibe.online/api/shibes?count=3&httpsUrls=true

    Request req = new Request.Builder()
            .get()
            .addHeader("Accept", "application/json")
            .url(httpurl)
            .build();

    client.newCall(req).enqueue(new Callback() {

        public void onResponse(Call call, Response res) throws IOException {
            if(res.isSuccessful()) {
                String response = res.body().string();
                // passar resultado pra activity
                System.out.println(response);
            }
        }

        public void onFailure(Call call, IOException e) {
            e.printStackTrace();
        }
    });
```

Exemplo de uso do OkHttp

POST request

Payload

URL builder

Request builder

```
String body = ""
        + "# Header 1\n"
        + "## Header 2\n"
        + "- list\n"
        + "- list\n";

RequestBody reqBody = RequestBody.create(body, MediaType.parse("text/x-markdown"));

httpurl = new Builder()
        .scheme("https")
        .host("api.github.com")
        .addPathSegments("markdown/raw")
        .build();

req = new Request.Builder()
        .url(httpurl)
        .post(reqBody)
        .build();
```

Client Features

Timeout

```
OkHttpClient timeoutClient = new OkHttpClient.Builder()
        .connectTimeout(10, TimeUnit.SECONDS)
        .writeTimeout(10, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        .build();
```

Client Features

Interceptor (application)

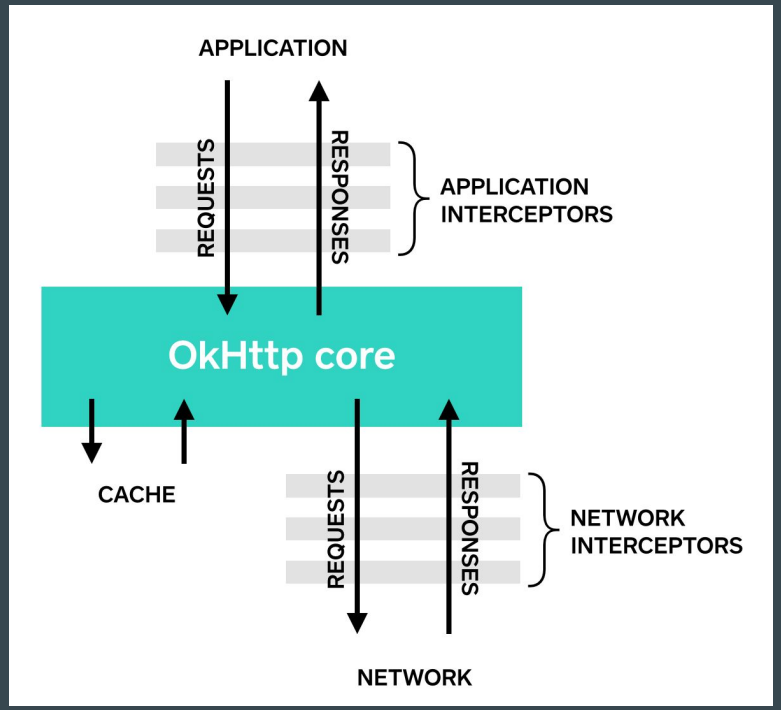
Manipular requisições http antes do envio

```
class AuthInterceptor implements Interceptor {
    public Response intercept(Chain chain) throws IOException {
        // Intercepta requisição
        Request request = chain.request();
        // Realiza procedimento com a requisição
        request.newBuilder()
        .addHeader("Authorization", "## SomeAPIAuthenticationKey ##")
        .build();
        // Envia requisição
        Response response = chain.proceed(request);
        // Retorna resposta
        return response;
    }
}
```

```
OkHttpClient interceptClient = new OkHttpClient.Builder()
        .addInterceptor(new AuthInterceptor())
        .build();
```

Client Features

Interceptor
(network)



Client Features

Authentication

Retry on 401 (Not Authorized)

```
Authenticator auth = new Authenticator() {
  public Request authenticate(Route route, Response response) throws IOException {
      if (response.request().header("Authorization") != null) {
        return null; // Give up, we've already attempted to authenticate.
      }
      String credential = Credentials.basic("jesse", "password1");
      return response.request().newBuilder()
          .header("Authorization", credential)
          .build();
    }
  };

OkHttpClient authClient = new OkHttpClient.Builder()
        .authenticator(auth)
        .build();
```

Retrofit

- Client HTTP "*type-safe*" para Android e Java.
- Com ele as APIs são interfaces.
- Retrofit é uma abstração de alto nível para REST construída com base em OkHttp.

Requisitos Retrofit

- Android 5.0+ / Java 8+
- Okio

```
compile 'com.squareup.okio:okio:2.4.1'
compile 'com.squareup.okhttp:okhttp:4.2.2'
compile 'com.squareup.retrofit2:retrofit:2.6.2'
```

- Você provavelmente vai precisar de um converter, por exemplo Moshi.

```
compile 'com.squareup.moshi:moshi:1.5.0'
compile 'com.squareup.retrofit2:converter-moshi:2.3.0'
```

Serviços

Podemos considerar um serviço como a declaração de vários endpoints relacionados de uma API.
Exemplo para github:

```java
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);

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

  @GET("group/{id}/users")
  Call<List<User>> groupList(@Path("id") int groupId, @QueryMap Map<String, String> options);

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

Serviços

Mais alguns exemplos:

```
13
14    @FormUrlEncoded
15    @POST("user/edit")
16    Call<User> updateUser(@Field("first_name") String first, @Field("last_name") String last);
17
18    @Headers({
19      "Accept: application/vnd.github.v3.full+json",
20      "User-Agent: Retrofit-Sample-App"
21    })
22    @GET("users/{username}")
23    Call<User> getUser(@Path("username") String username);
24
25    @GET("user")
26    Call<User> getUser(@Header("Authorization") String authorization);
27
28    @GET("user")
29    Call<User> getUser(@HeaderMap Map<String, String> headers);
30  }
```

Converters

- Por padrão, Retrofit só consegue desserializar os “bodies” em ResponseBody (de OkHttp) e só aceita variáveis tipo RequestBody anotadas com @Body.
- Converters são usados para suportar outros tipos. Algumas bibliotecas de serialização populares já foram adaptadas e podem ser utilizadas:
  - 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`
  - Scalars (primitives, boxed, and String): `com.squareup.retrofit2:converter-scalars`

Custom Converters

- Caso precise se comunicar com uma API que usa formato diferente dos suportados por Retrofit, basta criar uma classe que estende a classe *Converter.Factory* (e consequentemente implementar “alguns” converters).

```
1   public interface Converter<F, T> {
2     @Nullable T convert(F value) throws IOException;
3
4     abstract class Factory {
5       public @Nullable Converter<ResponseBody, ?> responseBodyConverter(Type type,
6           Annotation[] annotations, Retrofit retrofit) {
7         return null;
8       }
9
10      public @Nullable Converter<?, RequestBody> requestBodyConverter(Type type,
11          Annotation[] parameterAnnotations, Annotation[] methodAnnotations, Retrofit retrofit) {
12        return null;
13      }
14
15      public @Nullable Converter<?, String> stringConverter(Type type, Annotation[] annotations,
16          Retrofit retrofit) {
17        return null;
18      }
19    }
20  }
```

Retrofit Configuration

- Retrofit cria uma implementação para a interface do serviço.
- As instâncias de Call só podem ser executadas uma única vez, mas o método *clone()* cria uma nova instância.
- Em Android, os callbacks são executados na "*main thread*".
- Importante prestar atenção na indicação de converter factories utilizadas e da baseUrl.

```
1   Retrofit retrofit = new Retrofit.Builder()
2     .baseUrl("https://api.github.com/")
3     .addConverterFactory(MoshiConverterFactory.create())
4     .build();
5
6   GitHubService gitHubService = retrofit.create(GitHubService.class);
7
8   Call<List<Repo>> callListRepo = gitHubService.listRepos("luucasv");
9
```

Observação

- **Atenção!!! Resolução de URL não é trivial em Retrofit 2.0.**
  - Considere:
    - URL base: aquela definida ao inicializar o cliente do Retrofit.
    - Caminho relativo: aquele definido nas Annotations dos métodos do serviço.
  - Se o caminho relativo começa com '/', Retrofit vai usar somente o hostname da URL base.
  - Se o caminho relativo não começar com '/', Retrofit vai usar a URL base até o último '/'.

  Exemplo:
  - www.exemplo.com/path1 + path2 = www.exemplo.com/path2
  - www.exemplo.com/path1/ + path2 = www.exemplo.com/path1/path2
  - www.exemplo.com/path1/ + /path2 = www.exemplo.com/path2
  - www.exemplo.com + path2 = www.exemplo.com/path2

- **Nota: Se o "caminho relativo" for uma "URL completa" a URL base é ignorada.**

Observação

- Recomendação: Fazer com que a URL base seja apenas o hostname assim não importa se o caminho relativo começa com ‘/’ ou não.

Obrigado!

- O que acha? OkHttp ou Retrofit?
- Dúvidas?

Ah, lembramos de falar que tudo apresentado aqui é open source e desenvolvido pela Square?

Referências & Documentação
OkHttp: https://square.github.io/okhttp/
Retrofit: https://square.github.io/retrofit/