08

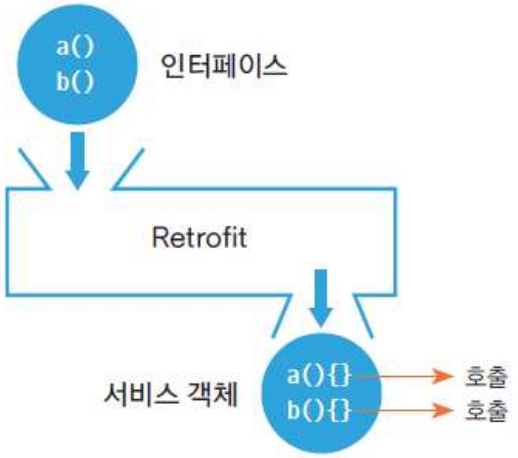
08-3. Retrofit API

네트워크 프로그래밍

18-2 HTTP 통신하기

Retrofit 라이브러리

- Retrofit*은 스퀘어에서 만든 HTTP 통신을 간편하게 만들어 주는 라이브러리
- Retrofit은 네트워크 통신 정보만 주면 그대로 네트워크 프로그래밍을 대신 구현
- Retrofit 동작 방식
  1. 통신용 함수를 선언한 인터페이스를 작성합니다.
  2. Retrofit에 인터페이스를 전달합니다.
  3. Retrofit이 통신용 서비스 객체를 반환합니다.
  4. 서비스의 통신용 함수를 호출한 후 Call 객체를 반환합니다.
  5. Call 객체의 enqueue() 함수를 호출하여 네티워크 통신을 수행합니다.



18-2 HTTP 통신하기

- 라이브러리 선언
  - Retrofit은 JSON이나 XML 데이터를 모델(VO 클래스) 객체로 변환
  - JSON, XML을 파싱하는 라이브러리가 필요
  - 파싱 라이브러리에 맞는 converter 라이브러리 필요
    - 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

• Retrofit2 사용 등록

```
implementation("com.google.code.gson:gson:2.10.1")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
implementation("com.squareup.retrofit2:retrofit:2.9.0")
```

18-2 HTTP 통신하기

- 모델 클래스 선언
  - 모델 클래스의 프로퍼티에 데이터가 자동으로 저장되는 기본 규칙은 데이터의 키와 프로퍼티 이름을 매칭
  - 만약 키와 프로퍼티 이름이 다를 때는 @SerializedName이라는 애너테이션으로 명시

```
{
    "id": 7,
    "email": "michael.lawson@reqres.in",
    "first_name": "Michael",
    "last_name": "Lawson",
    "avatar": "https://reqres.in/img/faces/7-image.jpg"
}
```

• 모델 클래스

```
data class UserModel(
    var id: String,
    @SerializedName("first_name")
    var firstName: String,
    // @SerializedName("last_name")
    var lastName: String,
    var avatar: String,
    var avatarBitmap: Bitmap
)
```

18-2 HTTP 통신하기

- 서비스 인터페이스 정의
  - @GET은 서버와 연동할 때 GET 방식으로 해달라는 의미
  - @Query는 서버에 전달되는 데이터
  - @Url은 요청 URL

• 서비스 인터페이스 정의

```
interface INetworkService {
    @GET("api/users")
    fun doGetUserList(@Query("page") page: String): Call<UserListModel>
    @GET
    fun getAvatarImage(@Url url: String): Call<ResponseBody>
}
```

18-2 HTTP 통신하기

- Retrofit 객체 생성
  - baseUrl() 함수로 URL을 설정
  - addConverterFactory() 함수로 데이터를 파싱해 모델 객체에 담는 역할자를 지정

• Retrofit 객체 생성

```
val retrofit: Retrofit
    get() = Retrofit.Builder()
        .baseUrl("https://reqres.in/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
```

18-2 HTTP 통신하기

- 인터페이스 타입의 서비스 객체 얻기
  - Retrofit의 create() 함수에 앞에서 만든 서비스 인터페이스 타입을 전달

• 서비스 객체 얻기

```
var networkService: INetworkService = retrofit.create(INetworkService::class.java)
```

18-2 HTTP 통신하기

- 네트워크 통신 시도
  - 터페이스의 함수를 호출하면 네트워크 통신을 시도
  - Call 객체가 반환
  - Call 객체의 enqueue() 함수를 호출하는 순간 네트워킹

• Call 객체 얻기

```
val userListCall = networkService.doGetUserList("1")
```

• 네트워크 통신 수행

```
userListCall.enqueue(object : Callback<UserListModel> {
    override fun onResponse(call: Call<UserListModel>,
                            response: Response<UserListModel>) {
        val userList = response.body()
        (... 생략 ...)
    }
    override fun onFailure(call: Call<UserListModel>, t: Throwable) {
        call.cancel()
    }
})
```

18-2 HTTP 통신하기

Retrofit 애너테이션

- @GET, @POST, @PUT, @DELETE, @HEAD
  - HTTP 메서드를 정의하는 애너테이션

• HTTP 메서드 애너테이션

• 인터페이스에 선언한 함수

```
@GET("users/list?sort=desc")
fun test1(): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test1()
```

• 최종 서버 요청 URL

```
https://reqres.in/users/list?sort=desc
```

18-2 HTTP 통신하기

- @Path
  - URL의 경로를 동적으로 지정

• 동적인 경로 애너테이션

• 인터페이스에 선언한 함수

```
@GET("group/{id}/users/{name}")
fun test2(
    @Path("id") userId: String,
    @Path("name") arg2: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test2("10", "kkang")
```

• 최종 서버 요청 URL

```
https://reqres.in/group/10/users/kkang
```

18-2 HTTP 통신하기

- @Query
  - 함수의 매개변숫값을 서버에 전달

• 질의 애너테이션 예

```
• 인터페이스에 선언한 함수
@GET("group/users")
fun test3(
    @Query("sort") arg1: String,
    @Query("name") arg2: String
): Call<UserModel>

• Call 객체를 얻는 구문
val call: Call<UserModel> = networkService.test3("age", "kkang")

• 최종 서버 요청 URL
https://reqres.in/group/users?sort=age&name=kkang
```

18-2 HTTP 통신하기

- @QueryMap
  - 서버에 전송할 데이터를 Map 타입의 매개변수 처리

• 질의 맵 애너테이션 예

• 인터페이스에 선언한 함수

```
@GET("group/users")
fun test4(
    @QueryMap options: Map<String, String>,
    @Query("name") name: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test4(
    mapOf<String, String>("one" to "hello", "two" to "world"),
    "kkang"
)
```

• 최종 서버 요청 URL

```
https://reqres.in/group/users?one=hello&two=world&name=kkang
```

18-2 HTTP 통신하기

- @Body
  - 서버에 전송할 데이터를 모델 객체로 지정

• 모델 객체 애너테이션 예

• 인터페이스에 선언한 함수

```
@POST("group/users")
fun test5(
    @Body user: UserModel,
    @Query("name") name: String
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test5(
    UserModel(id="1", firstName = "gildong", lastName = "hong", avatar = "someurl"),
    "kkang"
)
```

• 최종 서버 요청 URL

https://reqres.in/group/users?name=kkang

• 서버에 스트림으로 전송되는 데이터

{"id":"1","first_name":"gidong","last_name":"hong", "avatar":"someurl"}

18-2 HTTP 통신하기

- @FormUrlEncoded와 @Field
  - @FormUrlEncoded 애너테이션은 데이터를 URL 인코딩 형태로 만들어 전송
  - @Field 애너테이션이 추가된 데이터를 인코딩해서 전송

• URL 인코딩 애너테이션 예

• 인터페이스에 선언한 함수

```
@FormUrlEncoded
@POST("user/edit")
fun test6(
    @Field("first_name") first: String?,
    @Field("last_name") last: String?,
    @Query("name") name: String?
): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call: Call<UserModel> = networkService.test6(
    "gildong 길동",
    "hong 홍",
    "kkang"
)
```

• 최종 서버 요청 URL

```
https://reqres.in/user/edit?name=kkang
```

• 서버에 스트림으로 전송되는 데이터

```
first_name=gildong%20%EA%B8%B8%EB%8F%99&last_name=hong%20%ED%99%8D
```

18-2 HTTP 통신하기

- 배열이나 List 객체에 @Field 애너테이션을 사용하면 데이터 여러 건을 같은 키로 서버에 전달

• 리스트에 필드 애너테이션 사용 예

• 인터페이스에 선언한 함수

```
@FormUrlEncoded
@POST("tasks")
fun test7(@Field("title") titles: List<String>): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val list: MutableList<String> = ArrayList()
list.add("홍길동")
list.add("류현진")
val call = networkService.test7(list)
```

• 최종 서버 요청 URL

```
https://reqres.in/tasks
```

• 서버에 스트림으로 전송되는 데이터

```
title=%ED%99%8D%EA%B8%B8%EB%8F%99&title=%EB%A5%98%ED%98%84%EC%A7%84
```

18-2 HTTP 통신하기

- @Header
  - 서버 요청에서 헤더값을 조정

• 헤더 애너테이션 예

• 인터페이스에 선언한 함수

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
fun test8(): Call<UserModel>
```

- @Url
  - baseUrl을 무시하고 전혀 다른 URL을 지정

• URl 애너테이션 예

• 인터페이스에 선언한 함수

```
@GET
fun test9(@Url url: String, @Query("name") name: String): Call<UserModel>
```

• Call 객체를 얻는 구문

```
val call = networkService.test9("http://www.google.com", "kkang")
```

• 최종 서버 요청 URL

```
http://www.google.com/?name=kkang
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare