25장. HTTP 통신

25.1 Java API를 이용한 HTTP 통신

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

- API Level 28(Android 9)부터는 보안 강화 목적으로 앱에서의 HTTP 통신이 기본으로 금지
- HTTP를 계속 이용하겠다면 별도로 설정 필요

```
<?xml version="1.0" encoding="utf-8"?> <network-security-config> <domain-config
cleartextTrafficPermitted="true"> <domain includeSubdomains="true">xxx.xxx.xxx.xxx</domain> </domain-config>
</network-security-config>
```

```
<application … android:networkSecurityConfig="@xml/network_security_config">
```

- 아래의 방법도 가능

```
<application … android:usesCleartextTraffic="true">
```

25.1 Java API를 이용한 HTTP 통신

- 웹 서버와 연결하려면 URL 클래스와 HttpURLConnection 클래스를 이용

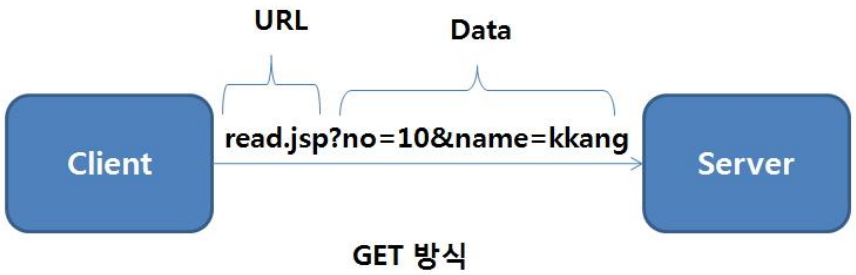
```
URL text = new URL(url);
HttpURLConnection http = (HttpURLConnection) text.openConnection();
```

HttpURLConnection 클래스
- setConnectTimeput(): 연결 타임아웃 시간 지정
- setReadTimeout(): 읽기 타임아웃 시간 지정
- setUseCaches(): 캐시 사용 여부 지정
- setDoInput(): 데이터를 읽을 것인지 지정
- setDoOutput(): 데이터를 요청 몸체에 쓸 것인지 지정
- setRequestProperty(): 요청 헤더 지정
- setRequestMethod(): HTTP 요청 Method 지정

```
http.setRequestProperty("Content-type", "application/x-www-form-                    urlencoded;charset=UTF-8");
http.setConnectTimeout(10 * 1000);
http.setReadTimeout(10 * 1000);
http.setRequestMethod("POST");
http.setDoInput(true);
http.setDoOutput(true);
```

25.1 Java API를 이용한 HTTP 통신

- 서버에 데이터 전송



- GET 방식으로 전송하면 HTTP 요청 헤더의 URL 뒤에 붙어서 서버에 전송
- POST 방식으로 데이터를 전송하려면 setRequestMethod( ) 함수를 이용하여 POST 방식을 사용하겠다고 선언해야 하며, 정식의 IO 객체를 이용해 전송

```
PrintWriter writer = new PrintWriter(new OutputStreamWriter( http.getOutputStream(), "UTF-8"));
writer.write(postData);
writer.flush();
```

25.1 Java API를 이용한 HTTP 통신

- 데이터를 수신

```
BufferedReader in = new BufferedReader(new                InputStreamReader( http.getInputStream(), "UTF-8"));
StringBuffer sb = new StringBuffer();
String inputLine;
while ((inputLine = in.readLine()) != null) {
   sb.append(inputLine);
}
response = sb.toString();
```

- 이미지 수신

```
Bitmap bitmap = BitmapFactory.decodeStream(http.getInputStream());
```

Step by Step 실습 25-1 – HTTP 통신

- 모듈 생성
- 파일 복사
- AndroidManifest 파일 작업
- HttpRequester.java 작성
- HttpImageRequester.java 작성
- MainActivity 작성
- 서버준비
- 실행

25.2 Volley API를 이용한 HTTP 통신

- Volley는 2013년 구글 IO 행사에서 공개된 API

```
implementation 'com.android.volley:volley:1.1.1'
```

- RequestQueue: 서버 요청자. 다른 Request 클래스들의 정보대로 서버에 요청을 보내는 역할
- StringRequest: 문자열을 결과로 받는 요청 정보
- ImageRequest: 이미지를 결과로 받는 요청 정보
- JsonObjectRequest: JSONObject를 결과로 받는 요청 정보
- JsonArrayRequest: JSONArray를 결과로 받는 요청 정보

```
StringRequest stringRequest = new StringRequest(...);

RequestQueue queue = Volley.newRequestQueue(this);
queue.add(stringRequest);
```

25.2 Volley API를 이용한 HTTP 통신

- StringRequest는 서버로부터 문자열 데이터를 얻을 목적

```
StringRequest stringRequest = new StringRequest(Request.Method.GET, url, new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        //결과 처리
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {

    }
});
```

- 클라이언트 데이터를 서버에 전송

```
StringRequest stringRequest = new StringRequest(...){
    protected Map<String, String> getParams() throws                    com.android.volley.AuthFailureError {
        Map<String, String> params = new HashMap<String, String>();
        params.put("a1", "kkang");
        return params;
    }
};
```

25.2 Volley API를 이용한 HTTP 통신

- JSON 데이터를 목적으로 JsonObjectRequest와 JsonArrayRequest을 제공

```
JsonObjectRequest jsonRequest = new JsonObjectRequest(Request.Method.GET, url,                    null, new
Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
        //서버 응답 후 사후처리
        //JSONObject에서 데이터 획득
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
    }
});
```

- 이미지 연동

```
ImageRequest imageRequest=new ImageRequest (url, new Response.Listener<Bitmap>() {
    @Override
    public void onResponse(Bitmap response) {
        //결과 처리
    }
    //maxWidth Maximum width 등의 사이즈 정보 설정 가능. 0은 설정 안하겠다는의미},0,0,
ImageView.ScaleType.CENTER_CROP,null, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {

    }
});
```

25.2 Volley API를 이용한 HTTP 통신

- 서버의 이미지 화면에 출력

```
<com.android.volley.toolbox.NetworkImageView    //.../>
```

```
ImageLoader imageLoader = new ImageLoader(queue, new ImageLoader.ImageCache() {
    public void putBitmap(String url, Bitmap bitmap) {
    }

    public Bitmap getBitmap(String url) {
        return null;
    }
});
//이미지 요청
imageView = (NetworkImageView) findViewById(R.id.lab2_image);
imageView.setImageUrl(url, imageLoader);
```

Step by Step 25-2 – Volley API

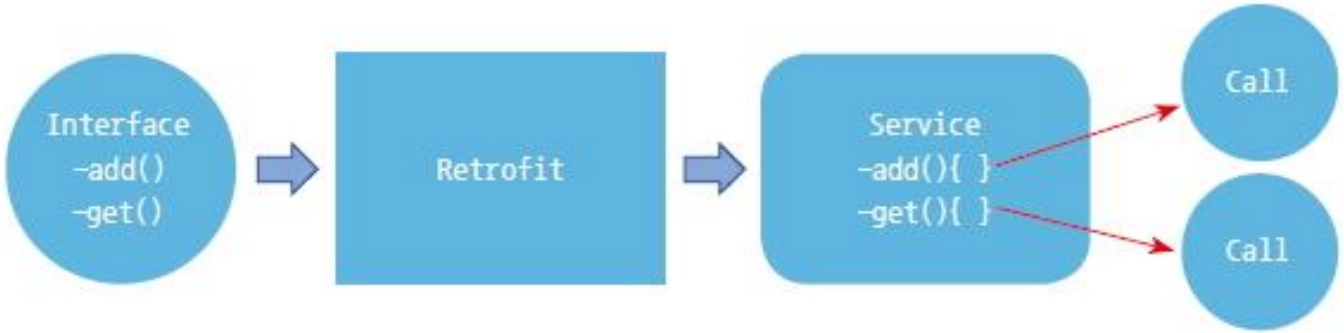
- 액티비티 생성
- 파일 복사
- 그레이들 설정
- activity_lab25_2.xml 작성
- Lab25_2Activity 작성
- 실행

25.3 Retrofit2

- HTTP 통신을 도와주는 유명한 라이브러리

25.3.1 Retrofit 이용 준비 및 구조

Retrofit을 이용해 서버 연동을 하려면 기본 Call 객체가 필요.
Call 객체는 실제 서버 연동을 실행하는 객체
그런데 이 Call 객체를 개발자가 직접 만들지 않고 Retrofit이 자동으로 만들어 준다는 개념



25.3 Retrofit2

Retrofit을 이용하면서 JSON, XML 데이터를 VO 객체로 변환해주는 외부 라이브러리 연동을 지원

- 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

```
implementation 'com.squareup.retrofit2:retrofit:2.5.0'
implementation 'com.google.code.gson:gson:2.8.5'
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
```

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

25.3 Retrofit2

25.3.2. Retrofit을 이용한 HTTP 통신

Model 정의
Model은 서버 연동을 위한 데이터 추상화 클래스

```
public class ItemModel {
public long id;
public String author;
public String title;
public String description;
public String urlToImage;
public String publishedAt;
}
```

컨버터는 변수명과 파싱된 데이터의 키값을 매핑해서 데이터를 저장
JSON의 키값과 변수명이 다를 때는 @SerializedName이라는 어노테이션 이용

```
public class ItemModel {
…
@SerializedName("publishedAt")
public String published_at;
}
```

25.3 Retrofit2

Retrofit 객체 생성

```
public class RetrofitHelper {
static Retrofit getRetrofit() {
return new Retrofit.Builder()
.baseUrl("https://~~~~~~~~")
.addConverterFactory(GsonConverterFactory.create())
.build();
}
}
```

Service 인터페이스

서버 네트워킹을 위한 함수를 인터페이스의 추상 함수로 만들고 그 함수에 어노테이션으로 GET/POST 등의 HTTP method 지정, 서버 전송 질의 지정 등을 하면 그 정보에 맞게 서버를 연동할 수 있는 Call 객체를 자동으로 만들어 주는 구조

```
public interface RetrofitService {

@GET("/v2/everything")
Call<PageListModel> getList(@Query("q") String q,
@Query("apiKey") String apiKey,
@Query("page") long page,
@Query("pageSize") int pageSize);
}
```

25.3 Retrofit2

Call 객체 획득

```
networkService = RetrofitHelper.getRetrofit().create(RetrofitService.class);
Call<PageListModel> call = networkService.getList(QUERY, API_KEY, 1, 2)
```

네트워킹 시도

```
call.enqueue(new Callback<PageListModel>() {
@Override
public void onResponse(Call<PageListModel> call, Response<PageListModel> response) {
if(response.isSuccessful()) {
//…
}
}
@Override
public void onFailure(Call<PageListModel> call, Throwable t) {
}
});
```

25.3 Retrofit2

25.3.3. Retrofit 어노테이션

- @GET, @POST, @PUT, @DELETE, @HEAD

HTTP Method를 지정할 때 사용

- @Path : URL의 Path 부분 동적 할당

① 인터페이스의 함수 선언

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

② 함수 호출

```
Call<List<UserModel>> test1 = testService.groupList(10);
```

③ URL

https://~~~~/group/10/users

25.3 Retrofit2

25.3.3. Retrofit 어노테이션

- @GET, @POST, @PUT, @DELETE, @HEAD

HTTP Method를 지정할 때 사용

- @Path : URL의 Path 부분 동적 할당

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

- @Query : 질의 문자열로 지정해야 하는 데이터 명시

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

- @QueryMap : Map 객체로 질의 데이터 지정

```
@GET("group/{id}/users")
Call<List<UserModel>> groupList2(@Path("id") int groupId,
@QueryMap Map<String, String> options,
@Query("name") String name);
```

25.3 Retrofit2

- @Body : 객체를 request body 에 포함(POST 방식에서 사용)

```
@POST("group/{id}/users")
Call<UserModel> groupList3(@Path("id") int groupId,
@Body UserModel user,
@Query("name") String name);
```

- @FormUrlEncoded : Form-encoded data(POST 방식에서 사용)

```
@FormUrlEncoded
@POST("user/edit")
Call<UserModel> groupList4(@Field("first_name") String first,
@Field("last_name") String last,
@Query("name") String name);
```

- @Headers : HTTP 요청 헤더 지정

```
@Headers("Cache-Control: max-age=640000")
@GET("widget/list")
Call<List<UserModel>> groupList7();
```

- @Url : base URL과 상관없는 특정 URL 지정

```
@GET
Call<UserModel> list(@Url String url);
```

Step by Step 25-3 – Retrofit

- 그레이들 설정
- AndroidManifest.xml 작업
- 서버 키 획득
- Model 클래스 작성
- RetrofitServcie 작성
- RetrofitFactory 작성
- Lab25_3Activity 작성
- 실행