

Lab12

API

本節目的：

- 了解 Http Get 與 Post 觀念
- 了解 JSON 觀念
- 使用 GSON 轉換 JSON 與 String。
- 了解透過 OkHttpClient 來進行網路請求。

12.1 觀念說明：

在 Android 中想要透過網路取得資料，或是與他人交換訊息，就需要使用 Http 通訊機制，Http 通訊協定被廣泛應用於網路環境，除了常見的瀏覽器外，Android 應用程式也能使用 Http 協定存取伺服器的資料。

12.1.1 Http 觀念

Http(Hypertext Transfer Protocol)是一種網路通訊協定，用於客戶端發出請求(Request)給伺服器，伺服器將要給予的資料回傳(Response)給客戶端使用，一般我們稱之為 API(Application Programming Interface)，如下圖 1 手機送出 Request 與伺服器建立連線，伺服器回傳 Response 資料，資料通常為 Xml 或 JSON...等格式的字串，需要再透過翻譯器轉換成 Android 看得懂的資料型態。

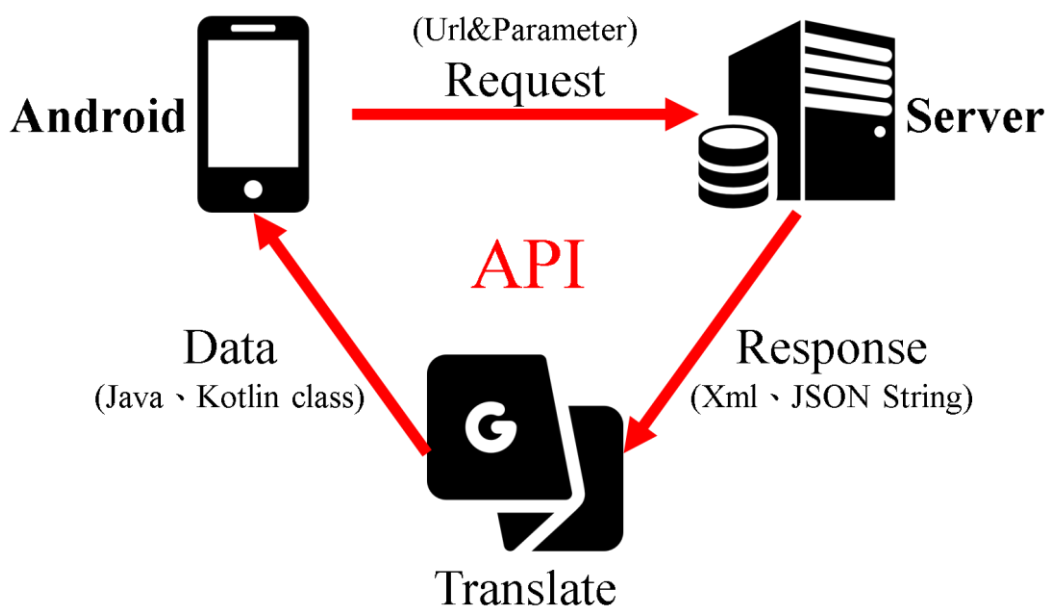


圖 1、API 示意圖

Http 通訊協定中常用的方法(HTTP methods)有 Http Get 與 Http Post 兩種，用信件來比喻的話，信封的格式就是 HTTP，信封外的內容為 http-header，信封內的書信為 message-body，那麼 HTTP Method 就是你要告訴郵差的寄信規則，Get 就像廣告信，參數直接附加於網址後面，人人都看得到，而 Post 就是私人信件，裡面的參數只有寄信者與收信者能看到，相對的安全性也較高。

● Http Get

當我們要從網路取得資料時，就需要從客戶端發出 Http Get Request 跟伺服器建立連線，Get Request 請求的參數會附在 URL 之後(就是直接加在網址後面)，網址以「?」分割 URL 和傳輸參數，參數之間以「&」相連，如下網址所示 name1 和 name2 是參數的名稱，value1 和 value2 是參數的值，傳送的連結如下，黑字的部分為網址，「?」前是 URL，「?」後是要傳輸的參數.多個在 and 之後便可加上我們的參數，並以「&」做區別：

```
http://網址?name1=value1&name2=value2
```

Get 可以直接使用瀏覽器顯示。圖 2 我們以瀏覽器測試以下網址，從伺服器撈出 postId 為 1 的所有文章資料：

```
https://jsonplaceholder.typicode.com/comments?postId=1
```

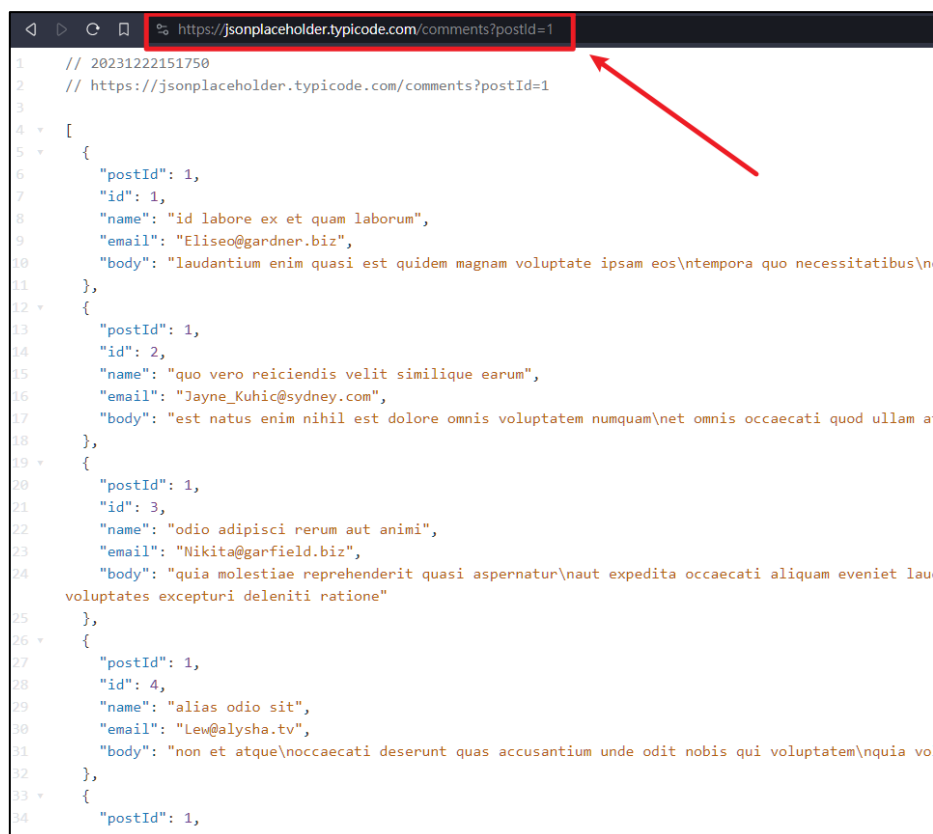


圖 2、Http Get 示範

執行之後，瀏覽器的顯示區會出現一組字串，此字串即為伺服器回傳的資料，並採用 JSON 字串格式表示，有關 JSON 字串格式會在下一節做說明。

Get 使用非常簡單方便，不過也有其缺點：

- 由於 Get 是直接將參數加在網址後面，任何人都能看見。因此不該使用 Get Request 傳送重要的資料。
- Get Request 有長度限制。

● Http Post

不同於 Get 直接在網址上寫上要傳送的資料，Post 將要傳送給伺服器的資料用 body 另外包起來，隨著 request 一起送給伺服器，如下圖 3 採用 <https://reqbin.com> 網頁來測試 Post。

圖 3 例子中，採用 Post 對 <https://tools-api.italkutalk.com/java/lab12/test> 做請求，搜尋的參數有 "name" 為一字串，id 為一整數型態的資料。

The screenshot displays the 'Online REST & SOAP API Testing Tool' interface. It includes a description of the tool and a form for configuring an API request. The form has five numbered annotations in red:

- 1. 添加目標請求的 URL**: Points to the URL input field containing `https://tools-api.italkutalk.com/java/lab12/test`.
- 2. 選擇參數為 JSON 格式**: Points to the 'Content' dropdown menu, which is set to `JSON (application/json)`.
- 3. 輸入 Request 的參數**: Points to the request body input area containing a JSON object:

```
{  "id": 111222333 ,  "name": "胖先生"}
```
- 4. 選擇 POST 方法**: Points to the 'POST' dropdown menu.
- 5. 送出請求**: Points to the 'Send' button.

Other visible elements include tabs for 'Content (4)', 'Authorization', 'Headers', and 'Raw (9)', and a 'Send' button.

圖 3、Http Post Request

執行之後，伺服器可以接收到客戶端發送過去的資料，經由伺服器執行相關處理後會 Response 一組 JSON 字串資料，如下圖 4 所示 id 為「111222333」且 result 帶有「Hello 胖先生」的資料。



圖 4、Response

說明

理解了 Get 與 Post 的觀念之後，本教學中會教導在 Android 上採用 JSON 字串格式，透過 OkHttp 來與後台溝通。

12.1.2 JSON 觀念

HTTP 的操作上，傳遞的資料必須是字串形式，因此為了讓資料能透過單一字串送出多組的資料，就必須要設計一套標準化的格式，此教學中會教導使用的是 JSON 字串格式。


JSON 是個以純文字為基底去儲存和傳送簡單結構資料，你可以透過特定的格式去儲存任何資料(字串、數字、陣列、物件)，也可以透過物件或陣列來傳送較複雜的資料。

JSON 字串採用 name : value 的表示方式，name 表示這個變數的名稱，而 value 表示其內容物件或陣列的 value 值的表示方式如下：

整數或浮點數	直接寫入數值
字串	加上""
布林函數 (boolean)	true 或 false
陣列	加上[]
物件	加上{ }

如下方所示，左邊的類別結構以 JSON 來表示後，會呈現圖 5 的結果，變數名稱以字串的方式表示，並以冒號連接變數內容，陣列物件會以一個大括弧包含所有成員，而布林、整數與字串則是直接顯示數值。

```
int[] myArray = {1, 2, 3};
boolean myBoolean = true;
int myNumber = 123;
String myString = "abc";
String[] myStringArray = {"a",
"b", "c"};
```



```
1 {
2   "myArray": [
3     1,
4     2,
5     3
6   ],
7   "myBoolean": true,
8   "myNumber": 123,
9   "myString": "abc",
10  "myStringArray": [
11    "a",
12    "b",
13    "c"
14  ]
15 }
```

圖 5、JSON 資料結構

12.1.3 GSON

圖 1 手機透過 API 與伺服器溝通,由於 Http 連線存在資料格式的問題,客戶端看不懂伺服器的資料型態,而伺服器看不懂客戶端的類別,因此在送出 Http Request 前,我們需要將程式的物件轉成 JSON 字串,這動作叫做序列化(Serialization),而接收 Http Response 後,需要將 JSON 字串轉成程式的物件,這動作叫做反序列化(Deserialization)。Google 提供一個開源 library **GSON** 可以快速的處理物件與 JSON 格式轉換。

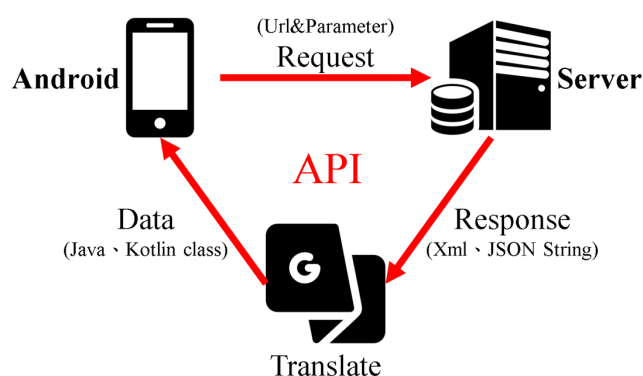


圖 1、API 示意圖

要使用 GSON，首先我們要引入 GSON 的 library，下圖 6 在 `libs.versions.toml` 中，我們加入 `gson = { group = "com.google.code.gson", name = "gson", version.ref = "gson" }` 後讓系統將其匯入。

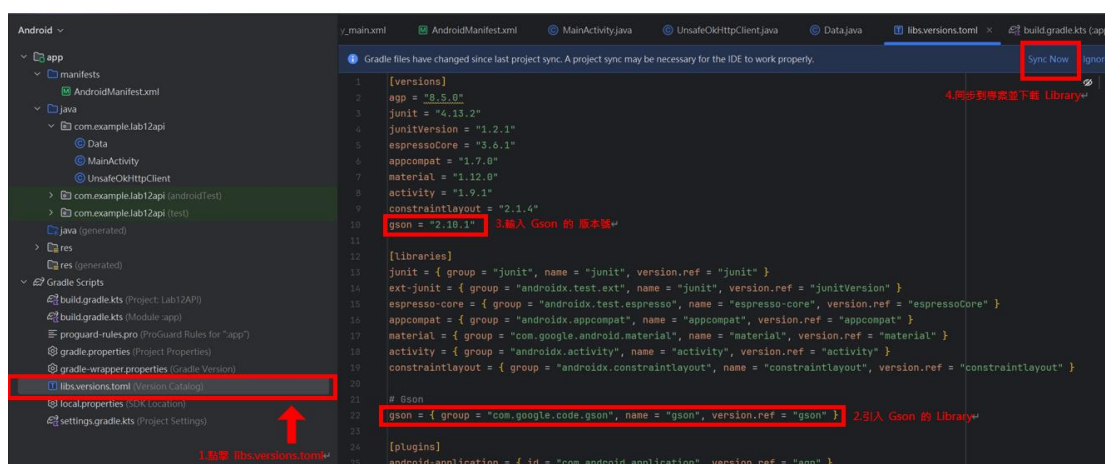


圖 6、加入 GSON library

下圖 7 在 gradle 的 dependencies 中，我們加入 implementation(libs.okhttp) 後讓系統將其匯入。

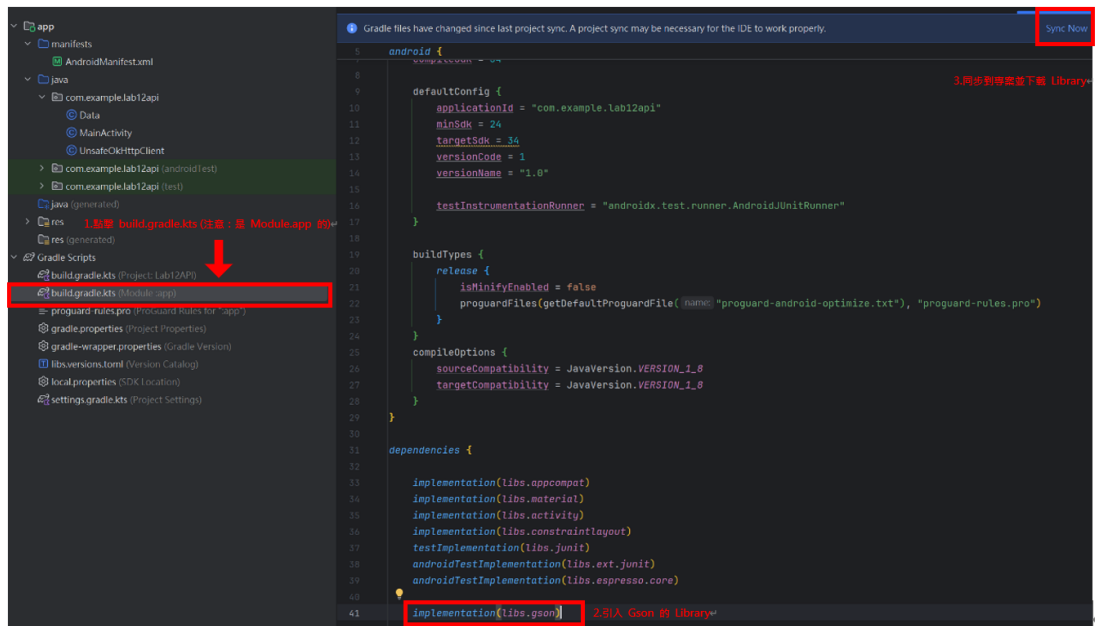


圖 7 加入 Gson 的 dependencies

● 序列化(把物件轉成 JSON 字串)

- 1) 在轉換之前，我們要先設計好一個要被轉換的 Data 類別，來接收來自伺服器的資料：

```
class Data{
    int myNumber;
    String myString;
}
```

- 2) 要透過 GSON 把該物件轉成 JSON 字串有兩個步驟：

```
Step1：建立一個物件，放入資料至物件中
Data data = new Data();
data.myNumber = 123;
data.myString = "abc";
Step2：使用 Gson().toJson()把物件轉成 JSON 字串
String json = new Gson().toJson(data);
```

- 3) 輸出字串後可以看到圖 8 的字串結果：

```
json: {"myNumber":123,"myString":"abc"}
```

圖 8、JSON 字串

● 反序列化(把 JSON 字串轉成物件)

1) 要透過 GSON 把 JSON 字串轉成物件有兩個步驟：

Step1：準備一個 JSON 字串

```
String json = "{ \"myNumber\":456, \"myString\": \"efg\" }";
```

Step2：使用 Gson().fromJson()把 json 字串以 Data 格式做轉換並輸出

```
Data data = new Gson().fromJson(json, Data.class);
```

2) 把 Data 物件中 myNumber 與 myString 輸出後的結果如下圖 9：

```
myNumber: 456
myString: efg
```

圖 9、JSON 資料

12.1.4 OkHttp

OkHttp 是 Square 出產的一個 Open source project，是一個 Http 連線的第三方 library，使用它快速實作 Http Get/Post 資料交換的工作，讓 HTTP 連線的過程更加有效率，能避免人為的錯誤設計，加快程式執行的速度。

圖 10 加入 OKHttp library 要使用 OkHttp，首先我們要引入 OkHttp 的 library，在 libs.versions.toml 中，我們加入 okhttp = { group = "com.squareup.okhttp3", name = "okhttp", version.ref = "okhttp" }，然後同步讓系統將其匯入。

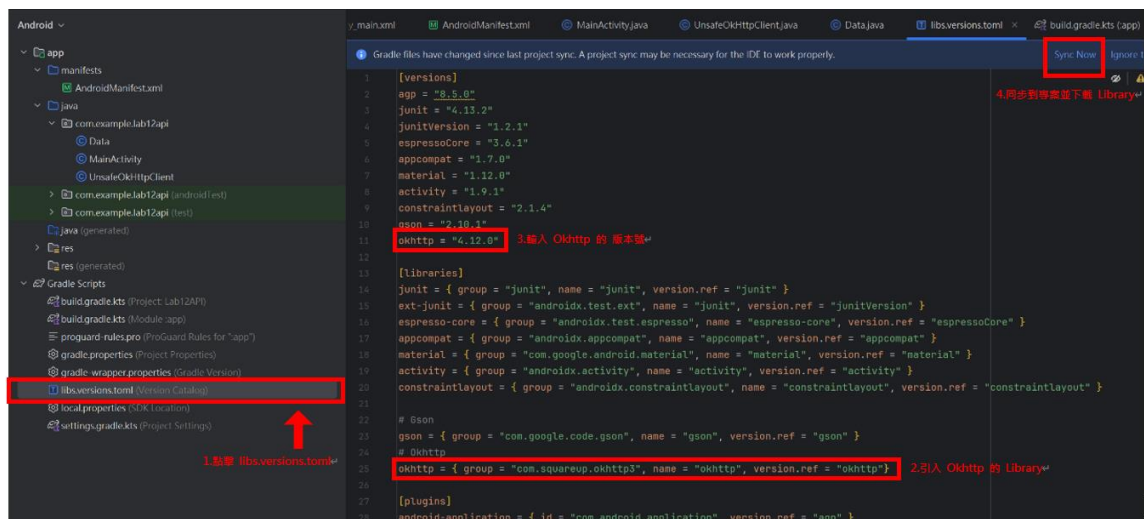


圖 10 加入 OKHttp library

下圖 11 在 gradle 的 dependencies 中，加入 implementation(libs.okhttp) 然後同步讓系統將其匯入。

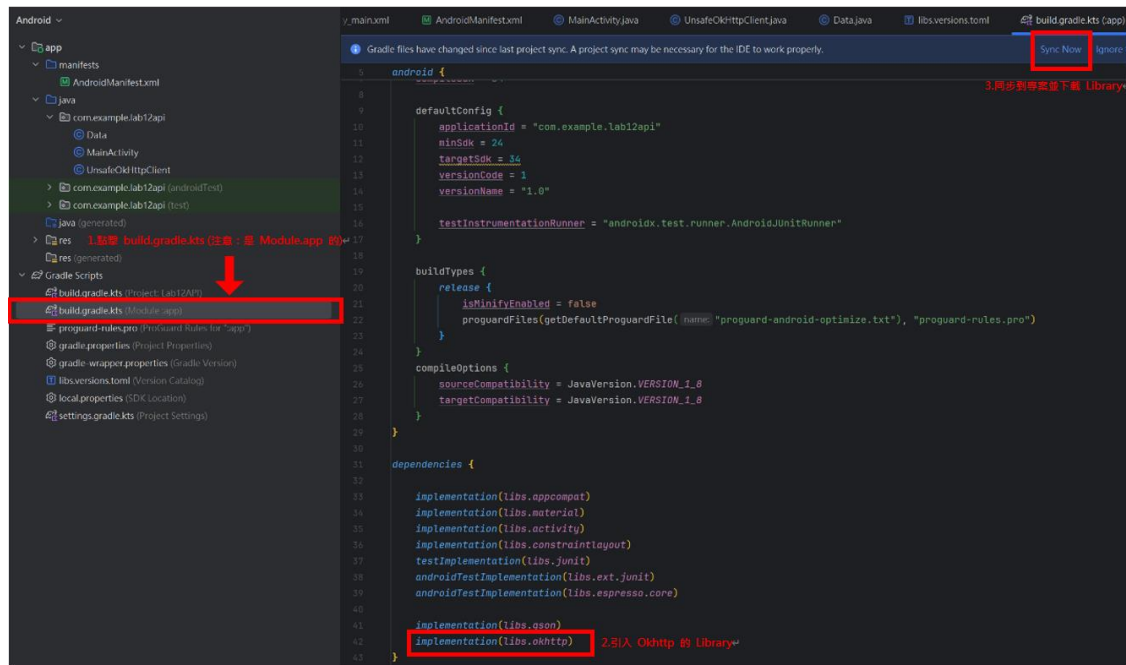
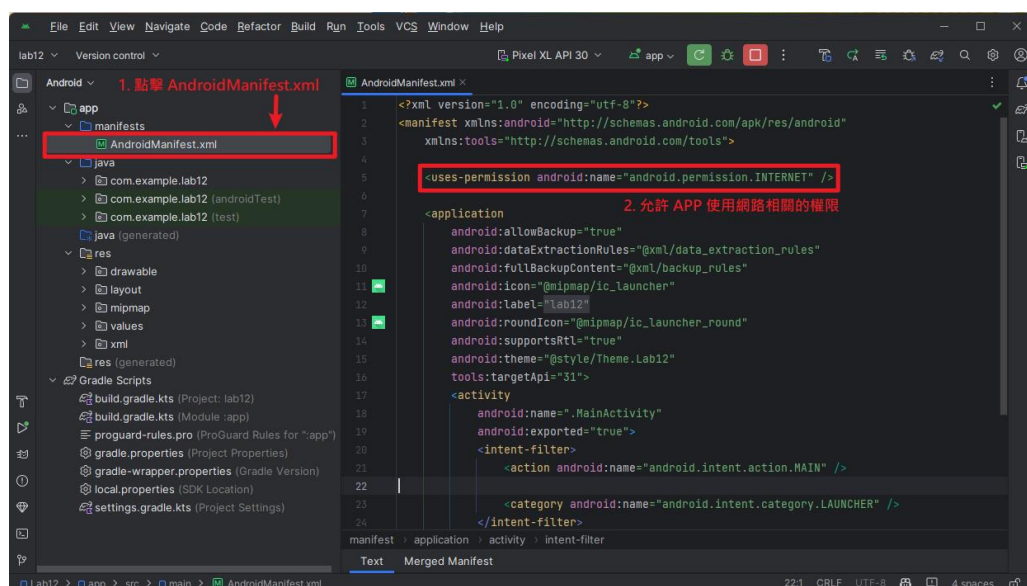


圖 11 加入 Okhttp 的 dependencies

接著，由於我們要讓 Android 進行網路服務，因此需要在 AndroidManifest.xml 加入網路連線的權限。



● Http Get

OkHttp 使用 Get Request 有以下步驟：

Step1：建立一個 Request 物件，並使用 url()方法加入 URL

```
Request req = new Request.Builder()
    .url("https://jsonplaceholder.typicode.com/comments?postId=1").build();
```

Step2：建立 OkHttpClient 物件，newCall()送出請求，enqueue()接收回傳

```
new OkHttpClient().newCall(req).enqueue(new Callback() {
    發送失敗執行此方法
    @Override
    public void onFailure(Call call, IOException e) {
    }
}
```

發送成功執行此方法

```
@Override
    public void onResponse(Call call, Response response) throws
IOException {
        Step3：用 response.body().string()取得 Json 字串
        String json = response.body().string();
    }
});
```

以上就是發送一個 get 請求的步驟，首先建立一個 Request 物件，參數要有 url 來設定網址或是 Get Request。

然後通過 request 的對象去產生得到一個 Call 物件，類似於將 Request 封裝成了任務，既然是任務，就會有 execute()和 cancel()等方法。

最後採用非同步執行方式去執行，這裡使用 newCall().enqueue 然後等待任務執行完成，我們便可在 Callback 中即可得到結果。

● Http Post

OkHttp 使用 Post Request 有以下步驟：

Step1：建立一個 RequestBody 物件，設定 Request 的參數

```
RequestBody body = RequestBody.create(  
    "{\"id\":123, \"name\":\"Guru\"}",  
    MediaType.parse("application/json; charset=utf-8"));
```

Step2：建立一個 Request 物件，加入 URL 與 RequestBody

```
Request request = new Request.Builder()  
    .url("https://tools-api.italkutalk.com/java/lab12/test")  
    .post(body).build();
```

Step3：建立 OkHttpClient 物件，使用 newCall()送出 request，enqueue()接收結果

```
new OkHttpClient().newCall(req).enqueue(new Callback() {
```

發送失敗執行此方法

```
@Override
```

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

發送成功執行此方法

```
@Override
```

```
public void onResponse(Call call, Response response) throws  
IOException {
```

Step4：用 response.body().string()取得 Json 字串

```
String json = response.body().string();
```

```
}
```

```
});
```

說明

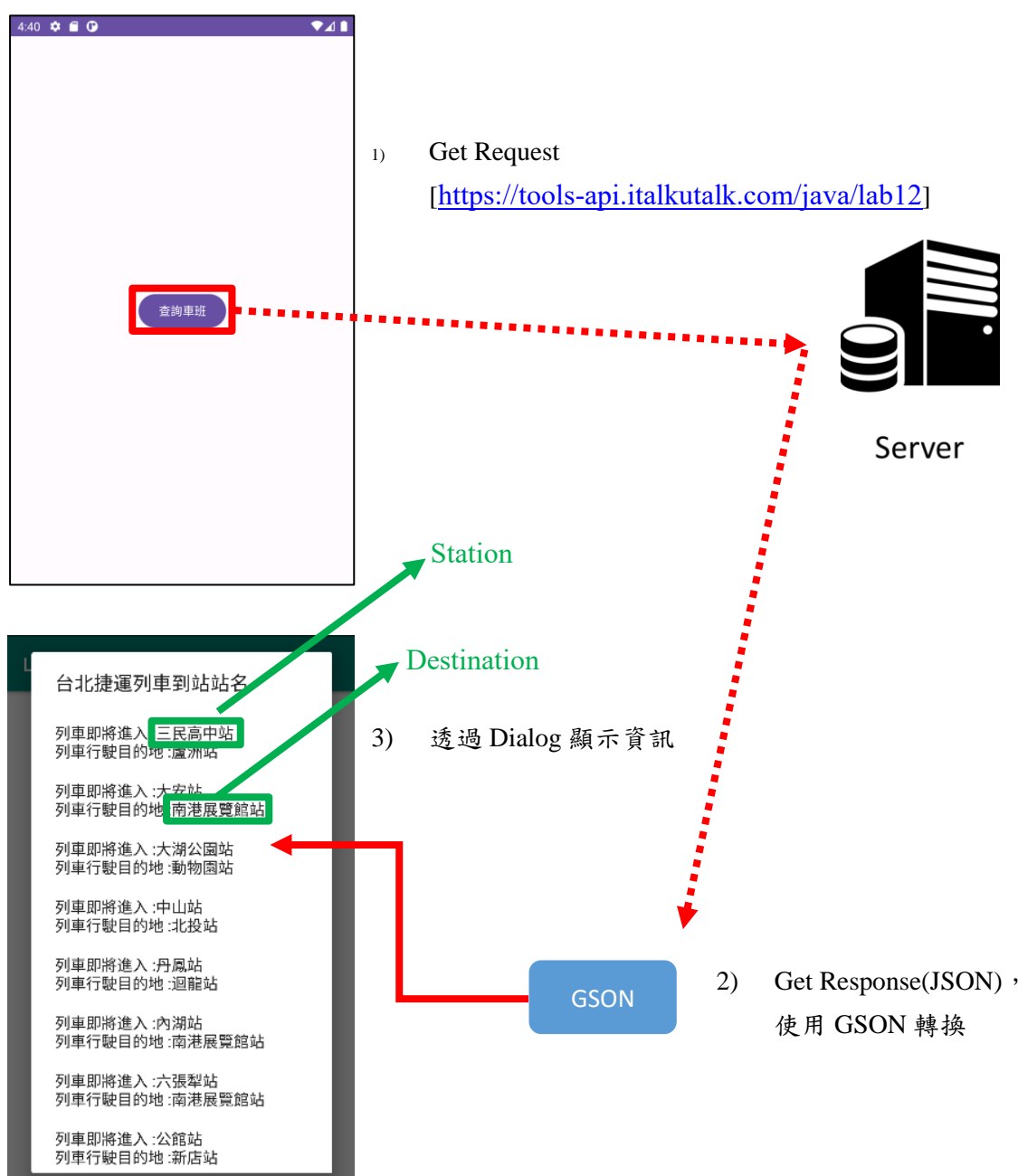
Http Post 與 Get 唯一的差別是 Request 物件需要多一個 post()參數，post()中我們要加入 RequestBody 物件，RequestBody 用來封裝資料(第一個參數)，以及 Body 的格式(第二個參數)。由於我們要採用 JSON 的格式傳送，因此格式部分要加入"application/json; charset=utf-8"，資料部分則加入要傳送過去的 JSON 字串。

12.2 設計重點：

- 練習使用 Http Get 取得“臺北捷運列車到站站名” API 資料，流程如下：

API 連結：<https://tools-api.italkutalk.com/java/lab12>

- 1) 按下按鈕後，送出 Http Get 取得 API response(JSON 字串)
- 2) 使用 GSON 將 Response 的 JSON 字串轉換成物件
- 3) 從物件中取得 Station(列車進入月台車站站名)與 Destination(行駛目的地)資訊，並使用 AlertDialog 顯示



12.3 設計步驟：

Step1 建立新專案，以及下圖 12 對應的 class 和 xml 檔：

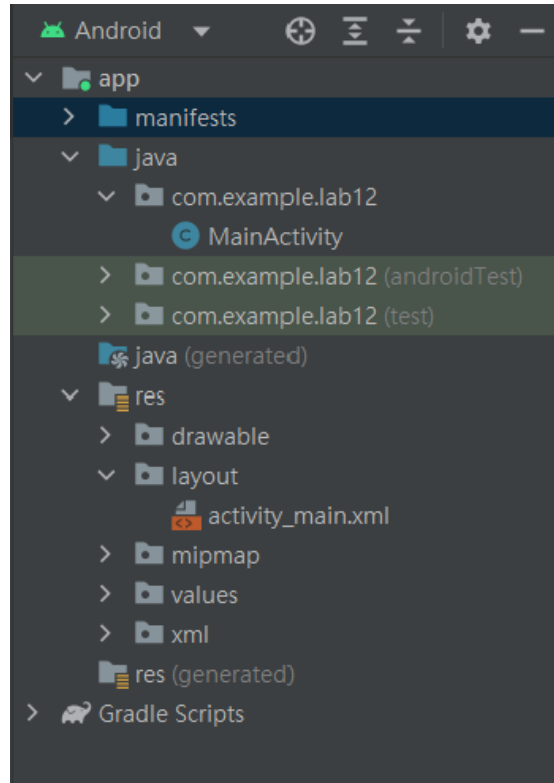


圖 12、專案架構

Step2 繪製 activity_main.xml 如圖 13 所示：

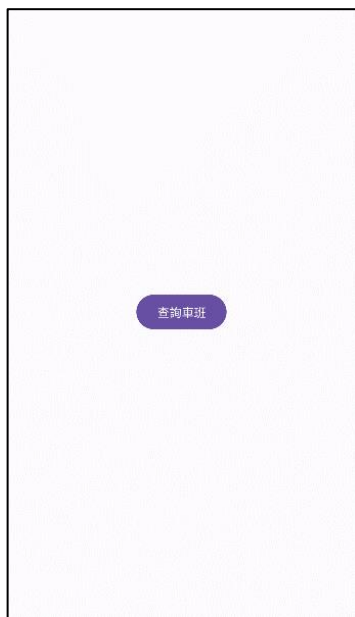


圖 13-1、預覽畫面

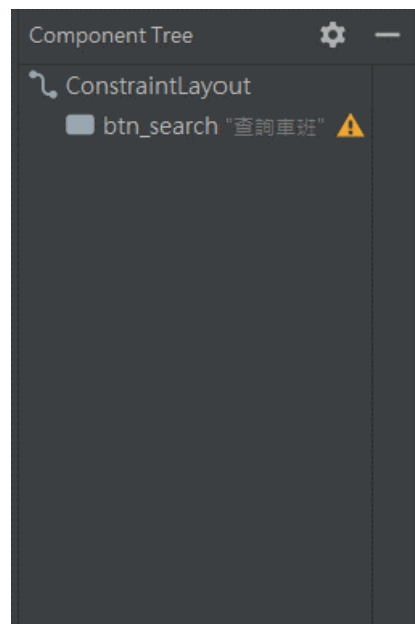


圖 13-2、元件樹

對應的 xml 如下：

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btn_search"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="查詢車班"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```


Step3 圖 14 在 libs.versions.toml 中，我們加入 gson = { group = "com.google.code.gson", name = "gson", version.ref = "gson" } 與 okhttp = { group = "com.squareup.okhttp3", name = "okhttp", version.ref = "okhttp" } 以及各自的版本號後讓系統將其匯入。

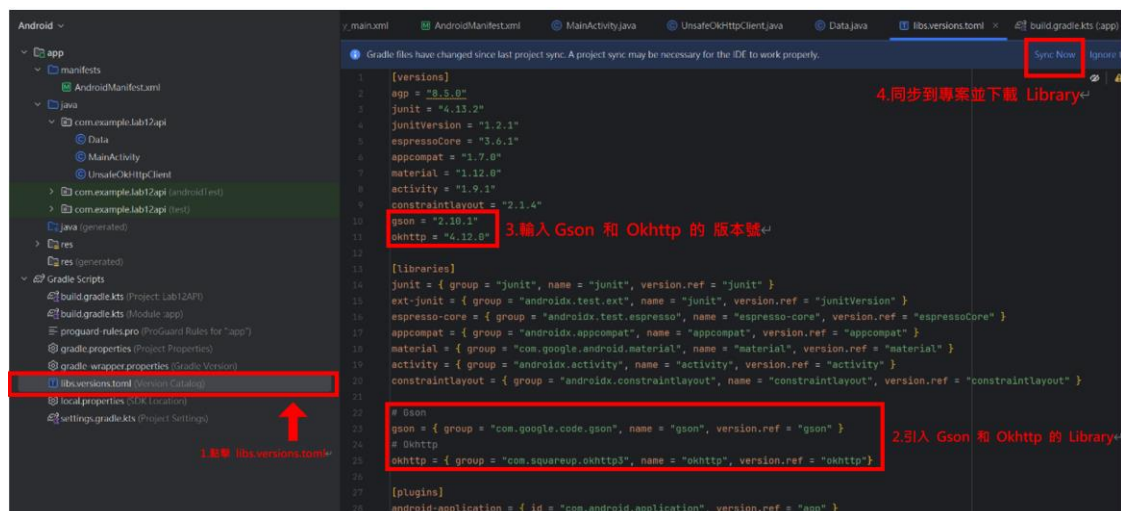


圖 14、加入 GSON 與 OKHttp 的 library

圖 15 在 gradle 的 dependencies 中，我們加入 `implementation(libs.gson)` 跟 `implementation(libs.okhttp)`後讓系統將其匯入。後讓系統將其匯入。

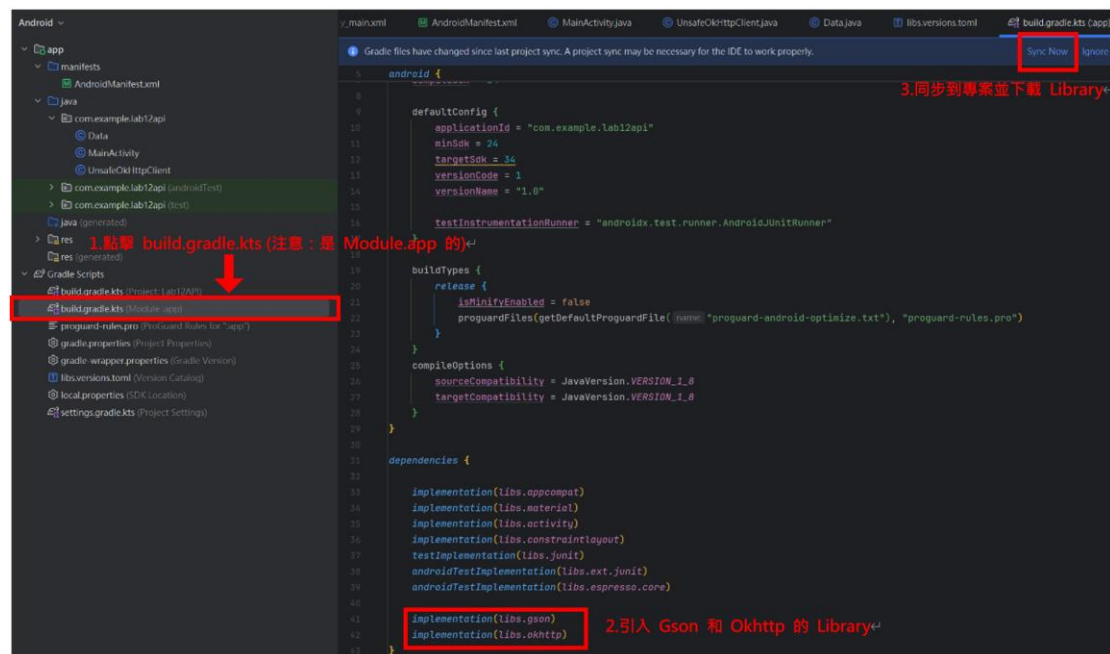


圖 15 加入 GSON 與 OKHttp 的 dependencies

Step4 在 AndroidManifest.xml 加入網路連線的權限及設定。

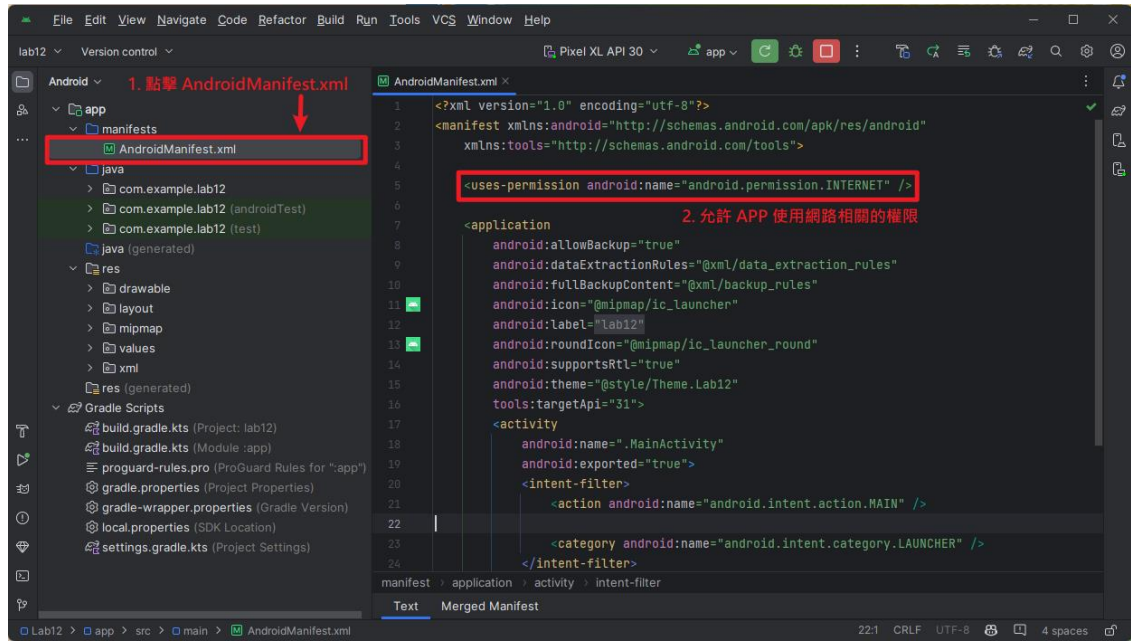



圖 16 添加網路相關的權限

Step5

[<https://tools-api.italkutalk.com/java/lab12>] 是否能成功，同時得到 Response。



```
{
  "result": {
    "limit": 1000,
    "offset": 0,
    "count": 22,
    "sort": "",
    "results": [
      {
        "Station": "三民高中站",
        "Destination": "蘆洲站",
        "_id": 1,
        "UpdateTime": "2017-11-28T09:59:41.91"
      },
      {
        "Station": "大安站",
        "Destination": "南港展覽館站",
        "_id": 2,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "大湖公園站",
        "Destination": "動物園站",
        "_id": 3,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "中山站",
        "Destination": "北投站",
        "_id": 4,
        "UpdateTime": "2017-11-28T09:59:36.72"
      },
      {
        "Station": "丹鳳站",
        "Destination": "迴龍站",
        "_id": 5,
        "UpdateTime": "2017-11-28T09:59:31.22"
      },
      {
        "Station": "內湖站",
        "Destination": "南港展覽館站",
        "_id": 6,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "六張犁站",
        "Destination": "南港展覽館站",
        "_id": 7,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "公館站",
        "Destination": "新店站",
        "_id": 8,
        "UpdateTime": "2017-11-28T09:59:43.503"
      },
      {
        "Station": "台大醫院站",
        "Destination": "象山站",
        "_id": 9,
        "UpdateTime": "2017-11-28T09:59:36.72"
      },
      {
        "Station": "行天宮站",
        "Destination": "南勢角站",
        "_id": 10,
        "UpdateTime": "2017-11-28T09:59:31.22"
      },
      {
        "Station": "西門站",
        "Destination": "松山站",
        "_id": 11,
        "UpdateTime": "2017-11-28T09:59:33.933"
      },
      {
        "Station": "東湖站",
        "Destination": "動物園站",
        "_id": 12,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "南京復興站",
        "Destination": "南港展覽館站",
        "_id": 13,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "後山埤站",
        "Destination": "南港展覽館站",
        "_id": 14,
        "UpdateTime": "2017-11-28T09:59:27.917"
      },
      {
        "Station": "徐匯中學站",
        "Destination": "南勢角站",
        "_id": 15,
        "UpdateTime": "2017-11-28T09:59:31.997"
      },
      {
        "Station": "漚哩岸站",
        "Destination": "大安站",
        "_id": 16,
        "UpdateTime": "2017-11-28T09:59:16.387"
      },
      {
        "Station": "景美站",
        "Destination": "新店站",
        "_id": 17,
        "UpdateTime": "2017-11-28T09:59:23.427"
      },
      {
        "Station": "港墘站",
        "Destination": "南港展覽館站",
        "_id": 18,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "圓山站",
        "Destination": "象山站",
        "_id": 19,
        "UpdateTime": "2017-11-28T09:59:36.72"
      },
      {
        "Station": "萬芳社區站",
        "Destination": "動物園站",
        "_id": 20,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "劍南路站",
        "Destination": "南港展覽館站",
        "_id": 21,
        "UpdateTime": "2017-11-28T09:59:34"
      },
      {
        "Station": "頭前庄站",
        "Destination": "南勢角站",
        "_id": 22,
        "UpdateTime": "2017-11-28T09:59:41.247"
      }
    ]
  }
}
```

圖 17、測試 Http GET

Response

Step6 圖 18 使用這網頁：<https://jsonformatter.org/json-parser>，將 Response 做排版，然後可以得到右邊的資料結構。

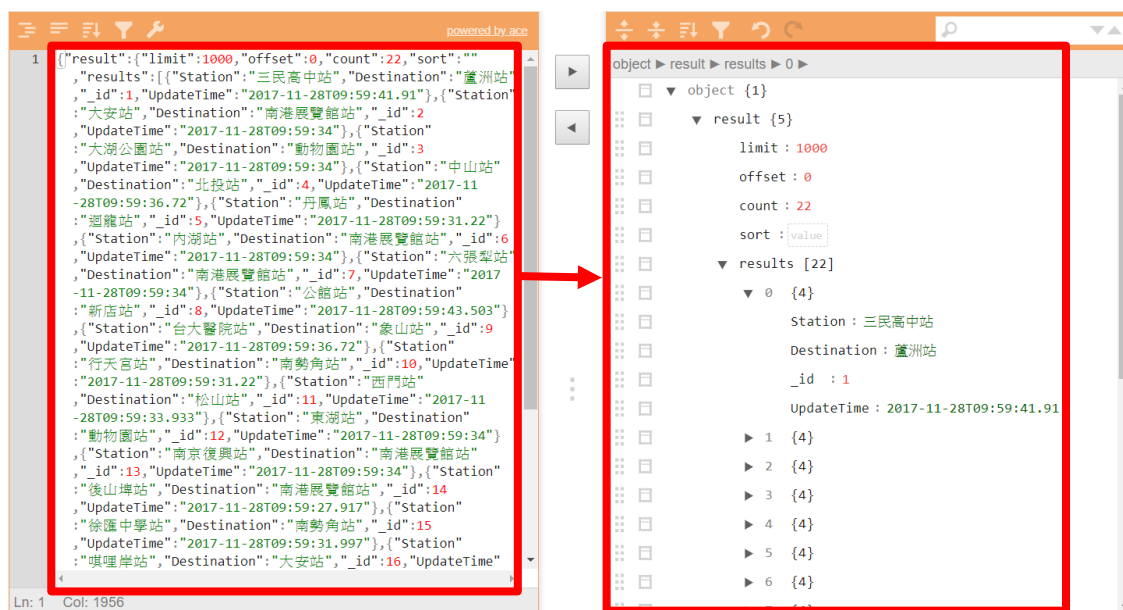


圖 18、排版後的資料

Step7 根據 Step6 資料結構的 JSON，在 MainActivity 中客製化 Data 類別，該資料結構必須依照 Response 來做設計，因此要比對網頁的 Response 結構去規劃類別內容。由於我們只需要 Station 與 Destination 的資訊，因此只需要提取兩者變數即可。

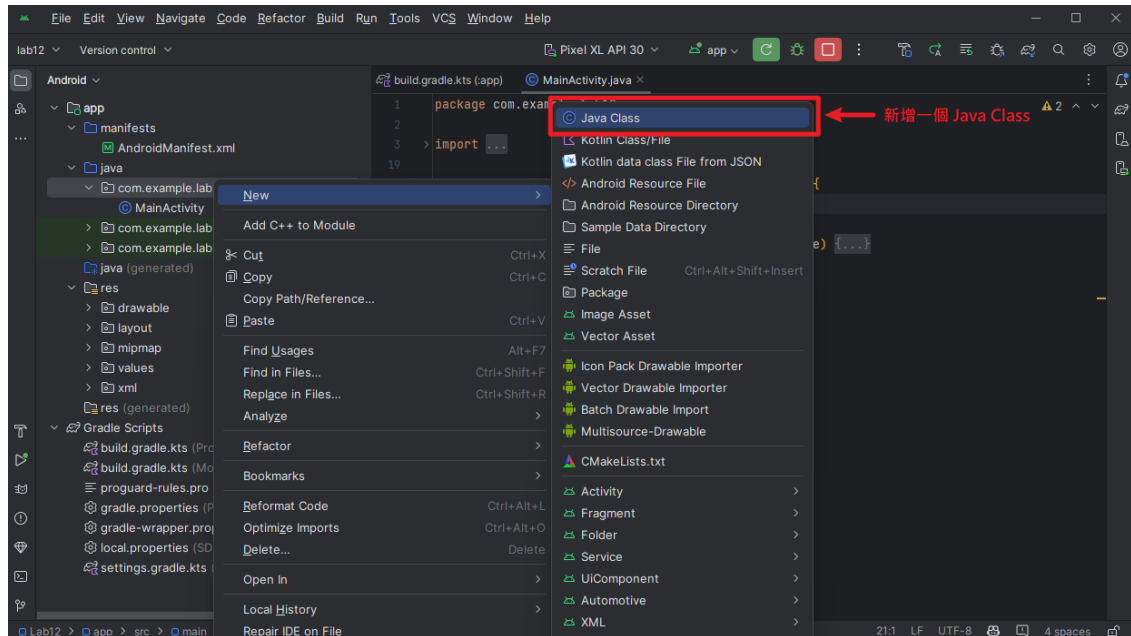


圖 19、建立一個 Java Class



圖 20、將 Java Class 命名為 Data

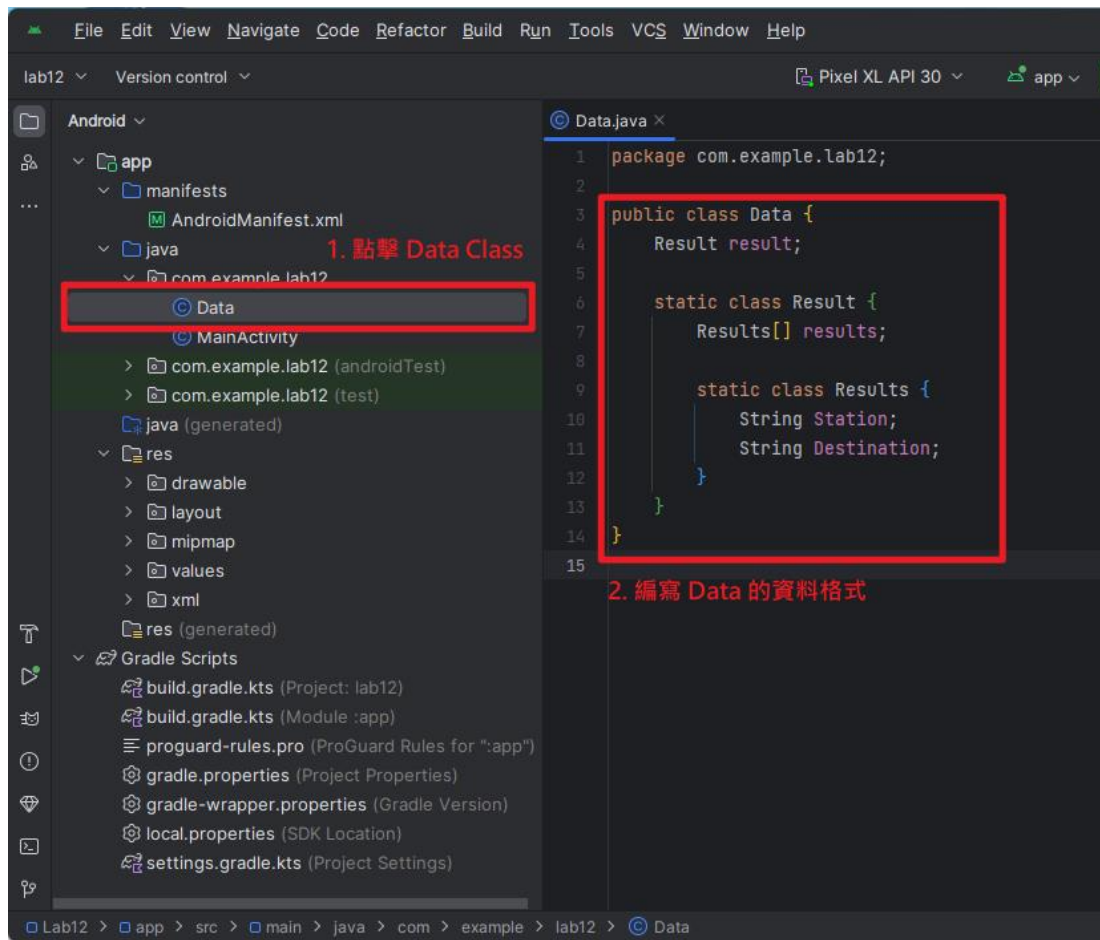
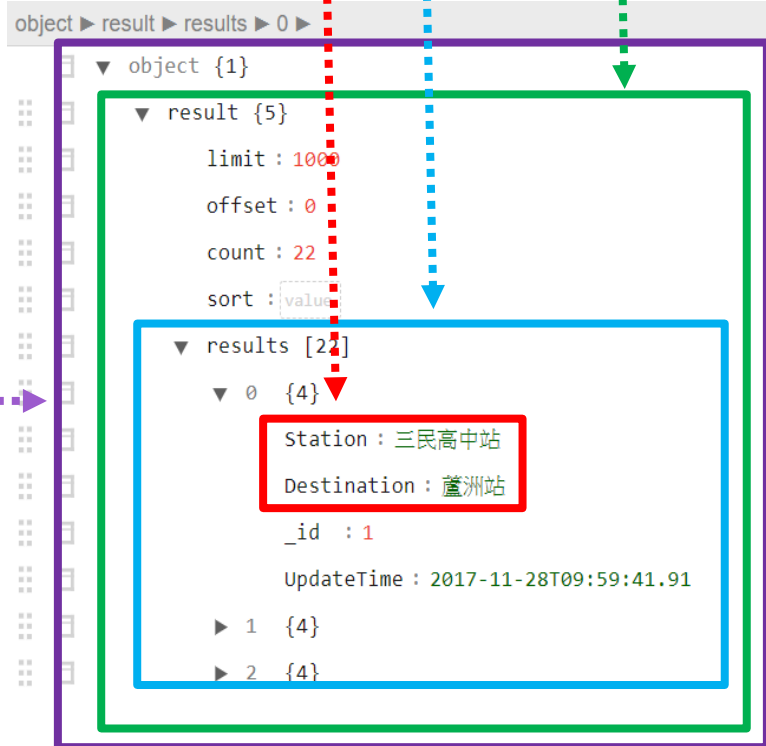


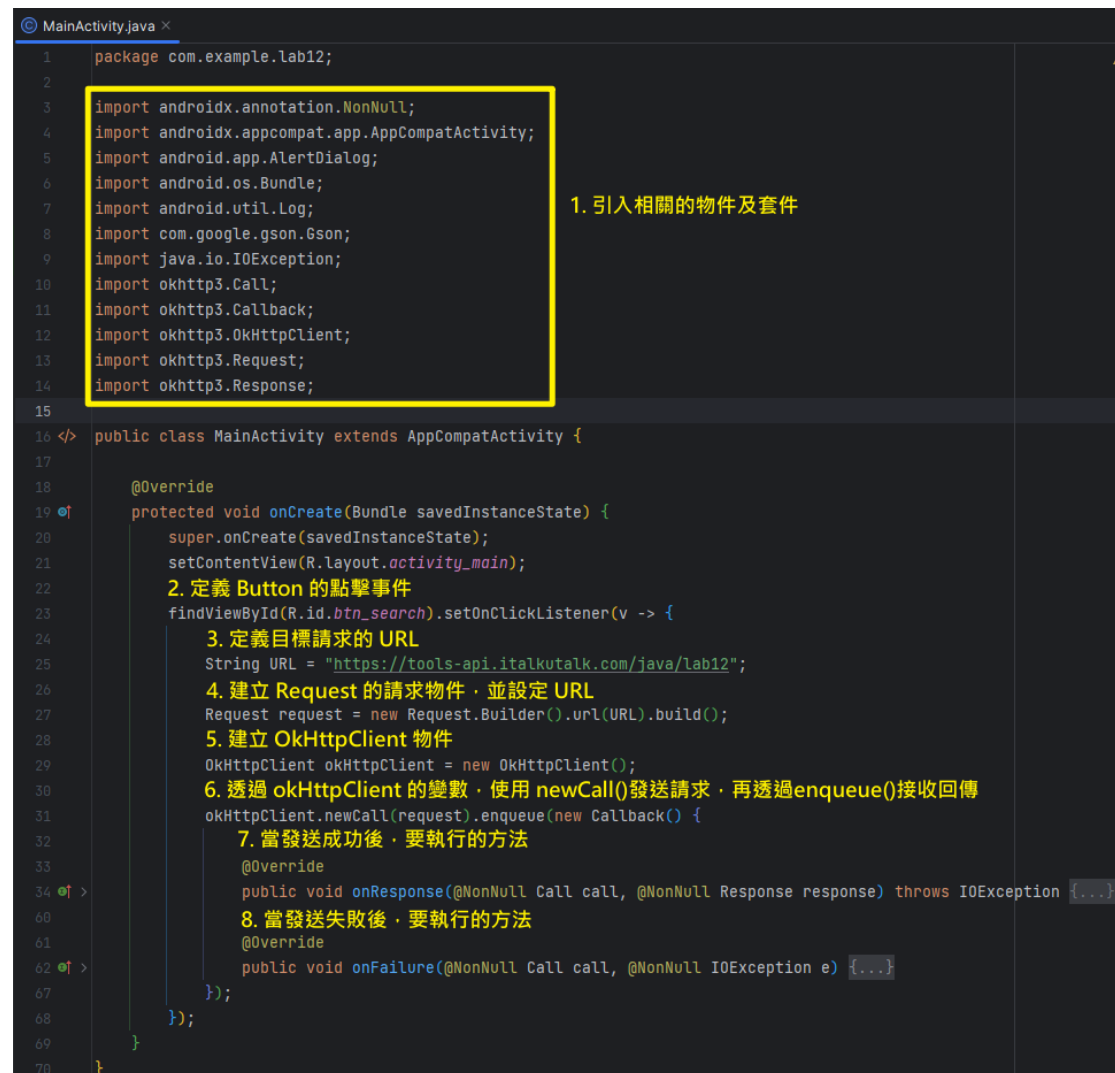
圖 21、編寫 Data 的資料格式

```
public class Data {  
    Result result;  
  
    static class Result {  
        Results[] results;  
  
        static class Results {  
            String Station;  
            String Destination;  
        }  
    }  
}
```

The diagram illustrates the nested class structure of the provided code. A purple box encloses the entire `Data` class. Inside it, a green box encloses the `Result` static class. Within the `Result` class, a blue box encloses the `Results` static class. Finally, a red box highlights the `Station` and `Destination` attributes within the `Results` class. Colored dashed arrows connect these boxes to the corresponding parts of the object model below: a purple arrow from the `Data` class to the `object {1}` container, a green arrow from the `Result` class to the `result {5}` object, a blue arrow from the `Results` class to the `results [22]` array, and a red arrow from the `Station` and `Destination` attributes to the specific object in the array.



Step8 編寫查詢按鈕按下後，使用 OkHttpClient 來發出 Http Get Request 至 <https://tools-api.italkutalk.com/java/lab12>，接收到 Response 之後，將 JSON 字串經由 GSON 轉換至 Data 物件之中，再將 Data 物件 轉換放入 AlertDialog 之中。



```
1 package com.example.lab12;
2
3 import androidx.annotation.NonNull;
4 import androidx.appcompat.app.AppCompatActivity;
5 import android.app.AlertDialog;
6 import android.os.Bundle;
7 import android.util.Log;
8 import com.google.gson.Gson;
9 import java.io.IOException;
10 import okhttp3.Call;
11 import okhttp3.Callback;
12 import okhttp3.OkHttpClient;
13 import okhttp3.Request;
14 import okhttp3.Response;
```

1. 引入相關的物件及套件

```
15
16 </> public class MainActivity extends AppCompatActivity {
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22         2. 定義 Button 的點擊事件
23         findViewById(R.id.btn_search).setOnClickListener(v -> {
24             3. 定義目標請求的 URL
25             String URL = "https://tools-api.italkutalk.com/java/lab12";
26             4. 建立 Request 的請求物件，並設定 URL
27             Request request = new Request.Builder().url(URL).build();
28             5. 建立 OkHttpClient 物件
29             OkHttpClient okHttpClient = new OkHttpClient();
30             6. 透過 okHttpClient 的變數，使用 newCall() 發送請求，再透過 enqueue() 接收回傳
31             okHttpClient.newCall(request).enqueue(new Callback() {
32                 7. 當發送成功後，要執行的方法
33                 @Override
34                 public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {...}
35             });
36             8. 當發送失敗後，要執行的方法
37             @Override
38             public void onFailure(@NonNull Call call, @NonNull IOException e) {...}
39         });
40     }
41 }
```



```

okHttpClient.newCall(request).enqueue(new Callback() {
    7. 當發送成功後・要執行的方法
    @Override
    public void onResponse(@NonNull Call call, @NonNull Response response) throws IOException {
        if (response.code() == 200) {判斷回傳的HTTP狀態碼是否為 200・以及回傳是否有值?
            if (response.body() == null) return;

            使用 Gson 轉換伺服器回傳的 Response 資料
            Data data = new Gson().fromJson(response.body().string(), Data.class);
            建立 String Array 來儲存伺服器響應的資料
            final String[] items = new String[data.result.results.length];
            使用迴圈將資料填入String Array中
            for (int i = 0; i < items.length; i++) {
                items[i] = "\n列車即將進入：" + data.result.results[i].Station +
                    "\n列車行駛目的地：" + data.result.results[i].Destination;
            }

            由於API請求屬於複雜任務・所以OkHttp請求會發生在背景執行緒
            所以需要透過runOnUiThread切換執行緒回【主執行緒】
            runOnUiThread() -> {
                new AlertDialog.Builder(context: MainActivity.this)
                    .setTitle("台北捷運列車到站站名")
                    .setItems(items, listener: null)
                    .show(); 使用 Dialog 顯示伺服器回傳的資料
            });
        } else if (!response.isSuccessful()) {
            Log.e(tag: "伺服器錯誤", msg: response.code() + " " + response.message());
        }
        else {
            Log.e(tag: "其他錯誤", msg: response.code() + " " + response.message());
        }
    }
});
    8. 當發送失敗後・要執行的方法
    @Override
    public void onFailure(@NonNull Call call, @NonNull IOException e) {
        if (e.getMessage() != null) {
            Log.e(tag: "查詢失敗", e.getMessage());
        }
    }
});

```

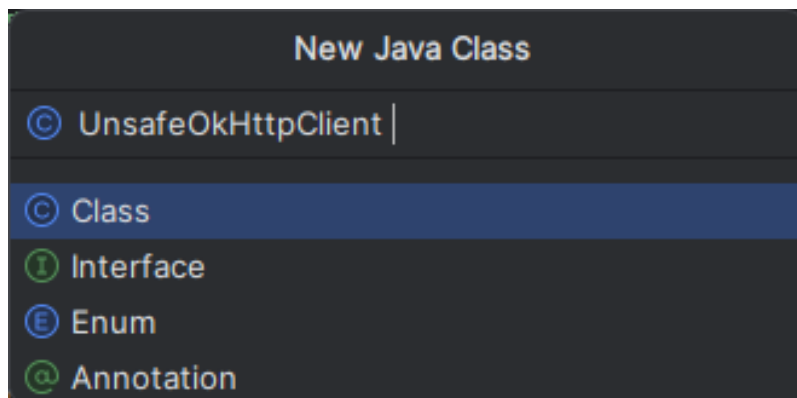
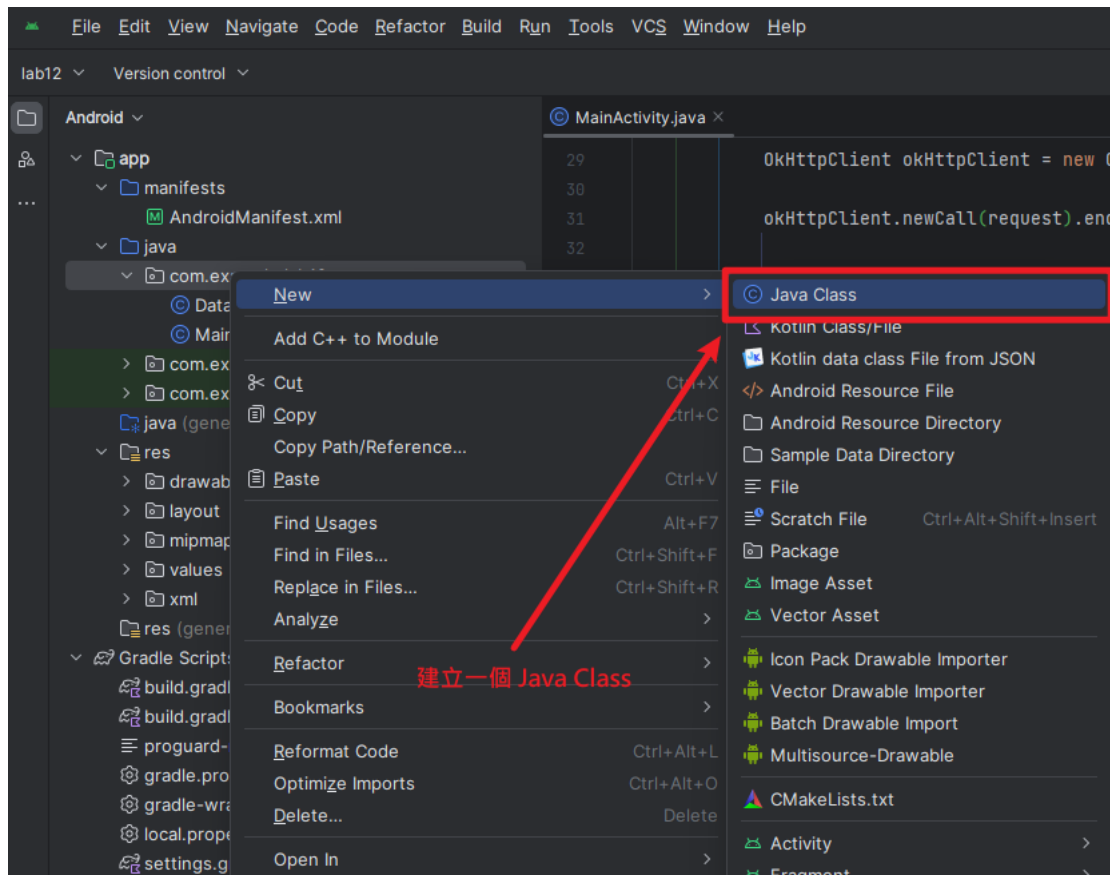

助教補充說明：

由於通過 Android 模擬器發送 API 請求，無法通過學校的 SSL 驗證。所以我們需要通過一些特殊方式，使 **OkHttpClient** 物件可以繞過學校的 SSL 驗證。

說明

如果使用的是自己的網路環境，就不會出現 SSL 驗證失敗的問題。所以同學們在回家練習的時候，可以不用繞過 SSL 驗證的方式來呼叫 API。

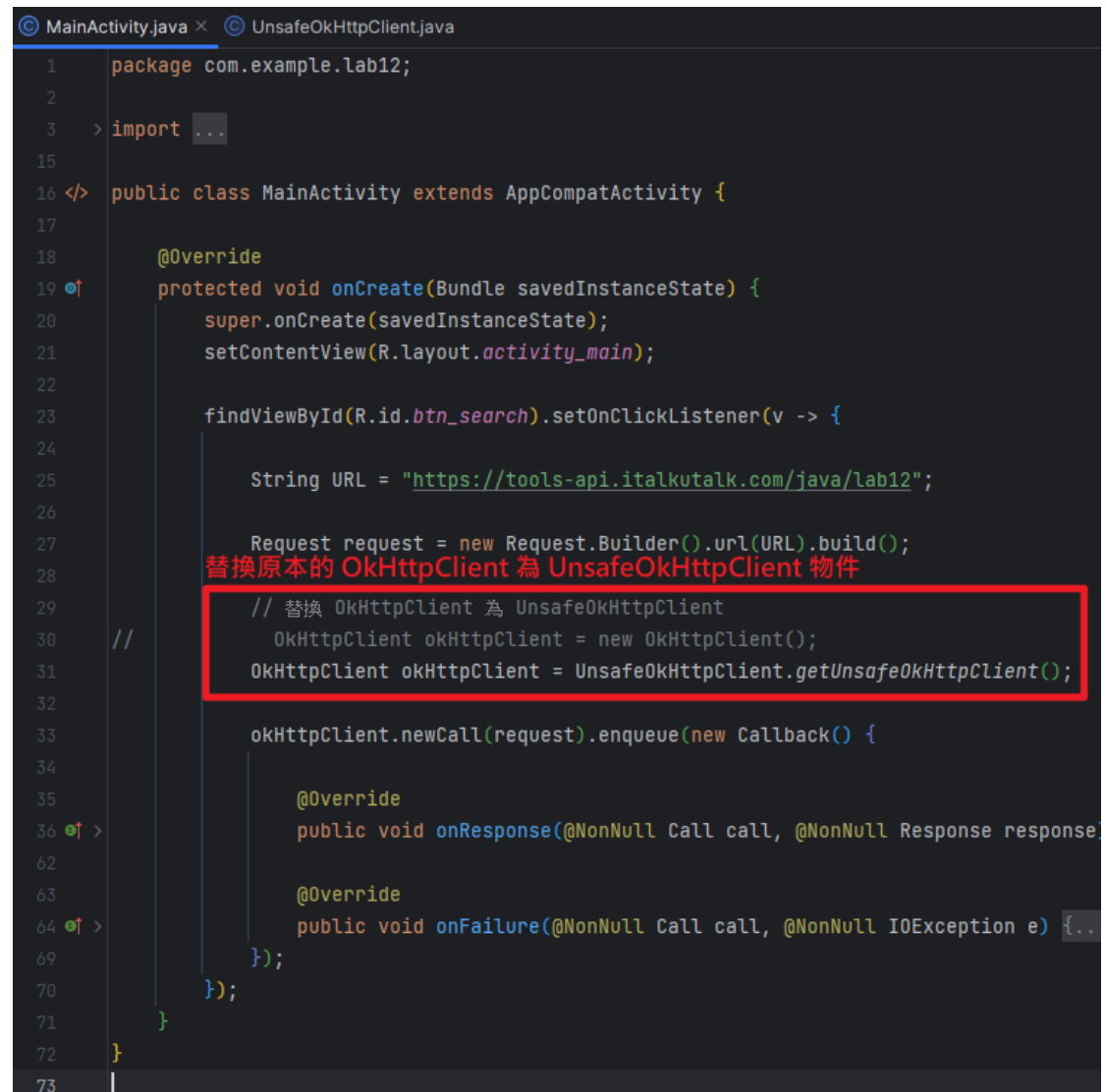
1. 建立一個名為 **UnsafeOkHttpClient** 的 Java 物件。



2. 編寫 `UnsafeOkHttpClient` 的程式碼，裡面包含 `getUnsafeOkHttpClient()` 的靜態方法，用來回傳可以信任所有憑證的 `OkHttpClient` 物件。

```
1 package com.example.lab12;
2
3 import java.security.SecureRandom;
4 import java.security.cert.X509Certificate;
5 import javax.net.ssl.SSLContext;
6 import javax.net.ssl.TrustManager;
7 import javax.net.ssl.X509TrustManager;
8
9 import okhttp3.OkHttpClient;
10
11 public class UnsafeOkHttpClient {
12     取得一個信任所有憑證的 OkHttpClient
13     Returns: OkHttpClient
14
15     @
16     public static OkHttpClient getUnsafeOkHttpClient() {
17         try {
18             // 建立一個信任所有憑證的 X509TrustManager
19             final TrustManager[] trustAllCerts = new TrustManager[]{
20                 new X509TrustManager() {
21                     @Override
22                     public void checkClientTrusted(X509Certificate[] x509Certificates, String s) {
23                         // 不執行任何動作，接受所有用戶端的憑證。
24                     }
25                     @Override
26                     public void checkServerTrusted(X509Certificate[] x509Certificates, String s) {
27                         // 不執行任何動作，接受所有伺服器的憑證。
28                     }
29                     @Override
30                     public X509Certificate[] getAcceptedIssuers() {
31                         // 返回空陣列表示接受所有發行商的憑證。
32                         return new X509Certificate[]{};
33                     }
34                 }
35             };
36
37             // 初始化 SSLContext 物件，並使用剛剛建立的信任所有憑證的 X509TrustManager 初始化。
38             final SSLContext sslContext = SSLContext.getInstance("SSL");
39             sslContext.init(null, trustAllCerts, new SecureRandom());
40             // 建立一個新的 OkHttpClient 並設定 sslSocketFactory 和 hostnameVerifier，以此忽略憑證。
41             // 並將 OkHttpClient 返回。
42             return new OkHttpClient
43                 .Builder()
44                 .sslSocketFactory(sslContext.getSocketFactory(), (X509TrustManager) trustAllCerts[0])
45                 .hostnameVerifier((hostname, session) -> true)
46                 .build();
47         } catch (Exception e) {
48             throw new RuntimeException();
49         }
50     }
51 }
52
```

3. 使用 **UnsafeOkHttpClient** 建立一個新的 OkHttpClient 物件。



```
1 package com.example.lab12;
2
3 > import ...
15
16 </> public class MainActivity extends AppCompatActivity {
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_main);
22
23         findViewById(R.id.btn_search).setOnClickListener(v -> {
24
25             String URL = "https://tools-api.italkutalk.com/java/lab12";
26
27             Request request = new Request.Builder().url(URL).build();
28             替換原本的 OkHttpClient 為 UnsafeOkHttpClient 物件
29             // 替換 OkHttpClient 為 UnsafeOkHttpClient
30             OkHttpClient okHttpClient = new OkHttpClient();
31             OkHttpClient okHttpClient = UnsafeOkHttpClient.getUnsafeOkHttpClient();
32
33             okHttpClient.newCall(request).enqueue(new Callback() {
34
35                 @Override
36                 public void onResponse(@NonNull Call call, @NonNull Response response) {
37
38                 }
39
40                 @Override
41                 public void onFailure(@NonNull Call call, @NonNull IOException e) {
42
43                 }
44             });
45         });
46     }
47 }
```