12.1网络请求之Retrofit

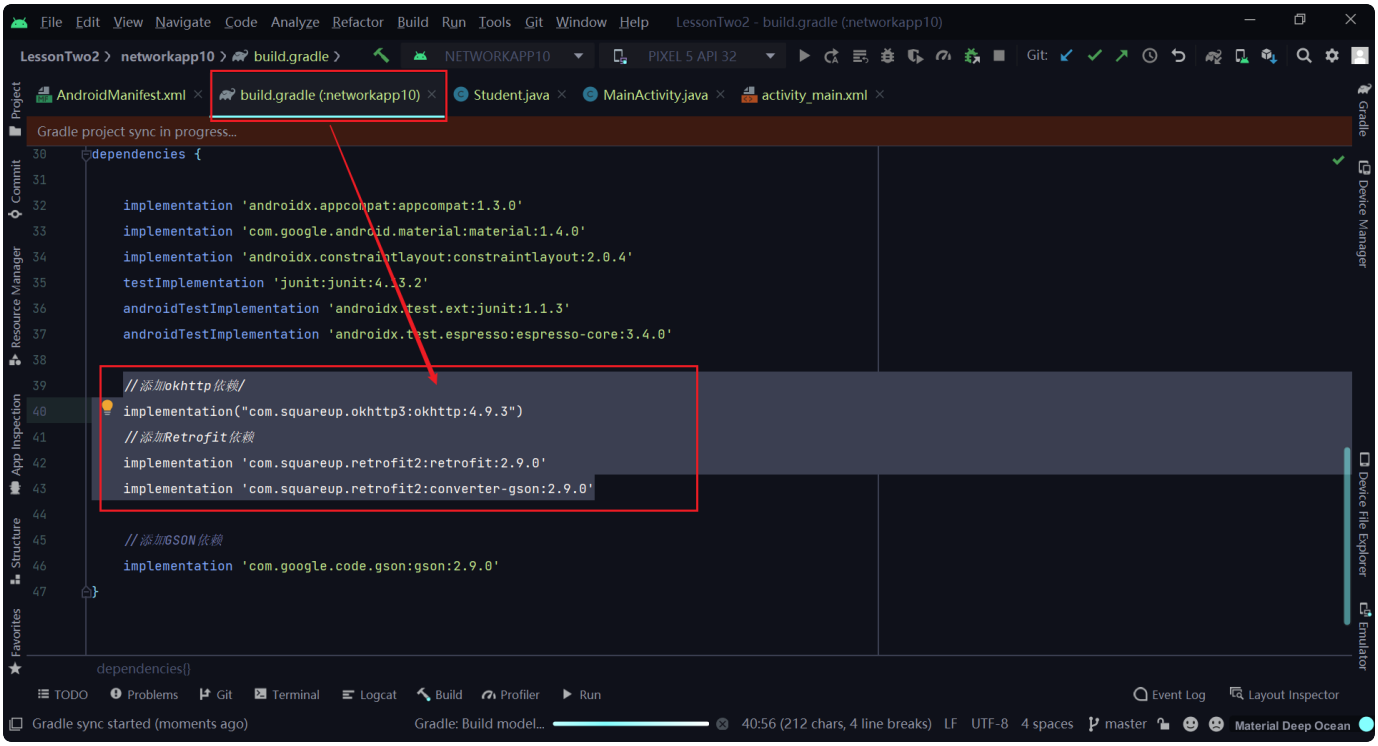
OkhttpPlus

`Retrofit` 是对http网络请求框架的封装，一般由 `okhttp` 来负责网络请求， `retrofit` 对请求接口进行封装。 `retrofit` 通过接口和注解来描述我们的网络请求，然后通过 `client` 去调用 `okhttp` 框架，通过 `addConverterFactory` 来实现对返回的json数据进行处理，转换成我们需要的数据类型，可以理解为**okhttp的加强版，底层封装了Okhttp**。

框架引入

Java

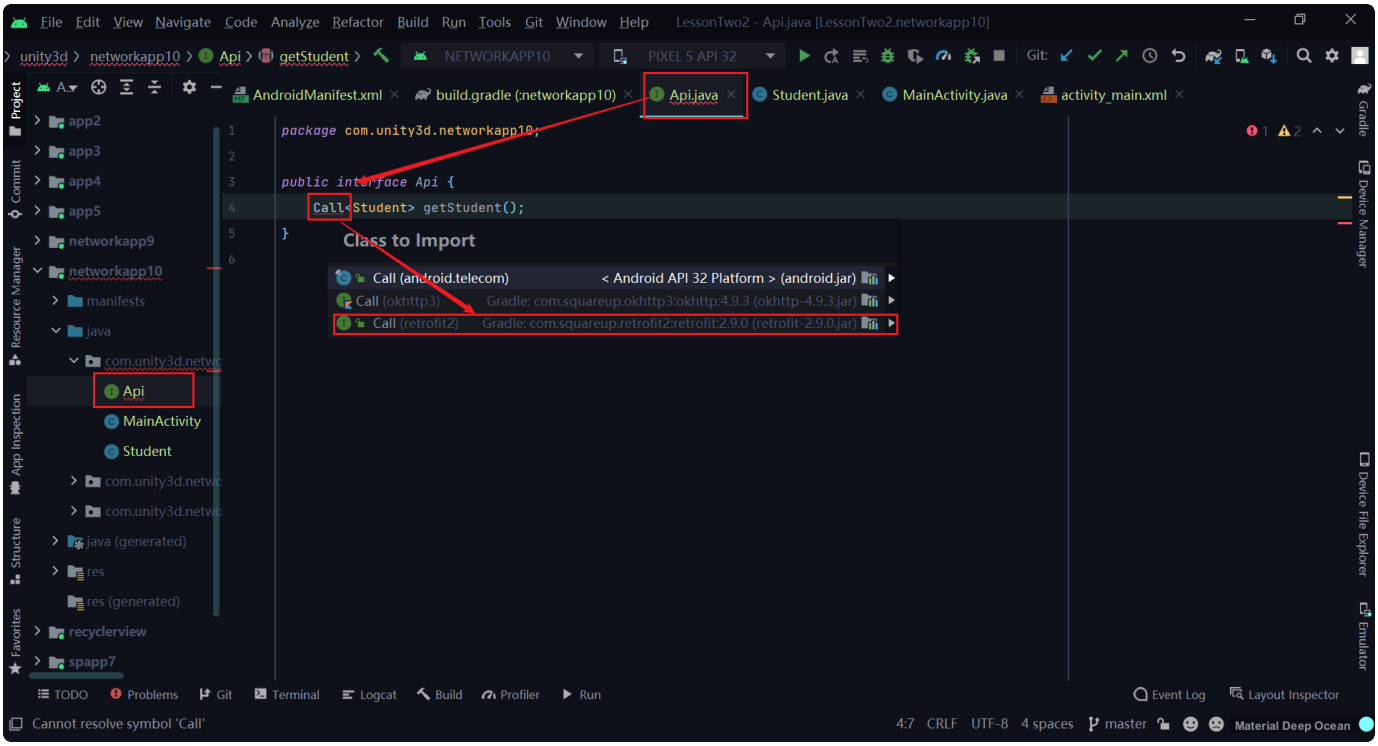
```
//添加okhttp依赖/
implementation("com.squareup.okhttp3:okhttp:4.9.3")
//添加Retrofit依赖
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```



使用步骤

5.2.1 创建接口

创建一个接口，返回值是Call类型，使用泛型封装Student类，注意引入包不要用okhttp的



Java

```java
import retrofit2.Call;
import retrofit2.http.GET;

public interface Api {
    //http://121.4.44.56/object
    //使用get注解，放入请求的地址的最后一部分，前面的部分会在别处拼接
    @GET("object")
    Call<Student> getStudent();
}
```

5.2.2 数据请求&数据解析

回到MainActivity方法，将之前写的用okhttp框架的所有代码注释，以及删除对于的所有包

使用Retrofit框架来实现同样的功能

值得注意的是Retrofit框架在okhttp的基础上进一步封装，我们无需手动解析数据，也无需抛出线程

```java
//使用Retrofit框架
//1..创建一个Retrofit对象
Retrofit retrofit=new Retrofit.Builder()
    .baseUrl("http://121.4.44.56/")
    .addConverterFactory(GsonConverterFactory.create())//表示以GSON框架解析数据
    .build();

//2.获取到Api接口
//返回一个接口实例
Api api = retrofit.create(Api.class);

//3.通过Api获取到实体类Student
retrofit2.Call<Student> call=api.getStudent();

//4.enqueue一个队列，程序可以发起多个请求，将这些请求存在队列中一个一个的处理
call.enqueue(new Callback<Student>() {
    @Override
    public void onResponse(Call<Student> call, Response<Student> response) {
        //请求成功：返回的结果是一个call对象，而不再是response，
        //默认将请求的结果抛回到主线程
        Student student = response.body();
        txtView.setText(student.name);
    }

    @Override
    public void onFailure(Call<Student> call, Throwable t) {
        //请求失败
    }
});
```

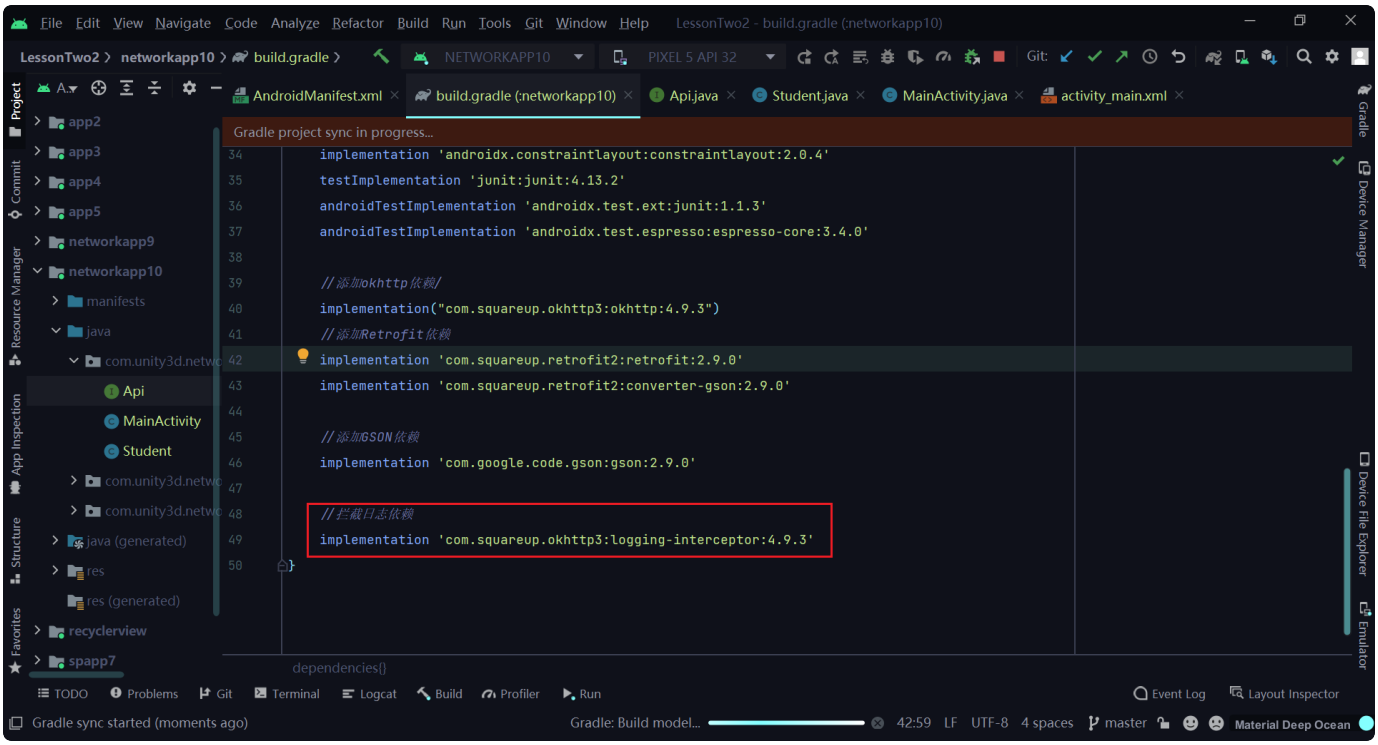
5.2.3 日志拦截（Retrofit优化）

Retrofit框架看上去已经很完美了，但是因为它帮我们完成了数据解析，我们无法捕获数据解析是否成功，所以当数据解析失败时通过 `response.body();` 是获取不到数据的

面对这个问题同样也有解决办法

我们引入一下依赖

```java
//拦截日志依赖
implementation 'com.squareup.okhttp3:logging-interceptor:4.9.3'
```

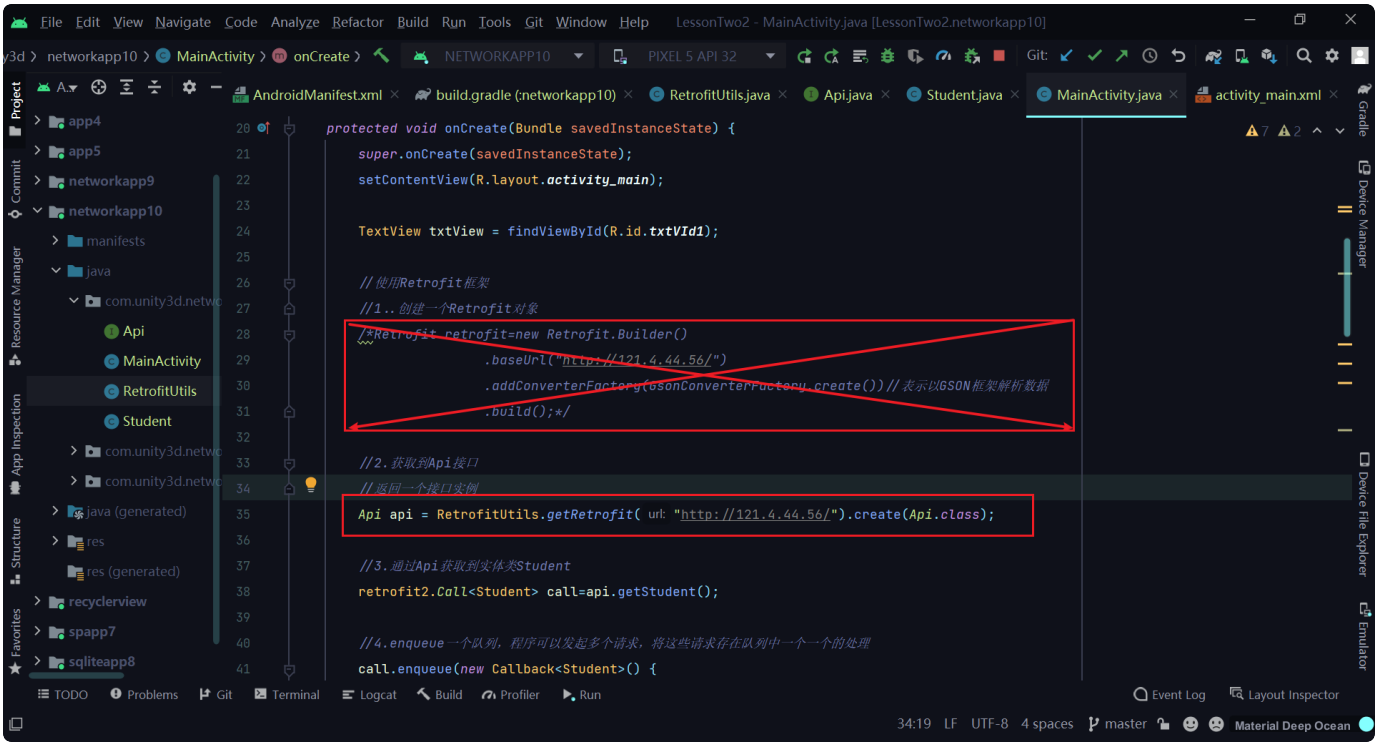


通过如下工具类进行拦截，使用这个工具类我们通过日志过滤的方法查看请求的状态和数据解析是否成功

```java

public class RetrofitUtils {

    public static Retrofit getRetrofit(String url) {
        //日志显示级别
        HttpLoggingInterceptor.Level level= HttpLoggingInterceptor.Level.BODY;
        //新建log拦截器
        HttpLoggingInterceptor loggingInterceptor=new HttpLoggingInterceptor(new HttpLoggingInterceptor.Logger() {
            @Override
            public void log(String message) {
                Log.d("RetrofitMessage","OkHttp====Message:"+message);
            }
        });
        loggingInterceptor.setLevel(level);
        //定制OkHttp
        OkHttpClient.Builder httpClientBuilder = new OkHttpClient
                .Builder();
        //OkHttp进行添加拦截器loggingInterceptor
        httpClientBuilder.addInterceptor(loggingInterceptor);

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(url)
                .addConverterFactory(GsonConverterFactory.create())
                .client( httpClientBuilder.build())
                .build();

        return retrofit;
    }

}
```

使用这个工具类也能简化请求数据的代码，我们只需这么一行代码就能拿到Api接口实例

```java
Api api = RetrofitUtils.getRetrofit("http://121.4.44.56/").create(Api.class);
```



通过debug级的日志过滤我们可以查看数据请求的状态信息和数据解析结果，也方便我们排错分析