retrofit基础资料

Retrofit是什么

官网介绍是A type-safe HTTP client for Android and Java，是一个 RESTful 的 HTTP 网络请求框架的封装，但网络请求不是Retrofit来完成的，它只是封装了请求参数、Header、Url、返回结果处理等信息，而请求是由OkHttp3来完成的。

入门

1. 导包

```
    //网络请求相关
       implementation "com.squareup.retrofit2:retrofit:$rootProject.retrofitVersion"
       implementation "com.squareup.retrofit2:retrofit-mock:$rootProject.retrofitVersion"
       implementation "com.squareup.retrofit2:converter-gson:$rootProject.retrofitVersion"
       implementation 'com.squareup.okhttp3:logging-interceptor:3.5.0'
       implementation "com.squareup.retrofit2:converter-scalars:$rootProject.retrofitVersion"
       implementation "com.squareup.retrofit2:adapter-rxjava2:$rootProject.retrofitVersion"
       implementation "com.squareup.retrofit2:converter-gson:$rootProject.retrofitVersion"
```

2. 定义一个HTTP API接口类

```
interface WanAndroidApi {
   @GET("project/tree/json")
    Call<ProjectBean> getProject();
}
```

3. 使用Retrofit类生成WanAndroidApi 接口实现

```
 Retrofit retrofit = new Retrofit.Builder()//建造者模式
                .baseUrl("https://www.wanandroid.com/")
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        WanAndroidApi wanAndroidApi = retrofit.create(WanAndroidApi.class);//代理实例
```

3. 发送HTTP请求，返回Response可以同步或者异步处理

```
 Call<ProjectBean> call = wanAndroidApi.getProject();//获取具体的某个业务
//同步请求
Response<ProjectBean> response = call.execute();
ProjectBean projectBean = response.body();
//异步请求
call.enqueue(new Callback<ProjectBean>() {
            @Override
            public void onResponse(final Call<ProjectBean> call, final
Response<ProjectBean> response) {
                Log.i("Zero","response: " + response.body());
            }
            @Override
            public void onFailure(final Call<ProjectBean> call, final
Throwable t) {}
        });
```

注解分类解析

请求方法类

序号	名称	说明
1	GET	get请求
2	POST	post请求
3	PUT	put请求
4	DELETE	delete请求
5	PATCH	patch请求，该请求是对put请求的补充，用于更新局部资源
6	HEAD	head请求
7	OPTIONS	option请求
8	HTTP	通用注解，可以替换以上所有的注解，其拥有method, path, hasBody 三个属性

序号 1 ~ 7

- 分别对应 HTTP 的请求方法；
- 接收一个字符串表示接口 path ， 与 baseUrl 组成完整的 Url；
- 可以不指定，结合 @Url 注解使用；
- url 中可以使用变量，如 {id} ，并使用 @Path(“id”) 注解为 {id} 提供值。

```
@GET("project/tree/json")
Call<ProjectBean> getProject1();
```

序号 8

- 可用于替代以上 7 个，及其他扩展方法；
- 有 3 个属性：method、path、hasBody、 举个例子

```
@HTTP(method = "get", path = "project/tree/json",hasBody = false)
    Call<ProjectBean> getProject2();
```

标记类

分类	名称	备注
表单请求	FormUrlEncoded	表示请求实体是一个Form表单，每个键值对需要使用@Field注解
请求参数	Multipart	表示请求实体是一个支持文件上传的Form表单，需要配合使用@Part,适用于 有文件 上传的场景
标记	Streaming	表示响应体的数据用流的方式返回，适用于返回的数据比较大，该注解在在下载大文件的特别有用

FormUrlEncoded

登录页面使用：`Content-Type:application/x-www-form-urlencoded`

- 用于修饰Field注解和FieldMap注解
- 使用该注解,表示请求正文将使用表单网址编码。字段应该声明为参数，并用@Field注释或FieldMap注释。使用FormUrlEncoded注解的请求将具”application/ x-www-form-urlencoded”MIME类型。字段名称和值将先进行UTF-8进行编码,再根据RFC-3986进行URI编码.

Multipart

上传文件使用：`Content-Type:multipart/form-data`

```
///////上传单张图片//////
    /**
     * Multipart：表示请求实体是一个支持文件上传的Form表单，需要配合使用@Part,适用于有文件 上传的场景
     * Part:用于表单字段,Part和PartMap与Multipart注解结合使用,适合文件上传的情况
     * PartMap:用于表单字段,默认接受的类型是Map<String,RequestBody>，可用于实现多文件上传
     * Part 后面支持三种类型，{@link RequestBody}、{@link okhttp3.MultipartBody.Part} 、任意类型；
     *
     * @param file 服务器指定的上传图片的key值
     * @return
     */

@Multipart
@POST("project/upload")
Call<ProjectBean> upload1(@Part("file" + "\";filename=\"" + "test.png") RequestBody file);

@Multipart
@POST("project/xxx")
Call<ProjectBean> upload2(@Part MultipartBody.Part file);

////////请求//////////
//上传单个图片1
File file = new File("");
```

```
RequestBody requestBody =
RequestBody.create(MediaType.parse("image/png"),file);
wanAndroidApi.upload1(requestBody).execute();
//上传单个图片2
 MultipartBody.Part imagePart = MultipartBody.Part.createFormData("上传的
key"
        ,file.getName(),requestBody);
wanAndroidApi.upload2(imagePart)
             .enqueue(new Callback<ProjectBean>() {
                @Override
                public void onResponse(Call<ProjectBean> call,
Response<ProjectBean> response) { }

                @Override
                 public void onFailure(Call<ProjectBean> call, Throwable
t) { }
                });

///////上传多张图片//////
@Multipart
@POST("project/upload")
Call<ProjectBean> upload3(@PartMap Map<String, RequestBody> map);

@Multipart
@POST("project/xxx")
Call<ProjectBean> upload4(@PartMap Map<String, MultipartBody.Part> map);

////////使用/////////
//上传多张图片1
//图片集合
List<File> files = new ArrayList<>();
Map<String, RequestBody> map = new HashMap<>();
for (int i = 0; i < files.size(); i++) {
      RequestBody requestBody =
RequestBody.create(MediaType.parse("image/png"), files.get(i));
       map.put("file" + i + "\";filename=\"" + files.get(i).getName(),
requestBody);
}
wanAndroidApi.upload3(map).execute();
//上传多张图片2
Map<String, MultipartBody.Part> map1 = new HashMap<>();
File file1 = new File("");
RequestBody requestBody1 =
RequestBody.create(MediaType.parse("image/png"), file1);
MultipartBody.Part part1 = MultipartBody.Part.createFormData("上传的key1",
file1.getName(), requestBody1);
map1.put("上传的key1", part1);

File file2 = new File("");
RequestBody requestBody2 =
RequestBody.create(MediaType.parse("image/png"), file2);
MultipartBody.Part part2 = MultipartBody.Part.createFormData("上传的key2",
file2.getName(), requestBody2);
map1.put("上传的key2", part2);
wanAndroidApi.upload4(map1).execute();

//////图文混传/////
    /**
```

```
     * @param params
     * @param files
     * @return
     */
    @Multipart
    @POST("upload/upload")
    Call<ProjectBean> upload5(@FieldMap() Map<String, String> params,
                              @PartMap() Map<String, RequestBody> files);

    /**
     * Part 后面支持三种类型，{@link RequestBody}、{@link okhttp3.MultipartBody.Part} 、任意类型；
     *
     * @param userName
     * @param passWord
     * @param file
     * @return
     */
    @Multipart
    @POST("project/xxx")
    Call<ProjectBean> upload6(@Part("username") RequestBody userName,
                              @Part("password") RequestBody passWord,
                              @Part MultipartBody.Part file);
//////使用///////
MediaType textType = MediaType.parse("text/plain");
        RequestBody name = RequestBody.create(textType, "zero");
        RequestBody password = RequestBody.create(textType, "123456");

        File file = new File("");
        RequestBody requestBody = RequestBody.create(MediaType.parse("image/png"), file);
        MultipartBody.Part part = MultipartBody.Part.createFormData("上传的key", file.getName(), requestBody);

        wanAndroidApi
                .upload6(name, password, part)
                .enqueue(new Callback<ProjectBean>() {
                    @Override
                    public void onResponse(Call<ProjectBean> call, Response<ProjectBean> response) {

                    }

                    @Override
                    public void onFailure(Call<ProjectBean> call, Throwable t) {

                    }
                });
```

Streaming

未使用该注解，默认会把数据全部载入内存，之后通过流获取数据也是读取内存中数据，所以返回数据较大时，需要使用该注解

```java
/**
     * 12.Streaming注解:表示响应体的数据用流的方式返回，适用于返回的数据比较大，该注解在在下载大文件的特别有用
     */
    @Streaming
    @GET
    Call<ProjectBean> downloadFile(@Url String fileUrl);
```

参数类

分类	名称	备注
作用于方法	Headers	用于添加固定请求头，可以同时添加多个。通过该注解添加的请求头不会相互覆盖，而是共同存在
作用于方法参数(形参)	Header	作为方法的参数传入，用于添加不固定值的Header，该注解会更新已有的请求头
请求参数	Body	多用于post请求发送非表单数据,比如想要以post方式传递json格式数据
请求参数	Field	多用于post请求中表单字段,Filed和FieldMap需要FormUrlEncoded结合使用
请求参数	FieldMap	表单字段，与 Field、FormUrlEncoded 配合；接受 Map<String,String>类型，非 String 类型会调用 toString() 方法
请求参数	Part	用于表单字段,Part和PartMap与Multipart注解结合使用,适合文件上传的情况
请求参数	PartMap	表单字段，与 Part 配合，适合文件上传情况；默认接受 Map<String, RequestBody> 类型，非 RequestBody 会通过 Converter 转换
请求参数	HeaderMap	用于URL，添加请求头
请求参数	Path	用于url中的占位符
请求参数	Query	用于Get中指定参数
请求参数	QueryMap	和Query使用类似
请求参数	Url	指定请求路径

注意:

- Map 用来组合复杂的参数;
- Query、QueryMap 与 Field、FieldMap 功能一样，生成的数据形式一样； Query、QueryMap 的数据体现在 Url 上； Field、FieldMap 的数据是请求体;
- {占位符}和 PATH 尽量只用在URL的 path 部分，url 中的参数使用 Query、QueryMap 代替，保证接口的简洁;
- Query、Field、Part 支持数组和实现了 Iterable 接口的类型， 如 List、Set等，方便向后台传递数组，示例如下:

Headers

使用 `@Headers` 注解设置固定的请求头，所有请求头不会相互覆盖，即使名字相同。

```
@Headers("Cache-Control: max-age=640000")
@GET("project/list")
Call<ProjectBean> getMsg1();

@Headers({ "Accept: application/vnd.github.v3.full+json","User-Agent:
Retrofit-Sample-App"})
@GET("project/{username}")
Call<ProjectBean> getMsg2(@Path("username") String username);
```

Header

使用 `@Header` 注解动态更新请求头，匹配的参数必须提供给 @Header ， 若参数值为 null ， 这个头会被省略，否则，会使用参数值的 toString 方法的返回值。

```
@GET("project")
Call<ProjectBean> getProject3(@Header("Authorization") String
authorization);
```

Body

使用 `@Body` 注解，指定一个对象作为 request body 。

```
@POST("project/new")
Call<ProjectBean> createProject(@Body ProjectBean user);
```

Field

- 作用于方法的参数
- 用于发送一个表单请求
- 用String.valueOf()把参数值转换为String,然后进行URL编码,当参数值为null值时,会自动忽略,如果传入的是一个List或array,则为每一个非空的item拼接一个键值对,每一个键值对中的键是相同的,值就是非空item的值,如:name=张三&name=李四&name=王五,另外,如果item的值有空格,在拼接时会自动忽略,例如某个item的值为:张三,则拼接后为name=张三.

```
//固定或可变数组
@FormUrlEncoded
@POST("/list")
Call<ResponseBody> example(@Field("name") String... names);
```

FieldMap

- 作用于方法的参数
- 用于发送一个表单请求
- map中每一项的键和值都不能为空,否则抛出IllegalArgumentException异常

```
FormUrlEncoded
@POST("/examples")
Call<ResponseBody> example(@FieldMap Map<String, String> fields);
```

Part

- 作用于方法的参数,用于定义Multipart请求的每个part
- 使用该注解定义的参数,参数值可以为空,为空时,则忽略
- 使用该注解定义的参数类型有以下3种方式可选:
    - 如果类型是okhttp3.MultipartBody.Part，内容将被直接使用。省略part中的名称,即 @Part MultipartBody.Part part
    - 如果类型是RequestBody，那么该值将直接与其内容类型一起使用。在注释中提供part名称（例如，@Part（“foo”）RequestBody foo）
    - 其他对象类型将通过使用转换器转换为适当的格式。 在注释中提供part名称（例如，@Part（“foo”）Image photo）

PartMap

- 作用于方法的参数,以map的方式定义Multipart请求的每个part
- map中每一项的键和值都不能为空,否则抛出IllegalArgumentException异常
- 使用该注解定义的参数类型有以下2种方式可选:
    - 如果类型是RequestBody，那么该值将直接与其内容类型一起使用
    - 其他对象类型将通过使用转换器转换为适当的格式

```
@Multipart
@POST("upload/upload")
Call<ProjectBean> upload5(@FieldMap() Map<String, String> params,
                          @PartMap() Map<String, RequestBody> files,
                          @Part("file") RequestBody file,
                          @PartMap Map<String,RequestBody> maps);
```

HeaderMap

- 作用于方法的参数,用于添加请求头
- 以map的方式添加多个请求头,map中的key为请求头的名称,value为请求头的值,且value使用String.valueOf()统一转换为String类型,
- map中每一项的键和值都不能为空,否则抛出IllegalArgumentException异常

```
@GET("/example1")
Call<ProjectBean> example1(@HeaderMap Map<String, String> headers);
/////使用///////
Map<String,String> headers = new HashMap<>();
headers.put("Accept","text/plain");
headers.put("Accept-Charset", "utf-8");

wanAndroidApi.example1(headers)
                .enqueue(new Callback<ProjectBean>() {
                    @Override
                    public void onResponse(Call<ProjectBean> call,
Response<ProjectBean> response) { }

                    @Override
                    public void onFailure(Call<ProjectBean> call, Throwable
t) { }
                });
```

Path

请求 URL 可以替换模块来动态改变，替换模块是 {}包含的字母数字字符串，替换的参数必须使用 @Path 注解的相同字符串

```
@GET("example5/{id}")
Call<ResponseBody> example5(@Path("id") int id);
```

Query

- 作用于方法的参数
- 用于添加查询参数,即请求参数
- 参数值通过String.valueOf()转换为String并进行URL编码
- 使用该注解定义的参数,参数值可以为空,为空时,忽略该值,当传入一个List或array时,为每个非空item拼接请求键值对,所有的键是统一的,如:
  name=张三&name=李四&name=王五.

```
@GET("example2/{id}")
Call<ResponseBody> example2(@Path("id") int id);
```

QueryMap

复杂的查询参数

```
@GET("example3/{id}")
Call<ResponseBody> example3(@Path("id") int id, @QueryMap Map<String,
String> options);
```

Url

- 作用于方法参数
- 用于添加请求的接口地址

```
@GET
Call<ResponseBody> example4(@Url String url);
```

原理分析

关键类功能说明

Retrofit

Retrofit提供的子系统

1. serviceMethodCache(自定义的接口映射对象集合)
2. baseUrl（请求地址）
3. callFactory（默认为OKHttpCall）
4. converterFactories（数据解析器工厂集合）
5. callAdapterFactories（Call适配器工厂集合）

6. callbackExecutor（回调执行，Android平台默认为MainThreadExecutor）

使用Builder模型构建(把对象依赖的零件创建、零件的组装封装起来；以使客户很方便的获取一个复杂对象；)

Platform

Retrofit中用来管理多平台的方法，支持Android、Java8。通过findPlatform获取对应的平台，同时也初始化了defaultCallAdapterFactory工厂

ServiceMethod

接口映射的网络请求对象，通过动态代理，将自定义接口的标注转换为该对象，将标注及参数生成OkHttp所需的Request对象。Retrofit的create通过动态代理拦截，将每一个自定义接口转换成为一个ServiceMethod对象，并通过通过serviceMethodCache进行缓存

Call

Retrofit定义的网络请求接口，包含execute、enqueue等方法

OkHttpCall

Ohttp的Call实现，通过createRawCall得到真正的 okhttp3.Call对象，用于进行实际的网络请求

CallAdapter.Factory

CallAdapter的静态工厂，包含get的抽象方法，用于生产CallAdapter对象

ExecutorCallAdapterFactory

Android平台默认的CallAdapter工厂，get方法使用匿名内部类实现CallAdapter，返回ExecutorCallbackCall，实现了Call

ExecutorCallbackCall

采用静态代理设计，delegate实际为OkHttpCall，使用callbackExecutor实现回调在主线程中执行

RxJavaCallAdapterFactory

Rxjava平台的CallAdapter工厂，get方法返回RxJavaCallAdapter对象

RxJavaCallAdapter

Rxjava平台的设配器，返回observable对象

Converter.Factory

数据解析器工厂，用于生产Converter实例

GsonConverterFactory

数据解析工厂实例，返回了GsonResponseBodyConverter数据解析器

GsonResponseBodyConverter

Gson的数据解析器，将服务端返回的json对象转换成对应的java模型

Response

Retrofit网络请求响应的Response

关键的几个流程

1. Retrofit 如何将定义的interface转换成网络请求?
2. Retrofit的Converter机制是如何实现?
3. Retrofit的CallAdapter机制是如何实现?

Converter种类

Retrofit支持多种数据解析方式，使用时需要在Gradle添加依赖。

数据解析器	Gradle依赖
Gson	com.squareup.retrofit2:converter-gson:version
Jackson	com.squareup.retrofit2:converter-jackson:version
Simple XML	com.squareup.retrofit2:converter-simplexml:version
Protobuf	com.squareup.retrofit2:converter-protobuf:version
Moshi	com.squareup.retrofit2:converter-moshi:version
Wire	com.squareup.retrofit2:converter-wire:version
Scalars	com.squareup.retrofit2:converter-scalars:version

CallAdapter种类

网络请求适配器	Gradle依赖
guava	com.squareup.retrofit2:adapter-guava:version
Java8	com.squareup.retrofit2:adapter-java8:version
rxjava	com.squareup.retrofit2:adapter-rxjava:version

如何自定义一个Converter及CallAdapter?

Retrofit中的设计模式

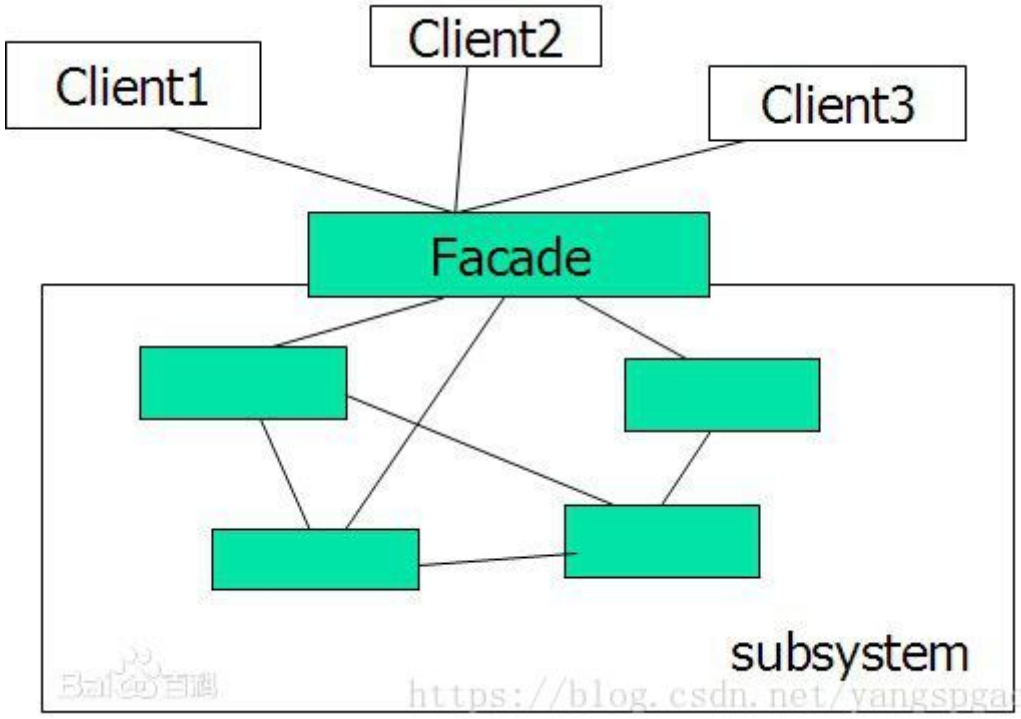
1. 建造者模式

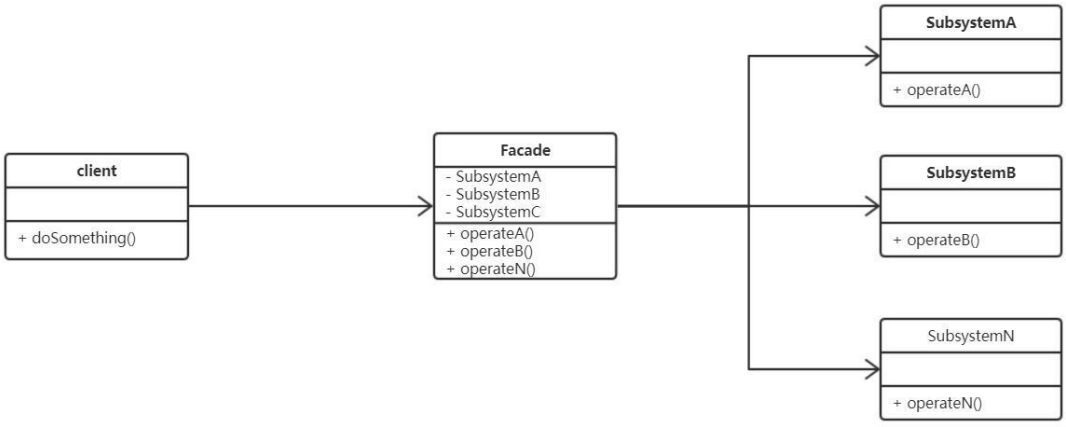
Retrofit对象的创建、ServiceMethod对象创建都使用Build模式，将复杂对象的创建和表示分离，调用者不需要知道复杂的创建过程，使用Build的相关方法进行配置创建对象。

2. 外观模式

Retrofit对外提供了统一的调度，屏蔽了内部的实现，使得使用该网络库简单便捷。

门面模式: 提供一个统一的接口去访问多个子系统的多个不同的接口，它为子系统中的一组接口提供一个统一的高层接口。使用子系统更容易使用





3. 动态代理模式

通过动态代理的方式，当调用Retrofit的create()方法时，会进行动态代理监听。当执行具体的接口方法时，会回调InvocationHandler。通过反射解析method的标注及参数，生成ServiceMethod对象。

4. 静态代理模式
   Android平台默认的适配器ExecutorCallbackCall，采用静态代理的模式。具体的实现delegate为OkHttpCall。
5. 工厂模式
   Converter及CallAdapter的创建都采用了工厂模式进行创建。
6. 适配器模式
   CallAdapter的adapt采用了适配器模式，使得interface的返回对象可以动态扩展，增强了灵活性

OkHttp, Retrofit, Volley应该如何选择?

网络请求开源库 - 介绍

网络请求库 / 基础介绍	*android-async-http*	*Volley*	*OkHttp*	*Retrofit*
作者	Loopj	Google	Square	Square
面世时间	android-async-http > Volley > OkHttp > Retrofit			
人们使用情况 *(GitHub Star数)*	Volley > android-async-http > OkHttp > Retrofit			

网络请求库- 对比

网络请求库 / 对比	*android-async-http*	*Volley*	*OkHttp*	*Retrofit*
功能	• 基于HttpClient • 在 UI 线程外、异步进行Http请求 • 在匿名回调中处理请求结果 callback使用了Android的Handler发送消息机制在创建它的线程中执行 • 自动智能请求重试 • 持久化cookie存储 保存cookie到你的应用程序的SharedPreferences	• 基于HttpUrlConnection • 封装了UIL图片加载框架，支持图片加载 • 网络请求的排序、优先级处理 • 缓存 • 多级别取消请求 • Activity和生命周期的联动（Activity结束时同时取消所有网络请求）	• 高性能Http请求库 可把它理解成是一个封装之后的类似HttpUrlConnection 的一个东西，属于同级并不是基于上述二者； • 支持 SPDY，共享同一个Socket来处理同一个服务器的所有请求； • 支持http 2.0、websocket • 支持同步、异步 • 封装了线程池、数据转换、参数使用、错误处理等 • 无缝的支持GZIP来减少数据流量 • 缓存响应数据来减少重复的网络请求 • 能从很多常用的连接问题中自动恢复 • 解决了代理服务器问题和SSL握手失败问题	• 基于OkHttp • RESTful Api设计风格 • 支持同步、异步； • 通过注解配置请求 包括请求方法，请求参数，请求头，返回值等 • 可以搭配多种Converter将获得的数据解析&序列化 支持Gson（默认）、 Jackson、Protobuf等 • 提供对 RxJava 的支持
性能	1. 作者已经停止对该项目维护； 2. Android5.0后不推荐用HttpClient； 所以不推荐在项目中使用了。	• 可拓展性好：可支持HttpClient、HttpUrlConnection和OkHttp	• 基于 NIO 和 Okio，所以性能更好：请求、处理速度快 （IO：阻塞式；NIO：非阻塞式；Okio 是 Square 公司基于 IO 和 NIO 基础上做的一个更简单、高效处理数据流的一个库）	• 性能最好，处理最快； • 扩展性差 高度封装所带来的必然后果；解析数据都是使用的统一的converter，如果服务器不能给出统一的API的形式，将很难进行处理。
开发者使用		• 封装性好：简单易用	• Api调用更加简单、方便； • 使用时需要进行多一层封装	• 简洁易用（RestfulAPI设计风格） • 代码简化（更加高度的封装性和注解用法） • 解耦的更彻底、职责更细分 • 易与其他框架联合使用（RxJava） • 使用方法较多，原理复杂，存在一定门槛
应用场景		• 适合轻量级网络交互：网络请求频繁、传输数据量小； • 不能进行大数据量的网络操作（比如下载视频、音频），所以不适合用来上传文件。	重量级网络交互场景：网络请求频繁、传输数据量大 （其实会更推荐Retrofit，反正Retrofit是基于Okhttp的）	任何场景下优先选择，特别是： 后台Api遵循RESTful的风格&项目中有使用RxJava；
备注		Volley的request和response都是把数据放到byte数组里，不支持输入输出流，把数据放到数组中，如果大文件多了，数组就会非常的大且多，消耗内存，所以不如直接返回Stream那样具备可操作性，比如下载一个大文件，不可能把整个文件都缓存到内存之后再写到文件里。	Android4.4的源码中可以看到HttpURLConnection已经替换成OkHttp实现了。所以我们更有理由相信OkHttp的强大。	