**## 框架特性说明**

\* 支持GET和POST请求

\* 基于POST的大文本数据、二进制文件上传，即通过Http Body提交

\* 普通的表单提交

\* 带有文件表单提交

\* 表单提交支持参数名重复，即在后台接收到的是数组或集合

\* 支持session保持

\* 支持链式调用

\* 支持可信任证书和自定义签名证书的https访问

\* Debug调试模式，开发环境推荐启用

\* 更多特性增加中

**## 全局配置**

框架会自动读取classpath下的`flood-okhttp.properties`配置文件，如果没有特别的要求话，一般不需要重载该配置文件。

\* `flood-okhttp.properties`配置文件的配置信息

+ connectTimeout 连接超时时间，默认设置10秒

+ readTimeout 内容读取超时时间，默认是30秒

+ writeTimeout 内容写入超时时间，默认30秒

\* 如果需要更改这些默认配置，有两种方式

+ 覆盖框架提供的配置`flood-okhttp.properties`，框架优先加载项目中的配置文件，超时时间的单位都是秒

+ 通过代码设置全局配置

\* 通过代码也可以设置全局参数，该配置操作只需要操作一次，因而可以放在项目启动时配置。

+ 设置连接超时时间 `HttpClient.Instance.setConnectTimeout(int)`

+ 设置读取超时时间 `HttpClient.Instance.setReadTimeout(int)`

+ 设置写入超时时间 `HttpClient.Instance.writeTimeout(int)`

+ 设置自定义签名证书 `HttpClient.Instance.customSSL(?)`

+ 设置默认Header `HttpClient.Instance.setDefaultHeader(?)`

\* SSL证书配置

2017年是SSL活跃率非常高的一年，所以增加了HTTPS单向认证和双向认证。HTTPS全局配置需要通过代码设置。

+ 网站启用了HTTPS，如果SSL证书由信任的Root CA发布的，那么框架自动信信任，不需要你做任何配置

+ 忽略HTTPS，即客户端不验证，非常不推荐这么使用，调用方式 `HttpClient.Instance.customSSL()`

+ 假如使用的自签证书（经典的12306网站）或系统不能自动信任的SSL证书（如Let's Encrypt） `HttpClient.Instance.customSSL(HttpClient.class.getClassLoader().getResourceAsStream("mzlion\_com.cer"))`

+ 最严格就是双向认证 `HttpClient.Instance.customSSL(InputStream pfxStream, char[] pfxPwd, InputStream... certificates)`

**## 基本示例**

**### 1.普通的GET请求无参数**

```java

String responseData = HttpClient

// 请求方式和请求url

.get("http://localhost:8080/user-sys/user/list")

.asString();

```

**### 2.普通的GET请求带参数**

```java

String responseData = HttpClient

// 请求方式和请求url

.get("http://localhost:8080/user-sys/user/list")

//设置请求参数

.queryString("mobile","18018110018")

.asString();

```

**### 3.POST普通表单提交**

```java

String responseData = HttpClient

// 请求方式和请求url

.post("http://localhost:8080/user-sys/user/add")

// 表单参数

.param("name","张三")

.param("mobile", "13023614020")

.param("langs", "Java")

.param("langs", "Python")

//url参数

//queryString("queryTime","20160530")

.asString();

```

**## 上传大文本数据、JSON类型的文本、大文件等**

一般针对POST提交数据内容较为复杂、接口约定需要POST上传时调用本方法非常有效。这种提交方式也是POST，但是数据内容和格式直接写入请求流中。比如微信接口就是这种模式。

**### 1.POST提交String**

```java

String responseData = HttpClient

// 请求方式和请求url

.textBody("http://localhost:8080/user-sys/user/body1")

.text("设施一串和服务端约定好的数据格式")

//设置编码

//.charset("utf-8")

.asString();

```

**### 2.POST提交JSON格式的文本**

```java

String responseData = HttpClient

// 请求方式和请求url

.textBody("http://localhost:8080/user-sys/user/import")

// post提交json

.json("[{\"name\": \"test-13\",\"mobile\": \"18321001200\",\"programLangs\": \"Java,Pyhton\",\"remark\": \"0\"}]")

//post提交xml

//.xml("<?xml version=\"1.0\" encoding=\"utf-8\" ?>")

//post提交html

//.html("function fun(){}")

//.charset("utf-8")

//设置编码

.asString();

```

**### 3.POST提交XML等其他格式的文本**

```java

String responseData = HttpClient

// 请求方式和请求url

.textBody("http://localhost:8080/user-sys/user/body2")

//post提交xml

.xml("<?xml version=\"1.0\" encoding=\"utf-8\" ?>")

//post提交html

//.html("function fun(){}")

//post提交一段javascript

//.javascript("function fn(){}")

//设置编码

//.charset("utf-8")

.asString();

```

**### 4.POST提交二进制文件**

```java

String responseData = HttpClient

// 请求方式和请求url

.binaryBody("http://localhost:8080/user-sys/user/body3")

// post提交流

.stream(this.getClass().getClassLoader().getResourceAsStream("avatar.png"))

//设置请求内容类型

.contentType(ContentType.IMAGE\_JPG)

//post提交文件

//.file(new File("d:/avatar.png"))

.asString();

//ContentType内置常见的MIME类型，基本上不用自己创建

```

**## 表单提交、一参数多值**

**### 1.POST表单提交含文件上传**

```java

String responseData = HttpClient

// 请求方式和请求url

.post("http://localhost:8080/user-sys/user/add")

.param("name", "李四")

.param("mobile", "13023614021")

.param("avatarFile", this.getClassLoader().getResourceAsStream("avatar.png"), "avatar.png")

//.param("avatarFile", new File("D:/avatar.png")

.asString();

```

**### 2.POST提交支持一个参数设置多个值或替换**

```java

String responseData = HttpClient

// 请求方式和请求url

.post("http://localhost:8080/user-sys/user/add")

// 表单参数

.param("name","张三")

.param("mobile", "13023614020")

.param("langs", "Java")

.param("langs", "Python")//会多种语言

.asString();

```

**## HttpResponse对象介绍**

以上展示的代码都是基于同步请求的，并且你会发现示例代码中总是出现`asString()`方法，这个方法来自`HttpResponse`类，

目前`as\*\*\*`方法已经提升和`execute`方法级别一样，用法和`HttpResponse`提供的方法一样。

1. `HttpResponse.isSuccess()`表示请求是否成功，只有请求成功才会有后面的功能。

2. `HttpResponse.getErrorMessage` 请求失败时错误消息。

**\*\*下面就是非常非常非常好用的方法\*\***

1. `public String asString()` 将响应结果直接转为字符串

2. `public <E> E asBean(Class<E> targetClass)` 将响应结果转为JavaBean

3. `public <E> E asBean(TypeRef<E> typeRef)` 也是将响应结果转为JavaBean，和上面的区别在于该方法能够提取到泛型信息

4. `public byte[] asByteData()` 将响应结果转为二进制内容，这是数据在网络请求的原始数据

5. `public void asFile(File saveFile)` 将响应结果转为文件存储，当远程是文件时该方法非常有用

6. `public void asStream(OutputStream out)` 直接将响应结果输出到另外一个流中

7. `public <T> T custom(DataHandler<T> dataHandler)` 当以上这些方法都满足不了你的话，这个方法可以自己DIY，随你怎么蹂躏

下面给出一些简单的代码，这些方法已经非常简单了。

```java

//将响应结果转为字符串输出

String responseData = httpResponse.asString();

//将响应结果转为文件保存

File frc = new File("d:\\web\\save.txt");

httpResponse.asFile(frc);

//json转换器

List<Person> personList = httpResponse.asBean(new TypeToken<List<Person>>(){});

//重载方法

//Person person = httpResponse.asBean(Person.class);

//将响应结果转为输出流中

ByteArrayOutputStream baos = new ByteArrayOutputStream();

httpResponse.asStream(baos);

```

**## HTTPS**

前面已经提到HTTPS的配置，下面给出一些简单的示例更直观了解用法。

```java

//由信任CA机构发布的，比如GitHub的https，框架不需要你做什么事情，正常使用即可

String githubContent = HttpClient

.get("https://www.mzlion.com")

.asString();

//不管三七二十几，直接忽略HTTPS

String mzlionIndexContent = HttpClient

.get("https://kyfw.12306.cn/otn/")

.customSSL()

.asString();

//自签SSL或程序不认可实际安全的，可以指定客户端证书

String mzlionIndexContent = HttpClient

.get("https://kyfw.12306.cn/otn/")

.customSSL(this.getClass().getClassLoader().getResourceAsStream("SRCA.cer"))

.asString();

```

**## 数据处理器DataHandler<T>**

这是新版本的新增的一个功能，数据处理器定义接口返回数据处理能力，该接口就只有一个函数

```java

T handle(final okhttp3.Response response) throws IOException;

```

该函数提供了将原始数据格式转为业务所需数据格式，框架提供了常用接口处理器实现，若以下的处理器无法满足需求可以自己实现一个处理器。

\* `StringDataHandler` 字符串处理器，该处理器比较常用，可以直接调用`StringDataHandler.create()`得到处理器的实例

\* `JsonDataHandler<T>` JSON处理器，该处理器也比较常用，即将JSON字符串转为Javabean

\* `FileDataHandler` 文件处理器，将结果存储到文件中，该处理器有两个构造参数，其中`dirPath`保存目录必填，`filename`保存的文件名可选，如果为空时框架自动从header、url中获取，如果获取不到则随机生成一个文件名。

**## 高级配置**

**### 异步请求**

异步请求不会阻塞当前线程（特别是网络慢的时候），适用于对返回结果不关心或不需要立即知晓的情况下，比如推送、通知等。

异步请求只有在执行网络请求的时候有一点区别，其他地方和同步请求配置和操作都是一样的。

```java

String githubContent = HttpClient

.get("https://www.github.com")

.execute(new CallbackAdaptor<String>(){

@Override

public DataHandler<String> getDataHandler() {

return StringDataHandler.create();

}

@Override

public void onSuccess(String data) {

//data就是经过处理后的数据，直接在这里写自己的业务逻辑

}

});

```

**### Callback回调接口**

`Callback`是回调定义接口，里面总共定义了6个函数，每个函数被调用的顺序不一样。

\* `onBefore()` 第一被调用，主要在请求网络之前，这个函数有返回值，如果返回`false`则阻止此次请求了；

\* `postProgress()` 第二被调用，上传进度回调函数

\* `onError()` 第三被调用，当只有请求失败时才会触发；

\* `onComplete()` 第四被调用，当请求接口完成后触发该函数；

\* `onSuccess()` 第五被调用，当请求接口成功（HTTP状态码为200）则会触发该函数，

该函数会依赖另外一个函数`getDataHandler()`，返回一个指定的数据处理器，处理原始数据。对于数据处理器前面已经了解过了。

异步回调接口`Callback`总共定义了6个函数，但是一般不会关心所有函数处理情况，所以提供了`CallbackAdaptor`空实现类，想要关注哪个函数的执行结果，重载那个函数即可。

**### 为单个请求设置超时**

当我们需要对单个请求设置连接超时时间、读取超时时间等属性时，可以在执行`execute`方法之前调用。主要有如下几个方法可以进行调用。

+ `connectTimeout(int)` 连接超时时间

+ `readTimeout(int)` 读取超时时间

+ `writeTimeout(int)` 写入超时时间

+ `customSSL()` 设置https证书

**## 关单单例说明**

一般一个项目只请求一个远程服务，所以这些配置可以在全局配置，共享一个`OkHttpClient`对象(相当于共享一个浏览器)，

以上这些方法的调用会使得框架创建一个新的`OkHttpClient`对象(相当于又打开了一个浏览器)。下面给出一个snippet加以说明。