作者：伍陆七  
链接：<https://juejin.cn/post/6898485806587969544>   
来源：掘金  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 前言

Retrofit是适用于Android和Java且类型安全的HTTP客户端，其最大的特性的是**支持通过接口的方式发起HTTP请求**。而spring-boot是使用最广泛的Java开发框架，但是Retrofit官方没有支持与spring-boot框架快速整合，因此我们开发了retrofit-spring-boot-starter。

**retrofit-spring-boot-starter实现了Retrofit与spring-boot框架快速整合，并且支持了诸多功能增强，极大简化开发**。

🚀项目持续优化迭代，欢迎大家提ISSUE和PR！麻烦大家能给一颗star✨，您的star是我们持续更新的动力！

## 功能特性

* [自定义注入OkHttpClient](#%E8%87%AA%E5%AE%9A%E4%B9%89%E6%B3%A8%E5)
* [注解式拦截器](#%E6%B3%A8%E8%A7%A3%E5%BC%8F%E6%8B%A6%E6)
* [连接池管理](#%E8%BF%9E%E6%8E%A5%E6%B1%A0%E7%AE%A1%E7)
* [日志打印](#%E6%97%A5%E5%BF%97%E6%89%93%E5%8D%B0)
* [请求重试](#%E8%AF%B7%E6%B1%82%E9%87%8D%E8%AF%95)
* [错误解码器](#%E9%94%99%E8%AF%AF%E8%A7%A3%E7%A0%81%E5)
* [全局拦截器](#%E5%85%A8%E5%B1%80%E6%8B%A6%E6%88%AA%E5)
* [熔断降级](#%E7%86%94%E6%96%AD%E9%99%8D%E7%BA%A7)
* [微服务之间的HTTP调用](#%E5%BE%AE%E6%9C%8D%E5%8A%A1%E4%B9%8B%E9)
* [调用适配器](#%E8%B0%83%E7%94%A8%E9%80%82%E9%85%8D%E5)
* [数据转换器](#%E6%95%B0%E6%8D%AE%E8%BD%AC%E7%A0%81%E5)

## 快速使用

### 引入依赖

<dependency>

<groupId>com.github.lianjiatech</groupId>

<artifactId>retrofit-spring-boot-starter</artifactId>

<version>2.2.2</version>

</dependency>

复制代码

### 定义http接口

**接口必须使用@RetrofitClient注解标记**！http相关注解可参考官方文档：[retrofit官方文档](https://square.github.io/retrofit/" \t "_blank)。

@RetrofitClient(baseUrl = "${test.baseUrl}")

public interface HttpApi {

@GET("person")

Result<Person> getPerson(@Query("id") Long id);

}

复制代码

### 注入使用

**将接口注入到其它Service中即可使用！**

@Service

public class TestService {

@Autowired

private HttpApi httpApi;

public void test() {

// 通过httpApi发起http请求

}

}

复制代码

## HTTP请求相关注解

HTTP请求相关注解，全部使用了retrofit原生注解。**详细信息可参考官方文档：**[**retrofit官方文档**](https://square.github.io/retrofit/)，以下是一个简单说明。

| **注解分类** | **支持的注解** |
| --- | --- |
| 请求方式 | @GET @HEAD @POST @PUT @DELETE @OPTIONS |
| 请求头 | @Header @HeaderMap @Headers |
| Query参数 | @Query @QueryMap @QueryName |
| path参数 | @Path |
| form-encoded参数 | @Field @FieldMap @FormUrlEncoded |
| 文件上传 | @Multipart @Part @PartMap |
| url参数 | @Url |

## 配置项说明

retrofit-spring-boot-starter支持了多个可配置的属性，用来应对不同的业务场景。您可以视情况进行修改，具体说明如下：

| **配置** | **默认值** | **说明** |
| --- | --- | --- |
| enable-log | true | 启用日志打印 |
| logging-interceptor | DefaultLoggingInterceptor | 日志打印拦截器 |
| pool |  | 连接池配置 |
| disable-void-return-type | false | 禁用java.lang.Void返回类型 |
| retry-interceptor | DefaultRetryInterceptor | 请求重试拦截器 |
| global-converter-factories | JacksonConverterFactory | 全局转换器工厂 |
| global-call-adapter-factories | BodyCallAdapterFactory,ResponseCallAdapterFactory | 全局调用适配器工厂 |
| enable-degrade | false | 是否启用熔断降级 |
| degrade-type | sentinel | 熔断降级实现方式(目前仅支持Sentinel) |
| resource-name-parser | DefaultResourceNameParser | 熔断资源名称解析器，用于解析资源名称 |

yml配置方式：

retrofit:

enable-response-call-adapter: true

# 启用日志打印

enable-log: true

# 连接池配置

pool:

test1:

max-idle-connections: 3

keep-alive-second: 100

test2:

max-idle-connections: 5

keep-alive-second: 50

# 禁用void返回值类型

disable-void-return-type: false

# 日志打印拦截器

logging-interceptor: com.github.lianjiatech.retrofit.spring.boot.interceptor.DefaultLoggingInterceptor

# 请求重试拦截器

retry-interceptor: com.github.lianjiatech.retrofit.spring.boot.retry.DefaultRetryInterceptor

# 全局转换器工厂

global-converter-factories:

- retrofit2.converter.jackson.JacksonConverterFactory

# 全局调用适配器工厂

global-call-adapter-factories:

- com.github.lianjiatech.retrofit.spring.boot.core.BodyCallAdapterFactory

- com.github.lianjiatech.retrofit.spring.boot.core.ResponseCallAdapterFactory

# 是否启用熔断降级

enable-degrade: true

# 熔断降级实现方式

degrade-type: sentinel

# 熔断资源名称解析器

resource-name-parser: com.github.lianjiatech.retrofit.spring.boot.degrade.DefaultResourceNameParser

## 高级功能

### 自定义注入OkHttpClient

通常情况下，通过@RetrofitClient注解属性动态创建OkHttpClient对象能够满足大部分使用场景。但是在某些情况下，用户可能需要自定义OkHttpClient，这个时候，可以在接口上定义返回类型是OkHttpClient.Builder的静态方法来实现。代码示例如下：

@RetrofitClient(baseUrl = "http://ke.com")

public interface HttpApi3 {

@OkHttpClientBuilder

static OkHttpClient.Builder okhttpClientBuilder() {

return new OkHttpClient.Builder()

.connectTimeout(1, TimeUnit.SECONDS)

.readTimeout(1, TimeUnit.SECONDS)

.writeTimeout(1, TimeUnit.SECONDS);

}

@GET

Result<Person> getPerson(@Url String url, @Query("id") Long id);

}

复制代码

方法必须使用@OkHttpClientBuilder注解标记！

### 注解式拦截器

很多时候，我们希望某个接口下的某些http请求执行统一的拦截处理逻辑。为了支持这个功能，retrofit-spring-boot-starter提供了**注解式拦截器**，做到了**基于url路径的匹配拦截**。使用的步骤主要分为2步：

1. 继承BasePathMatchInterceptor编写拦截处理器；
2. 接口上使用@Intercept进行标注。如需配置多个拦截器，在接口上标注多个@Intercept注解即可！

下面以给指定请求的url后面拼接timestamp时间戳为例，介绍下如何使用注解式拦截器。

#### 继承BasePathMatchInterceptor编写拦截处理器

@Component

public class TimeStampInterceptor extends BasePathMatchInterceptor {

@Override

public Response doIntercept(Chain chain) throws IOException {

Request request = chain.request();

HttpUrl url = request.url();

long timestamp = System.currentTimeMillis();

HttpUrl newUrl = url.newBuilder()

.addQueryParameter("timestamp", String.valueOf(timestamp))

.build();

Request newRequest = request.newBuilder()

.url(newUrl)

.build();

return chain.proceed(newRequest);

}

}

复制代码

#### 接口上使用@Intercept进行标注

@RetrofitClient(baseUrl = "${test.baseUrl}")

@Intercept(handler = TimeStampInterceptor.class, include = {"/api/\*\*"}, exclude = "/api/test/savePerson")

public interface HttpApi {

@GET("person")

Result<Person> getPerson(@Query("id") Long id);

@POST("savePerson")

Result<Person> savePerson(@Body Person person);

}

复制代码

上面的@Intercept配置表示：拦截HttpApi接口下/api/\*\*路径下（排除/api/test/savePerson）的请求，拦截处理器使用TimeStampInterceptor。

### 扩展注解式拦截器

有的时候，我们需要在**拦截注解**动态传入一些参数，然后再执行拦截的时候需要使用这个参数。这种时候，我们可以扩展实现**自定义拦截注解**。自定义拦截注解必须使用@InterceptMark标记，并且**注解中必须包括include()、exclude()、handler()属性信息**。使用的步骤主要分为3步：

1. 自定义拦截注解
2. 继承BasePathMatchInterceptor编写拦截处理器
3. 接口上使用自定义拦截注解；

例如我们需要**在请求头里面动态加入accessKeyId、accessKeySecret签名信息才能正常发起http请求**，这个时候**可以自定义一个加签拦截器注解@Sign来实现**。下面以自定义@Sign拦截注解为例进行说明。

#### 自定义@Sign注解

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

@Documented

@InterceptMark

public @interface Sign {

/\*\*

\* 密钥key

\* 支持占位符形式配置。

\*

\* @return

\*/

String accessKeyId();

/\*\*

\* 密钥

\* 支持占位符形式配置。

\*

\* @return

\*/

String accessKeySecret();

/\*\*

\* 拦截器匹配路径

\*

\* @return

\*/

String[] include() default {"/\*\*"};

/\*\*

\* 拦截器排除匹配，排除指定路径拦截

\*

\* @return

\*/

String[] exclude() default {};

/\*\*

\* 处理该注解的拦截器类

\* 优先从spring容器获取对应的Bean，如果获取不到，则使用反射创建一个！

\*

\* @return

\*/

Class<? extends BasePathMatchInterceptor> handler() default SignInterceptor.class;

}

复制代码

扩展自定义拦截注解有以下2点需要注意：

1. 自定义拦截注解必须使用@InterceptMark标记。
2. 注解中必须包括include()、exclude()、handler()属性信息。

#### 实现SignInterceptor

@Component

public class SignInterceptor extends BasePathMatchInterceptor {

private String accessKeyId;

private String accessKeySecret;

public void setAccessKeyId(String accessKeyId) {

this.accessKeyId = accessKeyId;

}

public void setAccessKeySecret(String accessKeySecret) {

this.accessKeySecret = accessKeySecret;

}

@Override

public Response doIntercept(Chain chain) throws IOException {

Request request = chain.request();

Request newReq = request.newBuilder()

.addHeader("accessKeyId", accessKeyId)

.addHeader("accessKeySecret", accessKeySecret)

.build();

return chain.proceed(newReq);

}

}

复制代码

**上述accessKeyId和accessKeySecret字段值会依据@Sign注解的accessKeyId()和accessKeySecret()值自动注入，如果@Sign指定的是占位符形式的字符串，则会取配置属性值进行注入**。另外，**accessKeyId和accessKeySecret字段必须提供setter方法**。

#### 接口上使用@Sign

@RetrofitClient(baseUrl = "${test.baseUrl}")

@Sign(accessKeyId = "${test.accessKeyId}", accessKeySecret = "${test.accessKeySecret}", exclude = {"/api/test/person"})

public interface HttpApi {

@GET("person")

Result<Person> getPerson(@Query("id") Long id);

@POST("savePerson")

Result<Person> savePerson(@Body Person person);

}

复制代码

这样就能在指定url的请求上，自动加上签名信息了。

### 连接池管理

默认情况下，所有通过Retrofit发送的http请求都会使用max-idle-connections=5 keep-alive-second=300的默认连接池。当然，我们也可以在配置文件中配置多个自定义的连接池，然后通过@RetrofitClient的poolName属性来指定使用。比如我们要让某个接口下的请求全部使用poolName=test1的连接池，代码实现如下：

1. 配置连接池。
2. retrofit:
3. # 连接池配置
4. pool:
5. test1:
6. max-idle-connections: 3
7. keep-alive-second: 100
8. test2:
9. max-idle-connections: 5
10. keep-alive-second: 50

复制代码

1. 通过@RetrofitClient的poolName属性来指定使用的连接池。
2. @RetrofitClient(baseUrl = "${test.baseUrl}", poolName="test1")
3. public interface HttpApi {
4. @GET("person")
5. Result<Person> getPerson(@Query("id") Long id);
6. }

复制代码

### 日志打印

很多情况下，我们希望将http请求日志记录下来。通过retrofit.enableLog配置可以全局控制日志是否开启。 针对每个接口，可以通过@RetrofitClient的enableLog控制是否开启，通过logLevel和logStrategy，可以指定每个接口的日志打印级别以及日志打印策略。retrofit-spring-boot-starter支持了5种日志打印级别(ERROR, WARN, INFO, DEBUG, TRACE)，默认INFO；支持了4种日志打印策略（NONE, BASIC, HEADERS, BODY），默认BASIC。4种日志打印策略含义如下：

1. NONE：No logs.
2. BASIC：Logs request and response lines.
3. HEADERS：Logs request and response lines and their respective headers.
4. BODY：Logs request and response lines and their respective headers and bodies (if present).

retrofit-spring-boot-starter默认使用了DefaultLoggingInterceptor执行真正的日志打印功能，其底层就是okhttp原生的HttpLoggingInterceptor。当然，你也可以自定义实现自己的日志打印拦截器，只需要继承BaseLoggingInterceptor（具体可以参考DefaultLoggingInterceptor的实现），然后在配置文件中进行相关配置即可。

retrofit:

# 日志打印拦截器

logging-interceptor: com.github.lianjiatech.retrofit.spring.boot.interceptor.DefaultLoggingInterceptor

复制代码

### 请求重试

retrofit-spring-boot-starter支持请求重试功能，只需要在接口或者方法上加上@Retry注解即可。**@Retry支持重试次数maxRetries、重试时间间隔intervalMs以及重试规则retryRules配置**。重试规则支持三种配置：

1. RESPONSE\_STATUS\_NOT\_2XX：响应状态码不是2xx时执行重试；
2. OCCUR\_IO\_EXCEPTION：发生IO异常时执行重试；
3. OCCUR\_EXCEPTION：发生任意异常时执行重试；

默认响应状态码不是2xx或者发生IO异常时自动进行重试。需要的话，你也可以继承BaseRetryInterceptor实现自己的请求重试拦截器，然后将其配置上去。

retrofit:

# 请求重试拦截器

retry-interceptor: com.github.lianjiatech.retrofit.spring.boot.retry.DefaultRetryInterceptor

复制代码

### 错误解码器

在HTTP发生请求错误(包括发生异常或者响应数据不符合预期)的时候，错误解码器可将HTTP相关信息解码到自定义异常中。你可以在@RetrofitClient注解的errorDecoder()指定当前接口的错误解码器，自定义错误解码器需要实现ErrorDecoder接口：

/\*\*

\* 错误解码器。ErrorDecoder.

\* 当请求发生异常或者收到无效响应结果的时候，将HTTP相关信息解码到异常中，无效响应由业务自己判断

\*

\* When an exception occurs in the request or an invalid response result is received, the HTTP related information is decoded into the exception,

\* and the invalid response is determined by the business itself.

\*

\* @author 陈添明

\*/

public interface ErrorDecoder {

/\*\*

\* 当无效响应的时候，将HTTP信息解码到异常中，无效响应由业务自行判断。

\* When the response is invalid, decode the HTTP information into the exception, invalid response is determined by business.

\*

\* @param request request

\* @param response response

\* @return If it returns null, the processing is ignored and the processing continues with the original response.

\*/

default RuntimeException invalidRespDecode(Request request, Response response) {

if (!response.isSuccessful()) {

throw RetrofitException.errorStatus(request, response);

}

return null;

}

/\*\*

\* 当请求发生IO异常时，将HTTP信息解码到异常中。

\* When an IO exception occurs in the request, the HTTP information is decoded into the exception.

\*

\* @param request request

\* @param cause IOException

\* @return RuntimeException

\*/

default RuntimeException ioExceptionDecode(Request request, IOException cause) {

return RetrofitException.errorExecuting(request, cause);

}

/\*\*

\* 当请求发生除IO异常之外的其它异常时，将HTTP信息解码到异常中。

\* When the request has an exception other than the IO exception, the HTTP information is decoded into the exception.

\*

\* @param request request

\* @param cause Exception

\* @return RuntimeException

\*/

default RuntimeException exceptionDecode(Request request, Exception cause) {

return RetrofitException.errorUnknown(request, cause);

}

}

复制代码

## 全局拦截器

### 全局应用拦截器

如果我们需要对整个系统的的http请求执行统一的拦截处理，可以自定义实现全局拦截器BaseGlobalInterceptor, 并配置成spring容器中的bean！例如我们需要在整个系统发起的http请求，都带上来源信息。

@Component

public class SourceInterceptor extends BaseGlobalInterceptor {

@Override

public Response doIntercept(Chain chain) throws IOException {

Request request = chain.request();

Request newReq = request.newBuilder()

.addHeader("source", "test")

.build();

return chain.proceed(newReq);

}

}

复制代码

### 全局网络拦截器

只需要实现NetworkInterceptor接口 并配置成spring容器中的bean就支持自动织入全局网络拦截器。

### 熔断降级

在分布式服务架构中，对不稳定的外部服务进行熔断降级是保证服务高可用的重要措施之一。由于外部服务的稳定性是不能保证的，当外部服务不稳定时，响应时间会变长。相应地，调用方的响应时间也会变长，线程会产生堆积，最终可能耗尽调用方的线程池，导致整个服务不可用。因此我们需要对不稳定的弱依赖服务调用进行熔断降级，暂时切断不稳定调用，避免局部不稳定导致整体服务雪崩。

retrofit-spring-boot-starter支持熔断降级功能，底层基于[Sentinel](https://sentinelguard.io/zh-cn/docs/introduction.html" \t "_blank)实现。具体来说，支持了**熔断资源自发现**和**注解式降级规则配置**。如需使用熔断降级，只需要进行以下操作即可：

#### 1. 开启熔断降级功能

**默认情况下，熔断降级功能是关闭的，需要设置相应的配置项来开启熔断降级功能**：

retrofit:

# 是否启用熔断降级

enable-degrade: true

# 熔断降级实现方式(目前仅支持Sentinel)

degrade-type: sentinel

# 资源名称解析器

resource-name-parser: com.github.lianjiatech.retrofit.spring.boot.degrade.DefaultResourceNameParser

复制代码

资源名称解析器用于实现用户自定义资源名称，默认配置是DefaultResourceNameParser，对应的资源名称格式为HTTP\_OUT:GET:http://localhost:8080/api/degrade/test。用户可以继承BaseResourceNameParser类实现自己的资源名称解析器。

另外，由于熔断降级功能是可选的，**因此启用熔断降级需要用户自行引入Sentinel依赖**：

<dependency>

<groupId>com.alibaba.csp</groupId>

<artifactId>sentinel-core</artifactId>

<version>1.6.3</version>

</dependency>

复制代码

#### 2. 配置降级规则（可选）

**retrofit-spring-boot-starter支持注解式配置降级规则，通过@Degrade注解来配置降级规则**。@Degrade注解可以配置在接口或者方法上，配置在方法上的优先级更高。

@Retention(RetentionPolicy.RUNTIME)

@Target({ElementType.METHOD, ElementType.TYPE})

@Documented

public @interface Degrade {

/\*\*

\* RT threshold or exception ratio threshold count.

\*/

double count();

/\*\*

\* Degrade recover timeout (in seconds) when degradation occurs.

\*/

int timeWindow() default 5;

/\*\*

\* Degrade strategy (0: average RT, 1: exception ratio).

\*/

DegradeStrategy degradeStrategy() default DegradeStrategy.AVERAGE\_RT;

}

复制代码

**如果应用项目已支持通过配置中心配置降级规则，可忽略注解式配置方式**。

#### 3. @RetrofitClient设置fallback或者fallbackFactory (可选)

如果@RetrofitClient不设置fallback或者fallbackFactory，当触发熔断时，会直接抛出RetrofitBlockException异常。**用户可以通过设置fallback或者fallbackFactory来定制熔断时的方法返回值**。fallback类必须是当前接口的实现类，fallbackFactory必须是FallbackFactory<T>实现类，泛型参数类型为当前接口类型。另外，fallback和fallbackFactory实例必须配置成Spring容器的Bean。

**fallbackFactory相对于fallback，主要差别在于能够感知每次熔断的异常原因(cause)**。参考示例如下：

@Slf4j

@Service

public class HttpDegradeFallback implements HttpDegradeApi {

@Override

public Result<Integer> test() {

Result<Integer> fallback = new Result<>();

fallback.setCode(100)

.setMsg("fallback")

.setBody(1000000);

return fallback;

}

}

复制代码

@Slf4j

@Service

public class HttpDegradeFallbackFactory implements FallbackFactory<HttpDegradeApi> {

/\*\*

\* Returns an instance of the fallback appropriate for the given cause

\*

\* @param cause fallback cause

\* @return 实现了retrofit接口的实例。an instance that implements the retrofit interface.

\*/

@Override

public HttpDegradeApi create(Throwable cause) {

log.error("触发熔断了! ", cause.getMessage(), cause);

return new HttpDegradeApi() {

@Override

public Result<Integer> test() {

Result<Integer> fallback = new Result<>();

fallback.setCode(100)

.setMsg("fallback")

.setBody(1000000);

return fallback;

}

}

}

复制代码

### 微服务之间的HTTP调用

为了能够使用微服务调用，需要进行如下配置：

#### 配置ServiceInstanceChooser为Spring容器Bean

用户可以自行实现ServiceInstanceChooser接口，完成服务实例的选取逻辑，并将其配置成Spring容器的Bean。对于Spring Cloud应用，retrofit-spring-boot-starter提供了SpringCloudServiceInstanceChooser实现，用户只需将其配置成Spring的Bean即可。

@Bean

@Autowired

public ServiceInstanceChooser serviceInstanceChooser(LoadBalancerClient loadBalancerClient) {

return new SpringCloudServiceInstanceChooser(loadBalancerClient);

}

复制代码

#### 使用@Retrofit的serviceId和path属性，可以实现微服务之间的HTTP调用

@RetrofitClient(serviceId = "${jy-helicarrier-api.serviceId}", path = "/m/count", errorDecoder = HelicarrierErrorDecoder.class)

@Retry

public interface ApiCountService {

}

复制代码

## 调用适配器和数据转码器

### 调用适配器

Retrofit可以通过调用适配器CallAdapterFactory将Call<T>对象适配成接口方法的返回值类型。retrofit-spring-boot-starter扩展2种CallAdapterFactory实现：

1. BodyCallAdapterFactory
   * 默认启用，可通过配置retrofit.enable-body-call-adapter=false关闭
   * 同步执行http请求，将响应体内容适配成接口方法的返回值类型实例。
   * 除了Retrofit.Call<T>、Retrofit.Response<T>、java.util.concurrent.CompletableFuture<T>之外，其它返回类型都可以使用该适配器。
2. ResponseCallAdapterFactory
   * 默认启用，可通过配置retrofit.enable-response-call-adapter=false关闭
   * 同步执行http请求，将响应体内容适配成Retrofit.Response<T>返回。
   * 如果方法的返回值类型为Retrofit.Response<T>，则可以使用该适配器。

**Retrofit自动根据方法返回值类型选用对应的CallAdapterFactory执行适配处理！加上Retrofit默认的CallAdapterFactory，可支持多种形式的方法返回值类型：**

* Call<T>: 不执行适配处理，直接返回Call<T>对象
* CompletableFuture<T>: 将响应体内容适配成CompletableFuture<T>对象返回
* Void: 不关注返回类型可以使用Void。如果http状态码不是2xx，直接抛错！
* Response<T>: 将响应内容适配成Response<T>对象返回
* 其他任意Java类型： 将响应体内容适配成一个对应的Java类型对象返回，如果http状态码不是2xx，直接抛错！

/\*\*

\* Call<T>

\* 不执行适配处理，直接返回Call<T>对象

\* @param id

\* @return

\*/

@GET("person")

Call<Result<Person>> getPersonCall(@Query("id") Long id);

/\*\*

\* CompletableFuture<T>

\* 将响应体内容适配成CompletableFuture<T>对象返回

\* @param id

\* @return

\*/

@GET("person")

CompletableFuture<Result<Person>> getPersonCompletableFuture(@Query("id") Long id);

/\*\*

\* Void

\* 不关注返回类型可以使用Void。如果http状态码不是2xx，直接抛错！

\* @param id

\* @return

\*/

@GET("person")

Void getPersonVoid(@Query("id") Long id);

/\*\*

\* Response<T>

\* 将响应内容适配成Response<T>对象返回

\* @param id

\* @return

\*/

@GET("person")

Response<Result<Person>> getPersonResponse(@Query("id") Long id);

/\*\*

\* 其他任意Java类型

\* 将响应体内容适配成一个对应的Java类型对象返回，如果http状态码不是2xx，直接抛错！

\* @param id

\* @return

\*/

@GET("person")

Result<Person> getPerson(@Query("id") Long id);

复制代码

**我们也可以通过继承CallAdapter.Factory扩展实现自己的CallAdapter**！

retrofit-spring-boot-starter支持通过retrofit.global-call-adapter-factories配置全局调用适配器工厂，工厂实例优先从Spring容器获取，如果没有获取到，则反射创建。默认的全局调用适配器工厂是[BodyCallAdapterFactory, ResponseCallAdapterFactory]！

retrofit:

# 全局调用适配器工厂

global-call-adapter-factories:

- com.github.lianjiatech.retrofit.spring.boot.core.BodyCallAdapterFactory

- com.github.lianjiatech.retrofit.spring.boot.core.ResponseCallAdapterFactory

复制代码

针对每个Java接口，还可以通过@RetrofitClient注解的callAdapterFactories()指定当前接口采用的CallAdapter.Factory，指定的工厂实例依然优先从Spring容器获取。

**注意：如果CallAdapter.Factory没有public的无参构造器，请手动将其配置成Spring容器的Bean对象**！

### 数据转码器

Retrofit使用Converter将@Body注解标注的对象转换成请求体，将响应体数据转换成一个Java对象，可以选用以下几种Converter：

* [Gson](https://github.com/google/gson): com.squareup.Retrofit:converter-gson
* [Jackson](https://github.com/FasterXML/jackson): com.squareup.Retrofit:converter-jackson
* [Moshi](https://github.com/square/moshi/): com.squareup.Retrofit:converter-moshi
* [Protobuf](https://developers.google.com/protocol-buffers/): com.squareup.Retrofit:converter-protobuf
* [Wire](https://github.com/square/wire): com.squareup.Retrofit:converter-wire
* [Simple XML](http://simple.sourceforge.net/): com.squareup.Retrofit:converter-simplexml
* [JAXB](https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html): com.squareup.retrofit2:converter-jaxb

retrofit-spring-boot-starter支持通过retrofit.global-converter-factories配置全局数据转换器工厂，转换器工厂实例优先从Spring容器获取，如果没有获取到，则反射创建。默认的全局数据转换器工厂是retrofit2.converter.jackson.JacksonConverterFactory，你可以直接通过spring.jackson.\*配置jackson序列化规则，配置可参考[Customize the Jackson ObjectMapper](https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE/reference/htmlsingle/" \l "howto-customize-the-jackson-objectmapper" \t "_blank)！

retrofit:

# 全局转换器工厂

global-converter-factories:

- retrofit2.converter.jackson.JacksonConverterFactory

复制代码

针对每个Java接口，还可以通过@RetrofitClient注解的converterFactories()指定当前接口采用的Converter.Factory，指定的转换器工厂实例依然优先从Spring容器获取。

**注意：如果Converter.Factory没有public的无参构造器，请手动将其配置成Spring容器的Bean对象**！

## 总结

retrofit-spring-boot-starter一个适用于SpringBoot项目的轻量级HTTP客户端框架，已在线上稳定运行一年多，并且已经有多个外部公司也接入使用。有兴趣的朋友可以尝试一下，如有任何问题，欢迎提issue或者加QQ群（806714302）反馈，极速响应支持。

作者：伍陆七  
链接：https://juejin.cn/post/6898485806587969544  
来源：掘金  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。