Spring Cloud Feign设计原理

亦山札记 (/u/802bbe244ebb) + 关注
1.3 2018.09.26 17:44 字数 1906 阅读 3741 评论 2 喜欢 51 赞赏 1
(/u/802bbe244ebb)

什么是Feign?

Feign 的英文表意为“假装，伪装，变形”， 是一个http请求调用的轻量级框架，可以以Java接口注解的方式调用Http请求，而不用像Java中通过封装HTTP请求报文的方式直接调用。Feign通过处理注解，将请求模板化，当实际调用的时候，传入参数，根据参数再应用到请求上，进而转化成真正的请求，这种请求相对而言比较直观。
Feign被广泛应用在Spring Cloud 的解决方案中，是学习基于Spring Cloud 微服务架构不可或缺的重要组件。
开源项目地址：
https://github.com/OpenFeign/feign (https://github.com/OpenFeign/feign)

Feign解决了什么问题?

封装了Http调用流程，更适合面向接口化的变成习惯
在服务调用的场景中，我们经常调用基于Http协议的服务，而我们经常使用到的框架可能有HttpURLConnection、Apache HttpComponnets、OkHttp3 、Netty等等，这些框架在基于自身的专注点提供了自身特性。而从角色划分上来看，他们的职能是一致的提供Http调用服务。具体流程如下：

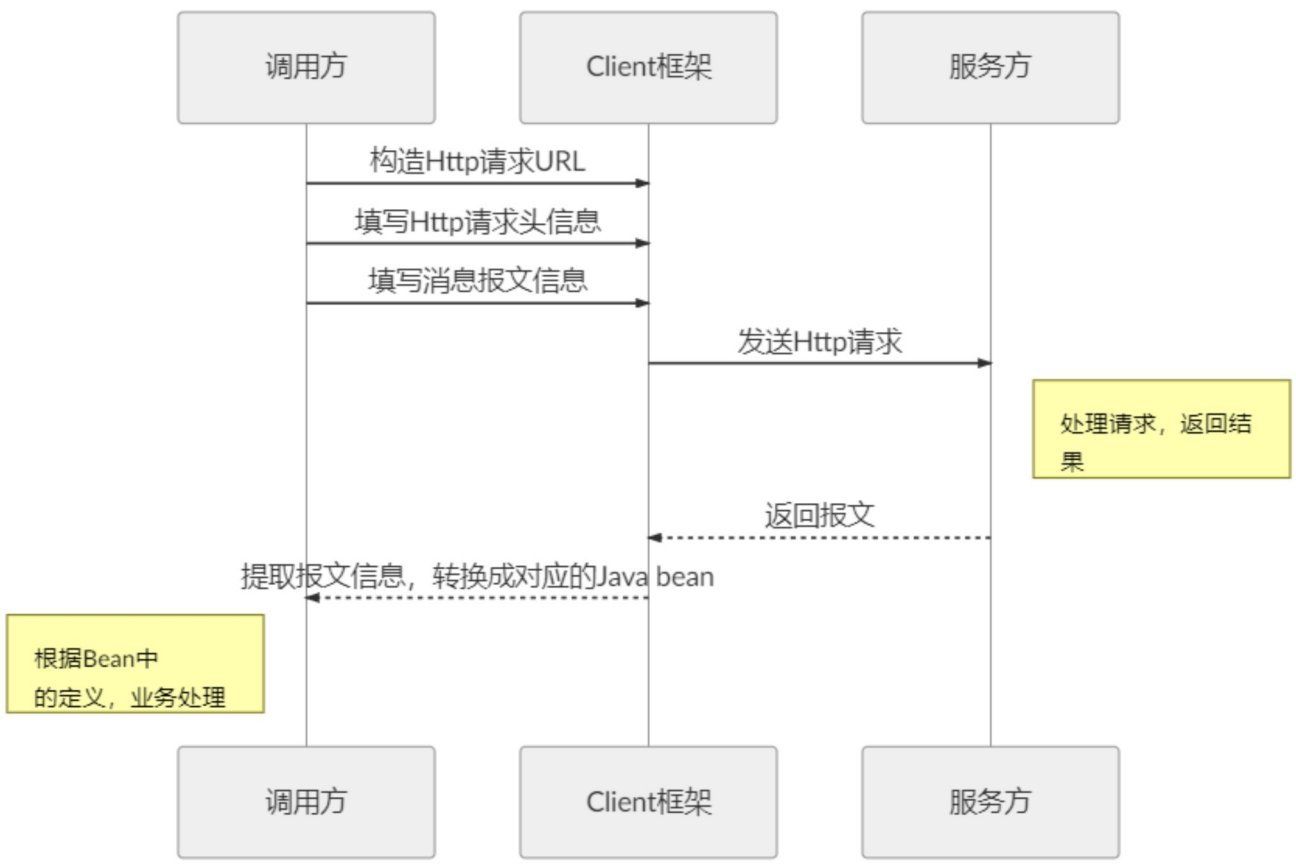


image.png

Feign是如何设计的?

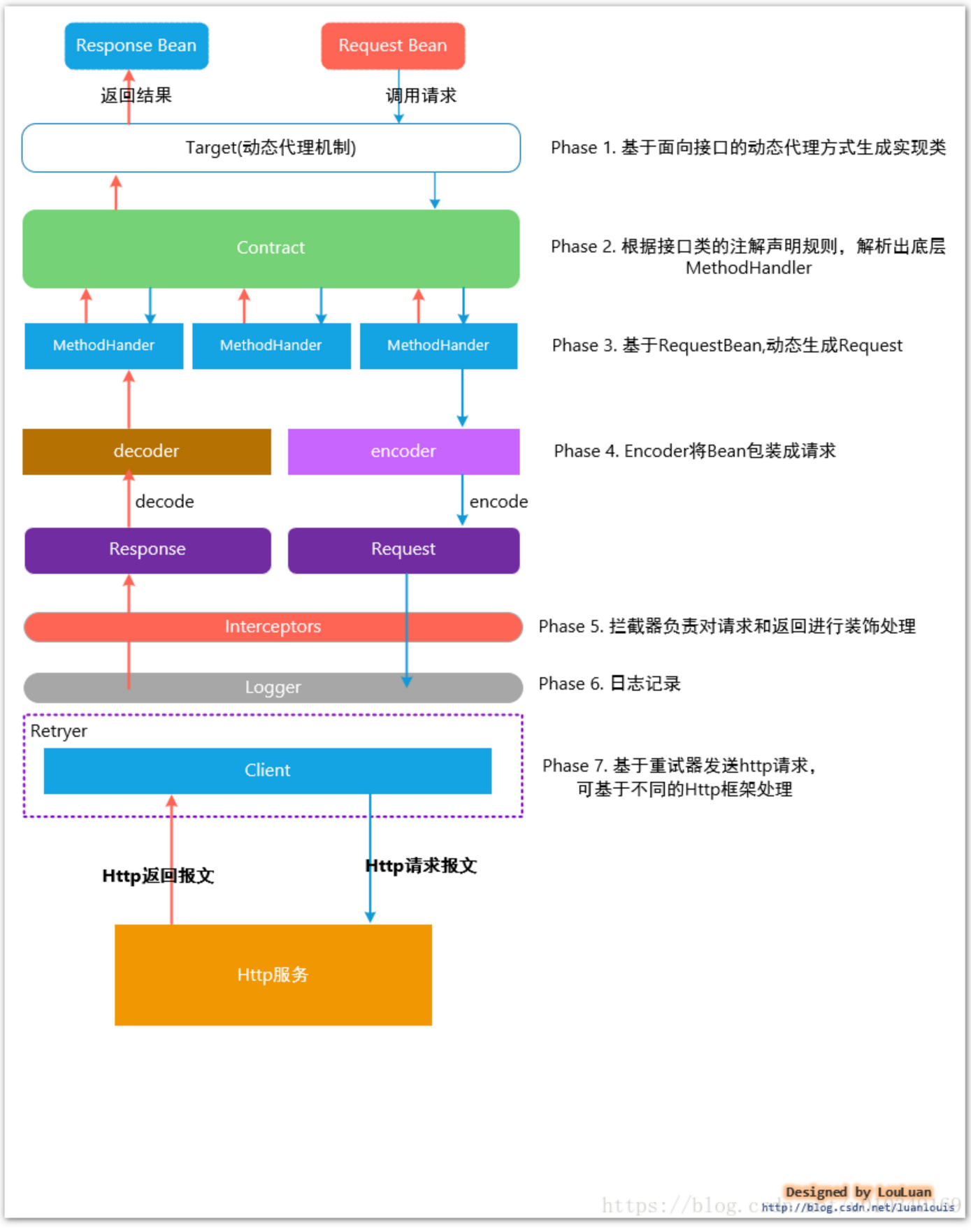


image.png

PHASE 1. 基于面向接口的动态代理方式生成实现类

在使用feign 时，会定义对应的接口类，在接口类上使用Http相关的注解，标识HTTP请求参数信息,如下所示：

```
interface GitHub {
  @RequestLine("GET /repos/{owner}/{repo}/contributors")
  List<Contributor> contributors(@Param("owner") String owner, @Param("repo") String
}

public static class Contributor {
  String login;
  int contributions;
}

public class MyApp {
  public static void main(String... args) {
    GitHub github = Feign.builder()
                         .decoder(new GsonDecoder())
                         .target(GitHub.class, "https://api.github.com");

    // Fetch and print a list of the contributors to this library.
    List<Contributor> contributors = github.contributors("OpenFeign", "feign");
    for (Contributor contributor : contributors) {
      System.out.println(contributor.login + " (" + contributor.contributions + ")");
    }
  }
}
```

(/apps
/redirect?utm_s
banner-click)

在Feign 底层，通过基于面向接口的动态代理方式生成实现类，将请求调用委托到动态代理实现类，基本原理如下所示：

(https://dsp-
click.youdao.co
/clk/request.s?s
k=7uuFgZizt9yH
%3D%3D&
youdao_bid=88
6fe5-49e1-b91a
eb59acd6c87c8
iid=%7B
%22-22799682
%22%3A4%7D
sid=17024)

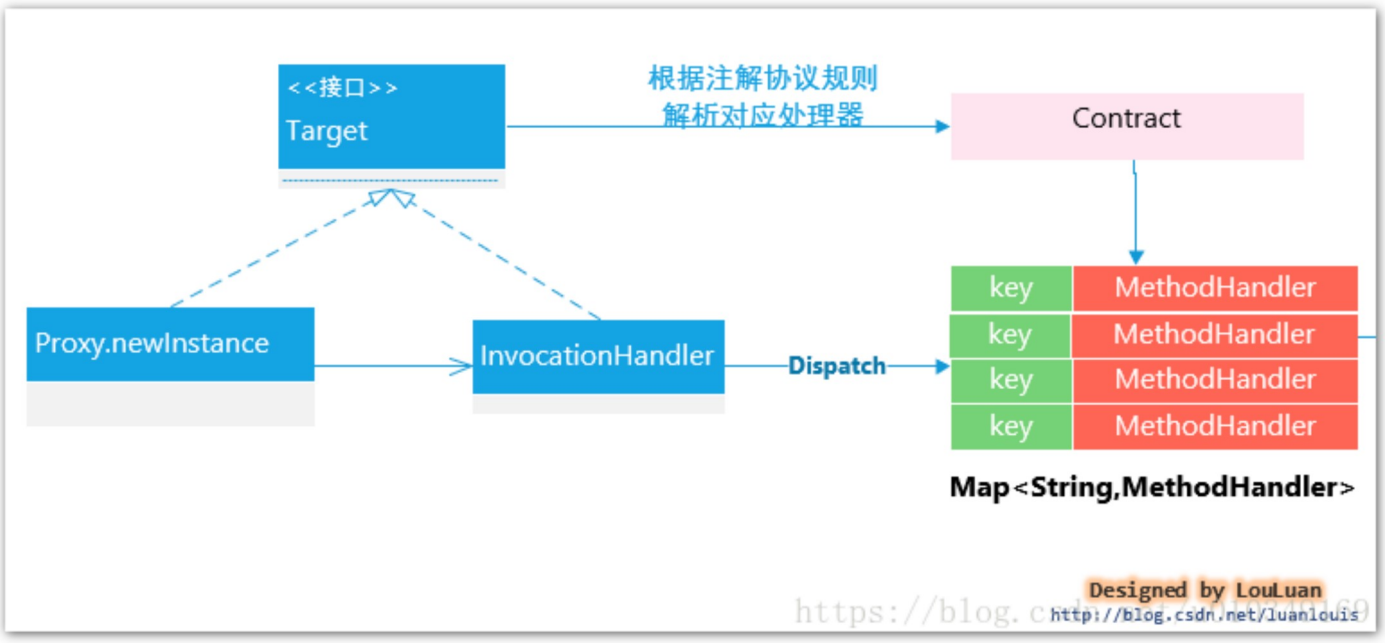


image.png

```
public class ReflectiveFeign extends Feign{
 ///省略部分代码
 @Override
 public <T> T newInstance(Target<T> target) {
   //根据接口类和Contract协议解析方式，解析接口类上的方法和注解，转换成内部的MethodHandle
   Map<String, MethodHandler> nameToHandler = targetToHandlersByName.apply(target);
   Map<Method, MethodHandler> methodToHandler = new LinkedHashMap<Method, MethodHand
   List<DefaultMethodHandler> defaultMethodHandlers = new LinkedList<DefaultMethodHa

   for (Method method : target.type().getMethods()) {
     if (method.getDeclaringClass() == Object.class) {
       continue;
     } else if(Util.isDefault(method)) {
       DefaultMethodHandler handler = new DefaultMethodHandler(method);
       defaultMethodHandlers.add(handler);
       methodToHandler.put(method, handler);
     } else {
       methodToHandler.put(method, nameToHandler.get(Feign.configKey(target.type(),
     }
   }
   InvocationHandler handler = factory.create(target, methodToHandler);
   // 基于Proxy.newProxyInstance 为接口类创建动态实现，将所有的请求转换给InvocationHandl
   T proxy = (T) Proxy.newProxyInstance(target.type().getClassLoader(), new Class<?>
   for(DefaultMethodHandler defaultMethodHandler : defaultMethodHandlers) {
     defaultMethodHandler.bindTo(proxy);
   }
   return proxy;
 }
 //省略部分代码
```

PHASE 2. 根据Contract协议规则，解析接口类的注解信息，解析成内部表现：

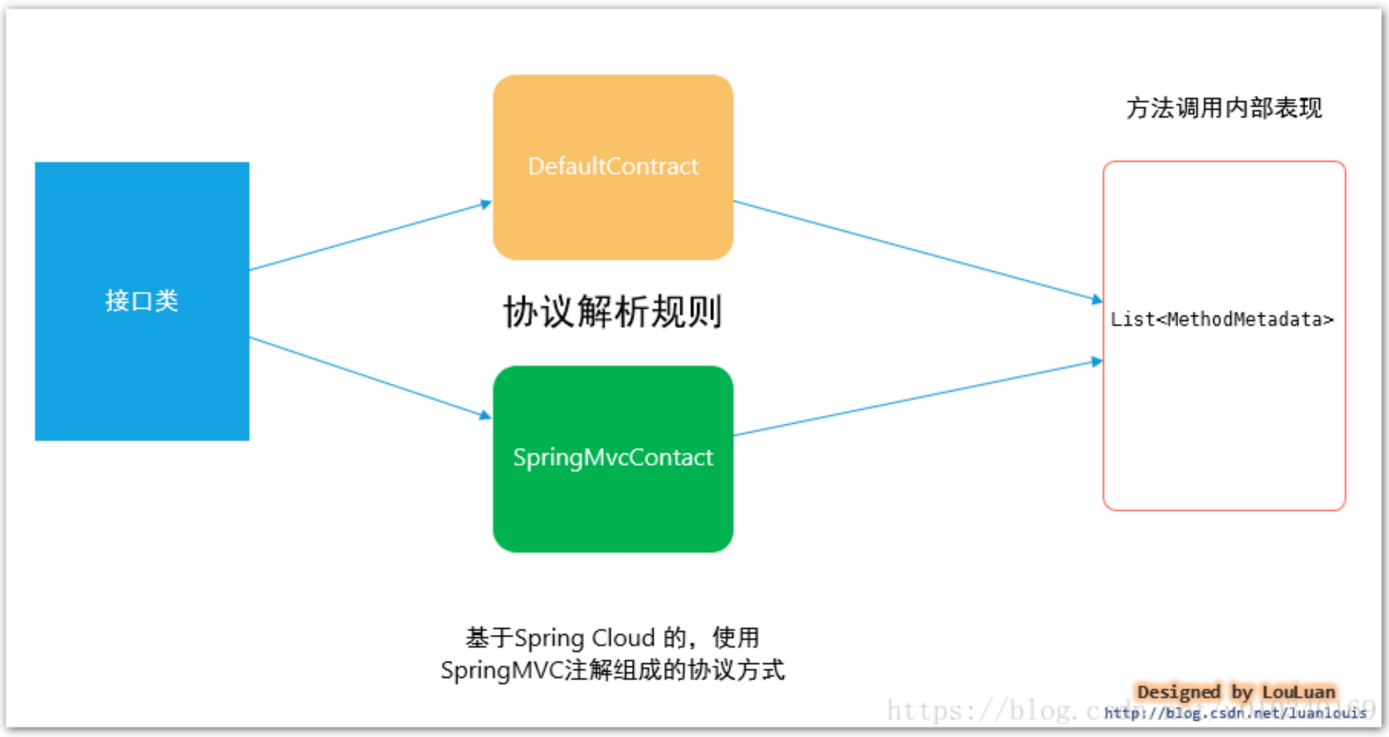


image.png

Feign 定义了转换协议，定义如下：

```
/**
 * Defines what annotations and values are valid on interfaces.
 */
public interface Contract {

  /**
   * Called to parse the methods in the class that are linked to HTTP requests.
   * 传入接口定义，解析成相应的方法内部元数据表示
   * @param targetType {@link feign.Target#type() type} of the Feign interface.
   */
  // TODO: break this and correct spelling at some point
  List<MethodMetadata> parseAndValidatateMetadata(Class<?> targetType);
}
```


默认Contract 实现

Feign 默认有一套自己的协议规范，规定了一些注解，可以映射成对应的Http请求，如官方的一个例子：

```
public interface GitHub {

  @RequestLine("GET /repos/{owner}/{repo}/contributors")
  List<Contributor> getContributors(@Param("owner") String owner, @Param("repo") Stri

  class Contributor {
    String login;
    int contributions;
  }
}
```


上述的例子中，尝试调用GitHub.getContributors("foo","myrepo")的的时候，会转换成如下的HTTP请求：

```
GET /repos/foo/myrepo/contributors
HOST XXXX.XXX.XXX
```

Feign 默认的协议规范

注解	接口 Target	使用说明
`@RequestLine`	方法上	定义HttpMethod 和 UriTemplate. UriTemplate 中使用 `{}` 包裹的表达式，可以通过在方法参数上使用@Param 自动注入
`@Param`	方法参数	定义模板变量，模板变量的值可以使用名称的方式使用模板注入解析
`@Headers`	类上或者方法上	定义头部模板变量，使用@Param 注解提供参数值的注入。如果该注解添加在接口类上，则所有的请求都会携带对应的Header信息；如果在方法上，则只会添加到对应的方法请求上
`@QueryMap`	方法上	定义一个键值对或者 pojo，参数值将会被转换成URL上的 query 字符串上
`@HeaderMap`	方法上	定义一个HeaderMap, 与 UrlTemplate 和HeaderTemplate 类型，可以使用@Param 注解提供参数值

具体FeignContract 是如何解析的，不在本文的介绍范围内，详情请参考代码：
https://github.com/OpenFeign/feign/blob/master/core/src/main/java/feign/Contract.java (https://github.com/OpenFeign/feign/blob/master/core/src/main/java/feign/Contract.java)

基于Spring MVC的协议规范SpringMvcContract:

当前Spring Cloud 微服务解决方案中，为了降低学习成本，采用了Spring MVC的部分注解来完成 请求协议解析，也就是说，写客户端请求接口和像写服务端代码一样：客户端和服务端可以通过SDK的方式进行约定，客户端只需要引入服务端发布的SDK API，就可以使用面向接口的编码方式对接服务：

(/apps/redirect?utm_s banner-click)

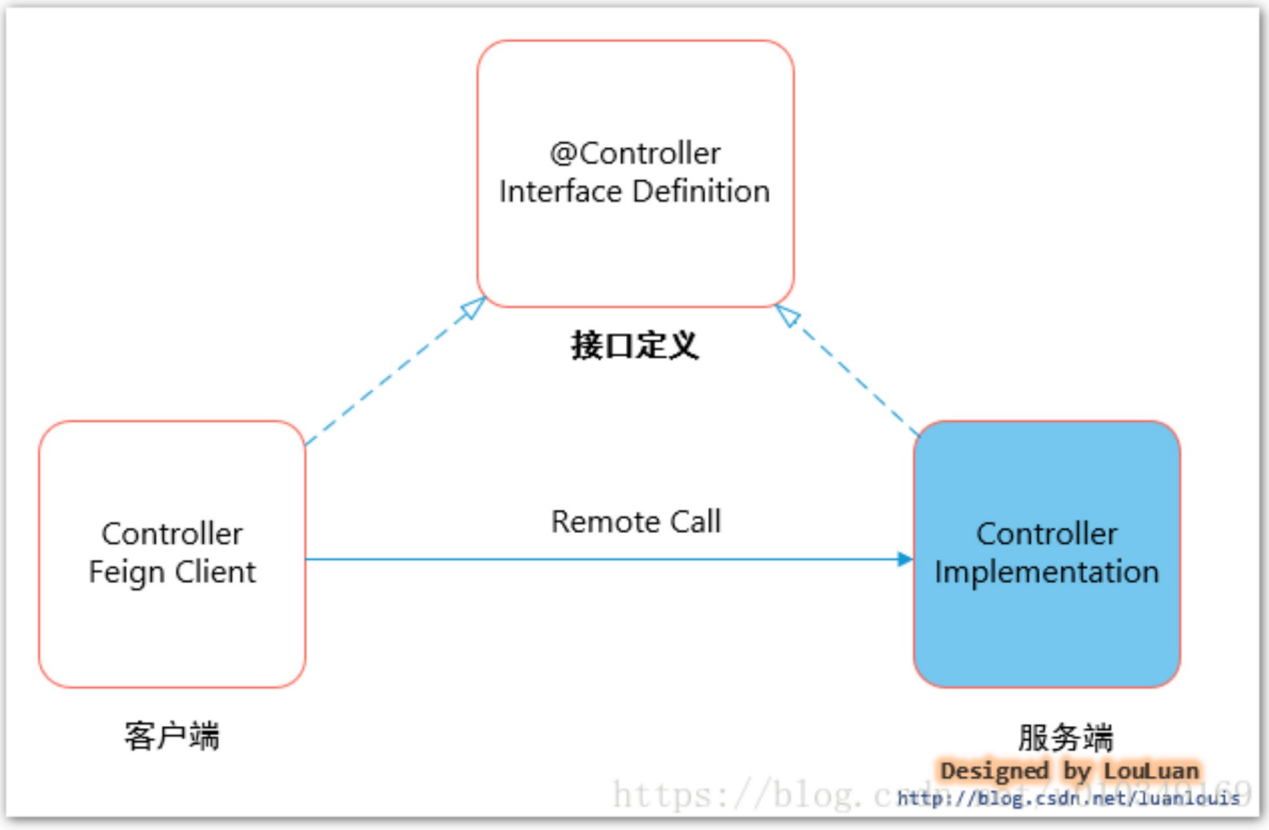


image.png

(https://dsp-click.youdao.co /clk/request.s?s k=7uuFgZizt9yH %3D%3D& youdao_bid=88 6fe5-49e1-b91a eb59acd6c87c8 iid=%7B %22-22799682 %22%3A4%7D sid=17024)

> 我们团队内部就是按照这种思路，结合Spring Boot Starter 的特性，定义了服务端starter,
> 服务消费者在使用的时候，只需要引入Starter，就可以调用服务。这个比较适合平台无关性，接口抽象出来的好处就是可以根据服务调用实现方式自有切换：
>
> 1. 可以基于简单的Http服务调用；
> 2. 可以基于Spring Cloud 微服务架构调用；
> 3. 可以基于Dubbo SOA服务治理
>
> 这种模式比较适合在SaSS混合软件服务的模式下自有切换，根据客户的硬件能力选择合适的方式部署，也可以基于自身的服务集群部署微服务

至于Spring Cloud 是如何实现 协议解析的，可参考代码：https://github.com/spring-cloud/spring-cloud-openfeign/blob/master/spring-cloud-openfeign-core/src/main/java/org/springframework/cloud/openfeign/support/SpringMvcContract.java (https://github.com/spring-cloud/spring-cloud-openfeign/blob/master/spring-cloud-openfeign-core/src/main/java/org/springframework/cloud/openfeign/support/SpringMvcContract.java)

当然，目前的Spring MVC的注解并不是可以完全使用的，有一些注解并不支持,如 `@GetMapping` , `@PutMapping` 等，仅支持使用 `@RequestMapping` 等，另外注解继承性方面也有些问题；具体限制细节，每个版本能会有些出入，可以参考上述的代码实现，比较简单。

> Spring Cloud 没有基于Spring MVC 全部注解来做Feign 客户端注解协议解析，个人认为这个是一个不小的坑。在刚入手Spring Cloud 的时候，就碰到这个问题。后来是深入代码才解决的.... 这个应该有人写了增强类来处理，暂且不表，先MARK一下，是一个开源代码练手的好机会。

PHASE 3. 基于 RequestBean，动态生成Request

根据传入的Bean对象和注解信息，从中提取出相应的值，来构造Http Request 对象：

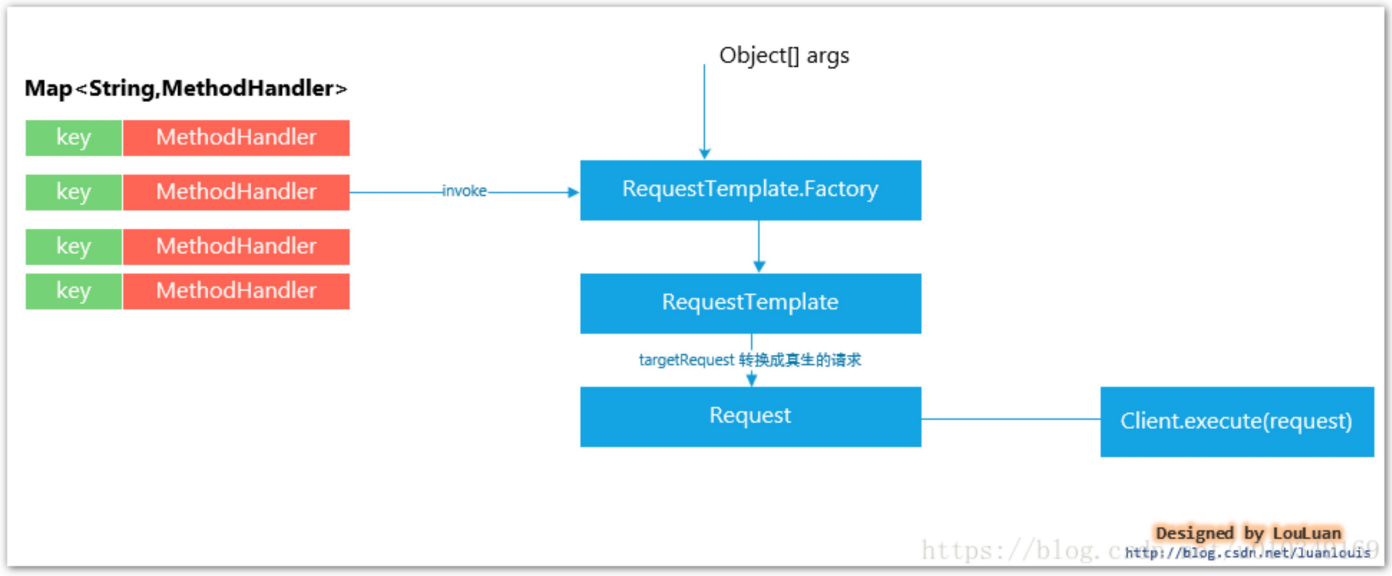


image.png

PHASE 4. 使用Encoder 将Bean转换成 Http报文正文（消息解析和转码逻辑）

Feign 最终会将请求转换成Http 消息发送出去，传入的请求对象最终会解析成消息体，如下所示：

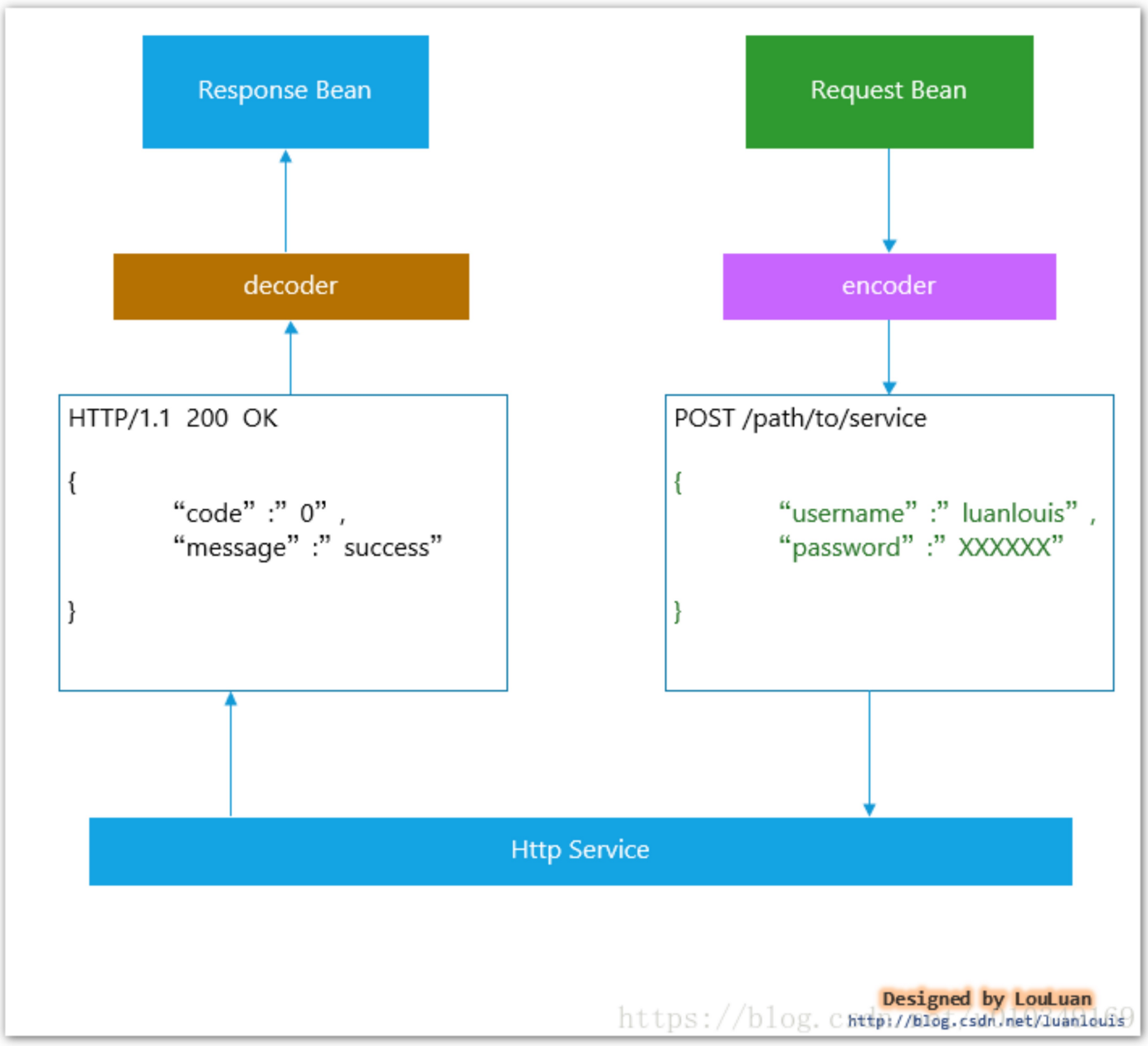


image.png

在接口定义上Feign做的比较简单，抽象出了Encoder 和decoder 接口：

```
public interface Encoder {
  /** Type literal for {@code Map<String, ?>}, indicating the object to encode is a f
  Type MAP_STRING_WILDCARD = Util.MAP_STRING_WILDCARD;

  /**
   * Converts objects to an appropriate representation in the template.
   *  将实体对象转换成Http请求的消息正文中
   * @param object   what to encode as the request body.
   * @param bodyType the type the object should be encoded as. {@link #MAP_STRING_WIL
   *                 indicates form encoding.
   * @param template the request template to populate.
   * @throws EncodeException when encoding failed due to a checked exception.
   */
  void encode(Object object, Type bodyType, RequestTemplate template) throws EncodeEx

  /**
   * Default implementation of {@code Encoder}.
   */
  class Default implements Encoder {

    @Override
    public void encode(Object object, Type bodyType, RequestTemplate template) {
      if (bodyType == String.class) {
        template.body(object.toString());
      } else if (bodyType == byte[].class) {
        template.body((byte[]) object, null);
      } else if (object != null) {
        throw new EncodeException(
            format("%s is not a type supported by this encoder.", object.getClass()))
      }
    }
  }
}
```

```
public interface Decoder {

  /**
   * Decodes an http response into an object corresponding to its {@link
   * java.lang.reflect.Method#getGenericReturnType() generic return type}. If you nee
   * exceptions, please do so via {@link DecodeException}.
   *  从Response 中提取Http消息正文，通过接口类声明的返回类型，消息自动装配
   * @param response the response to decode
   * @param type     {@link java.lang.reflect.Method#getGenericReturnType() generic r
   *                 the method corresponding to this {@code response}.
   * @return instance of {@code type}
   * @throws IOException     will be propagated safely to the caller.
   * @throws DecodeException when decoding failed due to a checked exception besides
   * @throws FeignException  when decoding succeeds, but conveys the operation failed
   */
  Object decode(Response response, Type type) throws IOException, DecodeException, Fe

  /** Default implementation of {@code Decoder}. */
  public class Default extends StringDecoder {

    @Override
    public Object decode(Response response, Type type) throws IOException {
      if (response.status() == 404) return Util.emptyValueOf(type);
      if (response.body() == null) return null;
      if (byte[].class.equals(type)) {
        return Util.toByteArray(response.body().asInputStream());
      }
      return super.decode(response, type);
    }
  }
}
```

目前Feign 有以下实现：

Encoder/ Decoder 实现	说明

Encoder/ Decoder 实现	说明
JacksonEncoder，JacksonDecoder	基于 Jackson 格式的持久化转换协议
GsonEncoder，GsonDecoder	基于Google GSON 格式的持久化转换协议
SaxEncoder，SaxDecoder	基于XML 格式的Sax 库持久化转换协议
JAXBEncoder，JAXBDecoder	基于XML 格式的JAXB 库持久化转换协议
ResponseEntityEncoder，ResponseEntityDecoder	Spring MVC 基于ResponseEntity< T > 返回格式的转换协议
SpringEncoder，SpringDecoder	基于Spring MVC HttpMessageConverters 一套机制实现的转换协议 ,应用于Spring Cloud 体系中

PHASE 5. 拦截器负责对请求和返回进行装饰处理

在请求转换的过程中，Feign 抽象出来了拦截器接口，用于用户自定义对请求的操作：

```
public interface RequestInterceptor {

  /**
   * 可以在构造RequestTemplate 请求时，增加或者修改Header, Method, Body 等信息
   * Called for every request. Add data using methods on the supplied {@link RequestT
   */
  void apply(RequestTemplate template);
}
```


比如，如果希望Http消息传递过程中被压缩，可以定义一个请求拦截器：

```
public class FeignAcceptGzipEncodingInterceptor extends BaseRequestInterceptor {

    /**
     * Creates new instance of {@link FeignAcceptGzipEncodingInterceptor}.
     *
     * @param properties the encoding properties
     */
    protected FeignAcceptGzipEncodingInterceptor(FeignClientEncodingProperties proper
        super(properties);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void apply(RequestTemplate template) {
        //  在Header 头部添加相应的数据信息
        addHeader(template, HttpEncoding.ACCEPT_ENCODING_HEADER, HttpEncoding.GZIP_EN
                HttpEncoding.DEFLATE_ENCODING);
    }
}
```

PHASE 6. 日志记录

在发送和接收请求的时候，Feign定义了统一的日志门面来输出日志信息，并且将日志的输出定义了四个等级：

级别	说明
NONE	不做任何记录

级别	说明
BASIC	只记录输出Http 方法名称、请求URL、返回状态码和执行时间
HEADERS	记录输出Http 方法名称、请求URL、返回状态码和执行时间 和 Header信息
FULL	记录Request 和Response的Header，Body和一些请求元数据

```
public abstract class Logger {

  protected static String methodTag(String configKey) {
    return new StringBuilder().append('[').append(configKey.substring(0, configKey.ir
        .append("] ").toString();
  }

  /**
   * Override to log requests and responses using your own implementation. Messages w
   * request and response text.
   *
   * @param configKey value of {@link Feign#configKey(Class, java.lang.reflect.Method
   * @param format    {@link java.util.Formatter format string}
   * @param args      arguments applied to {@code format}
   */
  protected abstract void log(String configKey, String format, Object... args);

  protected void logRequest(String configKey, Level logLevel, Request request) {
    log(configKey, "---> %s %s HTTP/1.1", request.method(), request.url());
    if (logLevel.ordinal() >= Level.HEADERS.ordinal()) {

      for (String field : request.headers().keySet()) {
        for (String value : valuesOrEmpty(request.headers(), field)) {
          log(configKey, "%s: %s", field, value);
        }
      }

      int bodyLength = 0;
      if (request.body() != null) {
        bodyLength = request.body().length;
        if (logLevel.ordinal() >= Level.FULL.ordinal()) {
          String
              bodyText =
              request.charset() != null ? new String(request.body(), request.charset(
          log(configKey, ""); // CRLF
          log(configKey, "%s", bodyText != null ? bodyText : "Binary data");
        }
      }
      log(configKey, "---> END HTTP (%s-byte body)", bodyLength);
    }
  }

  protected void logRetry(String configKey, Level logLevel) {
    log(configKey, "---> RETRYING");
  }

  protected Response logAndRebufferResponse(String configKey, Level logLevel, Respons
                                            long elapsedTime) throws IOException {
    String reason = response.reason() != null && logLevel.compareTo(Level.NONE) > 0 ?
        " " + response.reason() : "";
    int status = response.status();
    log(configKey, "<--- HTTP/1.1 %s%s (%sms)", status, reason, elapsedTime);
    if (logLevel.ordinal() >= Level.HEADERS.ordinal()) {

      for (String field : response.headers().keySet()) {
        for (String value : valuesOrEmpty(response.headers(), field)) {
          log(configKey, "%s: %s", field, value);
        }
      }

      int bodyLength = 0;
      if (response.body() != null && !(status == 204 || status == 205)) {
        // HTTP 204 No Content "...response MUST NOT include a message-body"
        // HTTP 205 Reset Content "...response MUST NOT include an entity"
        if (logLevel.ordinal() >= Level.FULL.ordinal()) {
          log(configKey, ""); // CRLF
        }
        byte[] bodyData = Util.toByteArray(response.body().asInputStream());
        bodyLength = bodyData.length;
        if (logLevel.ordinal() >= Level.FULL.ordinal() && bodyLength > 0) {
          log(configKey, "%s", decodeOrDefault(bodyData, UTF_8, "Binary data"));
        }
        log(configKey, "<--- END HTTP (%s-byte body)", bodyLength);
        return response.toBuilder().body(bodyData).build();
```

```
    } else {
      log(configKey, "<--- END HTTP (%s-byte body)", bodyLength);
    }
  }
  return response;
}

protected IOException logIOException(String configKey, Level logLevel, IOException
  log(configKey, "<--- ERROR %s: %s (%sms)", ioe.getClass().getSimpleName(), ioe.ge
      elapsedTime);
  if (logLevel.ordinal() >= Level.FULL.ordinal()) {
    StringWriter sw = new StringWriter();
    ioe.printStackTrace(new PrintWriter(sw));
    log(configKey, sw.toString());
    log(configKey, "<--- END ERROR");
  }
  return ioe;
}
```


PHASE 7 . 基于重试器发送HTTP请求

Feign 内置了一个重试器，当HTTP请求出现IO异常时，Feign会有一个最大尝试次数发送请求，以下是Feign核心

代码逻辑：

```
final class SynchronousMethodHandler implements MethodHandler {

  // 省略部分代码

  @Override
  public Object invoke(Object[] argv) throws Throwable {
   //根据输入参数，构造Http 请求。
    RequestTemplate template = buildTemplateFromArgs.create(argv);
    // 克隆出一份重试器
    Retryer retryer = this.retryer.clone();
    // 尝试最大次数，如果中间有结果，直接返回
    while (true) {
      try {
        return executeAndDecode(template);
      } catch (RetryableException e) {
        retryer.continueOrPropagate(e);
        if (logLevel != Logger.Level.NONE) {
          logger.logRetry(metadata.configKey(), logLevel);
        }
        continue;
      }
    }
  }
```


重试器有如下几个控制参数：

重试参数	说明	默认值
period	初始重试时间间隔，当请求失败后，重试器将会暂停 初始时间间隔(线程 sleep 的方式)后再开始，避免强刷请求，浪费性能	100ms
maxPeriod	当请求连续失败时，重试的时间间隔将按照： `long interval = (long) (period * Math.pow(1.5, attempt - 1));` 计算，按照等比例方式延长，但是最大间隔时间为 maxPeriod, 设置此值能够避免 重试次数过多的情况下执行周期太长	1000ms
maxAttempts	最大重试次数	5

具体的代码实现可参考：

https://github.com/OpenFeign/feign/blob/master/core/src/main/java/feign/Retryer.java (https://github.com/OpenFeign/feign/blob/master/core/src/main/java/feign/Retryer.java)

PHASE 8. 发送Http请求

Feign 真正发送HTTP请求是委托给 `feign.Client` 来做的：

```
public interface Client {

  /**
   * Executes a request against its {@link Request#url() url} and returns a response.
   *  执行Http请求，并返回Response
   * @param request safe to replay.
   * @param options options to apply to this request.
   * @return connected response, {@link Response.Body} is absent or unread.
   * @throws IOException on a network error connecting to {@link Request#url()}.
   */
  Response execute(Request request, Options options) throws IOException;
  }
```

Feign 默认底层通过JDK 的 `java.net.HttpURLConnection` 实现了 `feign.Client` 接口类,在每次发送请求的时候，都会创建新的HttpURLConnection 链接，这也就是为什么默认情况下Feign的性能很差的原因。可以通过拓展该接口，使用Apache HttpClient 或者OkHttp3等基于连接池的高性能Http客户端，我们项目内部使用的就是OkHttp3作为Http 客户端。

如下是Feign 的默认实现，供参考：

```
public static class Default implements Client {

    private final SSLSocketFactory sslContextFactory;
    private final HostnameVerifier hostnameVerifier;

    /**
     * Null parameters imply platform defaults.
     */
    public Default(SSLSocketFactory sslContextFactory, HostnameVerifier hostnameVerif
      this.sslContextFactory = sslContextFactory;
      this.hostnameVerifier = hostnameVerifier;
    }

    @Override
    public Response execute(Request request, Options options) throws IOException {
      HttpURLConnection connection = convertAndSend(request, options);
      return convertResponse(connection).toBuilder().request(request).build();
    }

    HttpURLConnection convertAndSend(Request request, Options options) throws IOExcep
      final HttpURLConnection
          connection =
          (HttpURLConnection) new URL(request.url()).openConnection();
      if (connection instanceof HttpsURLConnection) {
        HttpsURLConnection sslCon = (HttpsURLConnection) connection;
        if (sslContextFactory != null) {
          sslCon.setSSLSocketFactory(sslContextFactory);
        }
        if (hostnameVerifier != null) {
          sslCon.setHostnameVerifier(hostnameVerifier);
        }
      }
      connection.setConnectTimeout(options.connectTimeoutMillis());
      connection.setReadTimeout(options.readTimeoutMillis());
      connection.setAllowUserInteraction(false);
      connection.setInstanceFollowRedirects(true);
      connection.setRequestMethod(request.method());

      Collection<String> contentEncodingValues = request.headers().get(CONTENT_ENCOD
      boolean
          gzipEncodedRequest =
          contentEncodingValues != null && contentEncodingValues.contains(ENCODING_GZ
      boolean
          deflateEncodedRequest =
          contentEncodingValues != null && contentEncodingValues.contains(ENCODING_DE

      boolean hasAcceptHeader = false;
      Integer contentLength = null;
      for (String field : request.headers().keySet()) {
        if (field.equalsIgnoreCase("Accept")) {
          hasAcceptHeader = true;
        }
        for (String value : request.headers().get(field)) {
          if (field.equals(CONTENT_LENGTH)) {
            if (!gzipEncodedRequest && !deflateEncodedRequest) {
              contentLength = Integer.valueOf(value);
              connection.addRequestProperty(field, value);
            }
          } else {
            connection.addRequestProperty(field, value);
          }
        }
      }
      // Some servers choke on the default accept string.
      if (!hasAcceptHeader) {
        connection.addRequestProperty("Accept", "*/*");
      }

      if (request.body() != null) {
        if (contentLength != null) {
          connection.setFixedLengthStreamingMode(contentLength);
        } else {
          connection.setChunkedStreamingMode(8196);
        }
```

```
    connection.setDoOutput(true);
    OutputStream out = connection.getOutputStream();
    if (gzipEncodedRequest) {
      out = new GZIPOutputStream(out);
    } else if (deflateEncodedRequest) {
      out = new DeflaterOutputStream(out);
    }
    try {
      out.write(request.body());
    } finally {
      try {
        out.close();
      } catch (IOException suppressed) { // NOPMD
      }
    }
  }
  return connection;
}

Response convertResponse(HttpURLConnection connection) throws IOException {
  int status = connection.getResponseCode();
  String reason = connection.getResponseMessage();

  if (status < 0) {
    throw new IOException(format("Invalid status(%s) executing %s %s", status,
        connection.getRequestMethod(), connection.getURL()));
  }

  Map<String, Collection<String>> headers = new LinkedHashMap<String, Collection<
  for (Map.Entry<String, List<String>> field : connection.getHeaderFields().entry
    // response message
    if (field.getKey() != null) {
      headers.put(field.getKey(), field.getValue());
    }
  }

  Integer length = connection.getContentLength();
  if (length == -1) {
    length = null;
  }
  InputStream stream;
  if (status >= 400) {
    stream = connection.getErrorStream();
  } else {
    stream = connection.getInputStream();
  }
  return Response.builder()
          .status(status)
          .reason(reason)
          .headers(headers)
          .body(stream, length)
          .build();
    }
  }
```

Feign 的性能怎么样?

Feign 整体框架非常小巧，在处理请求转换和消息解析的过程中，基本上没什么时间消耗。真正影响性能的，是处理Http请求的环节。

如上所述，由于默认情况下，Feign采用的是JDK的 `HttpURLConnection` ,所以整体性能并不高，刚开始接触Spring Cloud 的同学，如果没注意这些细节，可能会对Spring Cloud有很大的偏见。

我们项目内部使用的是OkHttp3 作为连接客户端。

系统的压测方案后续在贴出来,有兴趣的同学可以持续关注~

觉得不错，打赏一下~

赞赏支持

(/u/27402dfef86f)

(/apps
/redirect?utm_s
banner-click)

日记本 (/nb/29623811)

举报文章

亦山札记 (/u/802bbe244ebb) ♂

写了 6688 字，被 25 人关注，获得了 65 个喜欢

(/u/802bbe244ebb)

+关注

喜欢 51

更多分享

下载简书 App ▸

随时随地发现和创作内容

(/apps/redirect?utm_source=note-bottom-click)

(https://dsp-
click.youdao.co
/clk/request.s?s
k=7uuFgZizt9yH
%3D%3D&
youdao_bid=88
6fe5-49e1-b91a
eb59acd6c87c&
iid=%7B
%22-22799682
%22%3A4%7D&
sid=17024)

登录 (/sign后发表评论source=desktop&utm_medium=not-signed-in-comment-

2条评论 只看作者 按时间倒序 按时间正序

juconcurrent (/u/794bce41b31b)

3楼 · 2018.12.07 15:33

(/u/794bce41b31b)

写得很好，已经比较全局地对feign进行了了解，👍

赞 回复

王世玮 (/u/27402dfef86f)

2楼 · 2018.09.26 19:54

(/u/27402dfef86f)

学习了～大佬，可以交个朋友么?

赞 回复

被以下专题收入，发现更多相似内容


Spring ... (/c/9d5648980d5f?utm_source=desktop&utm_medium=notes-included-collection)


Spring ... (/c/04217c089f34?utm_source=desktop&utm_medium=notes-included-collection)


SSM+shiro等 (/c/f4c65dcb6f21?utm_source=desktop&utm_medium=notes-included-collection)


java所有基础知识 (/c/9bd828fa75d5?utm_source=desktop&utm_medium=notes-included-collection)

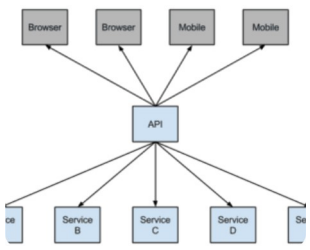

框架原理 (/c/866bb009b23f?utm_source=desktop&utm_medium=notes-included-collection)


开源架构 (/c/3a3952b0ad1e?utm_source=desktop&utm_medium=notes-included-collection)

geneseeq (/c/b3d002510354?utm_source=desktop&utm_medium=notes-included-collection)

展开更多

(/p/46fd0faecac1?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)



Spring Cloud (/p/46fd0faecac1?utm_campaign=ma…

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智能路由，微代理，控制总线）。分布式系统的协调导致了样板模式, 使用Spring Cloud开发人员可以快速地支持实现这些模式的服务和应用程序。他们将在任何分布式...

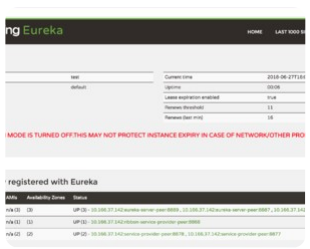

卡卡罗2017 (/u/d90908cb0d85?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

Spring boot参考指南 (/p/67a0e41dfe05?utm_campaign=maleskine&ut…

Spring Boot 参考指南 介绍 转载自:https://www.gitbook.com/book/qbgbook/spring-boot-reference-guide-zh/details带目录浏览地址:http://www.maoyupeng.com/sprin...


毛宇鹏 (/u/d3ea915e1e0f?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/59295c91dde7?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)



Spring Cloud Feign使用详解 (/p/59295c91dde7?utm…

通过前面两章对Spring Cloud Ribbon和Spring Cloud Hystrix的介绍，我们已经掌握了开发微服务应用时，两个重要武器，学会了如何在微服务架构中实现客户端负载均衡的服务调用以及如何通过断路器来保护我们的微服务应用。这两者将被作为基础工具类框架广...


Chandler_珏瑜 (/u/117796446366?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

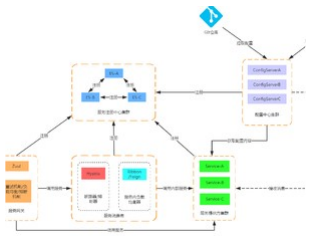
(/p/3899d7f47303?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

微服务架构集大成者—Spring Cloud (/p/3899d7f47303?utm_campaign=…

(/apps/redirect?utm_s
banner-click)

(https://dsp-click.youdao.co
/clk/request.s?s
k=7uuFgZizt9yH
%3D%3D&
youdao_bid=889
6fe5-49e1-b91a
eb59acd6c87c8
iid=%7B
%22-22799682
%22%3A4%7D
sid=17024)

软件是有生命的，你做出来的架构决定了这个软件它这一生是坎坷还是幸福。
本文不是讲解如何使用Spring Cloud的教程，而是探讨Spring Cloud是什么，以及它诞生的背景和意义。 1 背景 2008年以后，国内互联网行业飞速发展，我们对软件系统的需求已经不再是过去”...



Bobby0322 (/u/99ee95856388?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/apps/redirect?utm_s
banner-click)

Spring Cloud Sleuth使用简介 (/p/6d6b52c7624f?utm_campaign=males…

文章主要翻译自Spring Cloud Sleuth官方文档 Spring-Cloud Spring Cloud为开发者提供了在分布式系统(如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性Token、全局锁、决策竞选、分布式会话和集群状态)操作的开发工具。使...

楠倏之语 (/u/e85ebcea4e16?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

随笔 (/p/07481bc36170?utm_campaign=maleskine&utm_content=note…

生活无它，你选择走，我选择留，而已。

6cb508f53dde (/u/6cb508f53dde?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/2d8e25335704?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

坚持，是一种态度 (/p/2d8e25335704?utm_campaign…

2016年7月16日，一个很普通的周六，我开始填之前自己挖的坑 - - 跑步机马拉松全程。为什么说它是坑呢？对于一个从去年四月开始就没再刷过30+，且每月跑量几乎不超百的人来说，这真的就是个坑，而且很大很深。 7月焦阳似火，室内即使开着空调也无济于事。所幸有同行的小伙伴可以聊天，...

Love晒太阳的猫 (/u/1c2735f1cdcb?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(https://dsp-click.youdao.co
/clk/request.s?s
k=7uuFgZizt9yH
%3D%3D&
youdao_bid=88
6fe5-49e1-b91a
eb59acd6c87c8
iid=%7B
%22-22799682
%22%3A4%7D
sid=17024)

2018-08-30 (/p/fbe8ecf905cc?utm_campaign=maleskine&utm_content…

李健说的平静，八年的老手机。 现在人看手机，甚过于恋人。温情和浪漫的俞不可得。手机在杀死我们这代人吗。 3年的是否值得等待，时间会不会摧毁感情。等我3年，了无音信。感情最重要的是平淡的陪伴吗，其他是否真的不那么重要。 感情变淡了，是为什么。那么喜欢的人，现在一句话也不想多说...

别走别跑站着 (/u/e3cf384f18ab?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/483a18920da3?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

7月1日感恩日记 (/p/483a18920da3?utm_campaign=…

感恩今天是7月1日，党的生日。感恩自己生活在这样一个伟大的国度，国泰民安，国富民强，身为中国人，我很骄傲。 感恩亲爱的李姐。今天是小升初的日子，家里忙的不可开交，李姐主动过来帮忙，才得以我们七点就能出门送优优去考试。等到中午一点多我们才赶回来，而李姐早已做好了一桌子好吃的等...

武丹yoyo (/u/bbad4ec4683f?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)