HTTPCORE 项目源码分析报告:

报告人　：方相
学习机构：中国科学院大学（UCAS）
学号　　： 2016K8009906033
邮箱　　：fangxiang16@mails.ucas.ac.cn
分析项目： httpcomponents
主要功能： HTTP 请求处理

一．项目介绍和具体功能

（一）http 项目

该开源项目由 Apache 开源组织在 github 社区提供，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端/服务器编程工具包。

该项目有 4 个子项目：

HttpComponents Core
HttpComponents Client
HttpComponents AsyncClient
Commons HttpClient

在本次实例分析中，对 HttpComponents Core（简称 HttpCore）进行了分析.

（二）功能和目标

1.功能：HttpCore 是对 HTTP 协议的基础封装的一套组件。

（1）可以用它来建立客户端、代理、服务端 Http 服务.
（2）支持同步异步服务。
（3）一系列基于阻塞和非阻塞 IO 模型。

2.目标:

（1）实现最基本的 HTTP 传输
（2）保持良好的性能和清晰度和表现力之间的平衡
（3）尽量小的（预测）内存占用
（4）自我包含的类库（没有超越 JRE 的额外依赖）

3.不能替代 HttpClient，Servlet API。

（三）项目整体模块

1.基础：阻塞 I/O 模型

阻塞 Http 连接，http 异常处理，阻塞 http 协议处理。

2.扩展：NIO 扩展

I/O 反应器，I/O 反应器异常处理，非阻塞 Http 连接，非阻塞 http 协议处理

3.高级主题

自定义 Http 连接

连接类型：阻塞 http 连接，非阻塞 http 连接，自定义 HTTP 连接。

最终选取的分析部分为基础的阻塞 http 连接功能部分。

二．类的设计分析

http 协议所描述的可以用一句话概括:点对点的消息交换(一端向另一端发起请求(request),接收端处理请求并返回消息(response)). 不管是 http 请求还是 http 响应,我们都把它当做 http 消息(message)。

（一）请求消息和响应消息

1.一个 http message 包含许多描述 Message 属性的 header（content length, content type, cookie, user-agent 等 ）和可选的 Body.

其结构大致如下:

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
start-line = Request-Line | Status-Line
```

http message 提供了一系列方法用来取出、添加、移除、枚举这些 header:

例如:

```
Header h1 = httpResponse.getFirstHeader("Set-Cookie");
System.out.println(h1);

Header h2 = httpResponse.getLastHeader("Set-Cookie");
System.out.println(h2);
```

2. HttpRequest 和 HttpResponse 则分别对应请求消息和响应消息,

（1）请求消息包含应用于资源的方法，资源标识符，使用的协议版本等。

Request 结构如下:

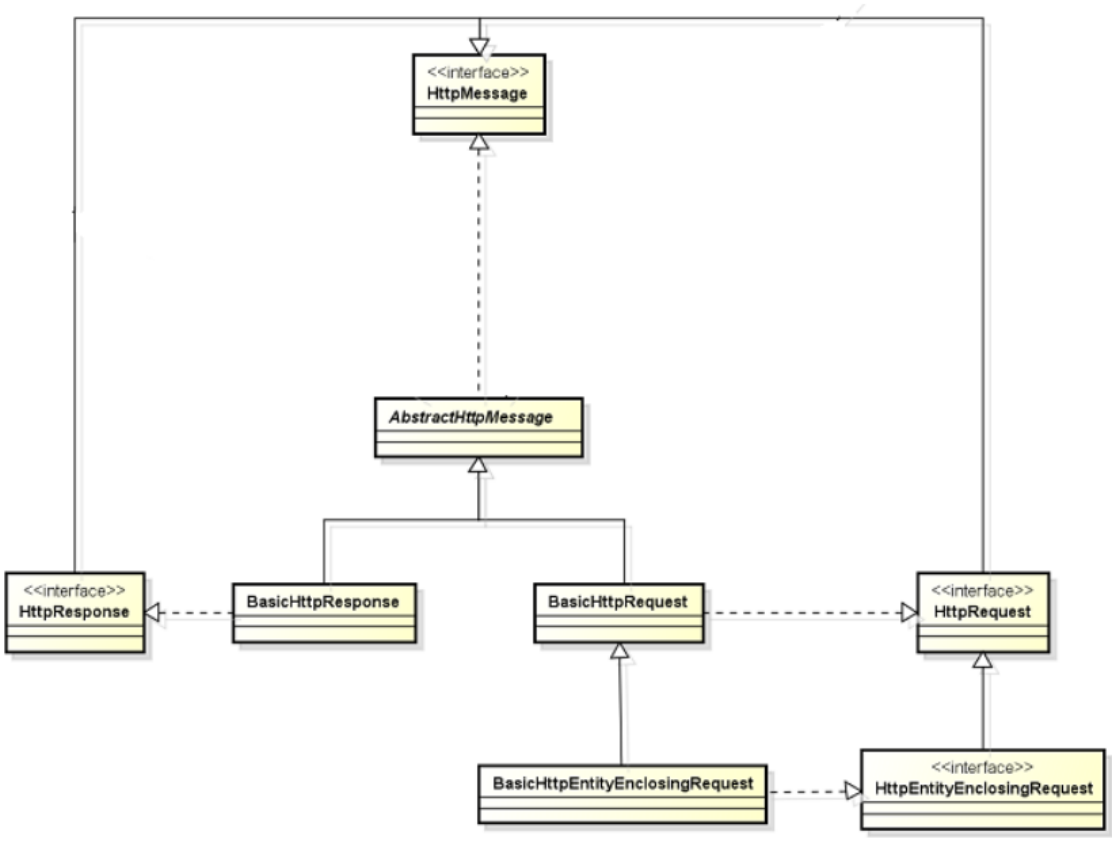
```
Request = Request-Line
    *(( general-header
    | request-header
    | entity-header ) CRLF)
    CRLF
    [ message-body ]
```

（2）响应消息包含协议版本，数字状态码，相关文本段等。

Response 结构如下:

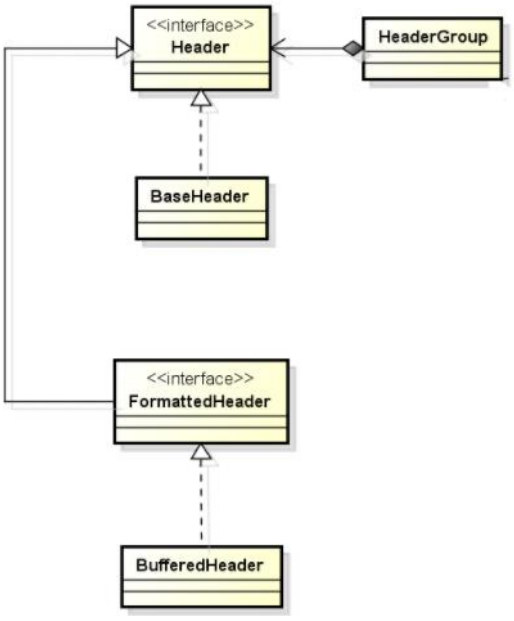
```
Response = Status-Line
    *(( general-header
    | response-header
    | entity-header ) CRLF)
    CRLF
    [ message-body ]
```

不同消息在项目的交互如下图:



3.消息头(Header)

消息头可以看成是消息的元数据描述对象，它是对请求消息(request),响应消息(response),消息实体(entity)的元描述，比如请求消息头 Accept 就是指定请求客户端可以理解哪些媒体类型，Content-Length 头指定 entity 的大小长度。



（二） HttpEntity（Http 实体）

1. Http 消息可以携带，请求和响应相关的内容实体。

（1）HttpCore 根据 entity 的表现存在形式分为以下几种:

ByteArrayEntity : byte 数组实现
FileEntity : 本地文件实现
InputStreamEntity : io 流文件实现
HttpEntityWrapper : 对别的 entity 进行包装，添加新功能，利用 decorator 模式
BufferedHttpEntity : 继承扩展 HttpEntityWrapper，实现可缓冲的 entity

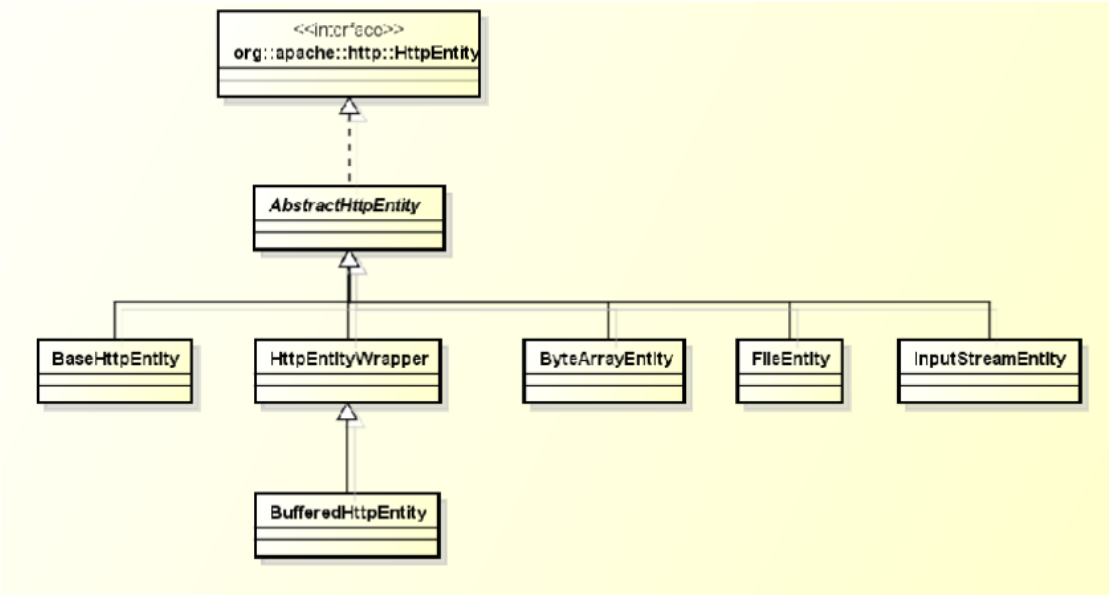
（2）HttpCore 根据内容来源又分成三种:

streamed，从流中接收，或者在运行中产生，不可重复读。
self-contained，存储于内存中或通过独立的连接或其他实体获得，可重复读。
.wrapping，从另外一个实体获得。

2.HttpEntity 接口定义了一系列方法来获取内容实体

InputStream getContent()（以流的方式返回实体内容），
void writeTo(OutputStream outputStream)（将实体内容写到一个输出流中）
Header getContentEncoding()，
long getContentLength()
Header getContentType()，
boolean isChunked()，
boolean isRepeatable()，
boolean isStreaming()

3．根据 entity 的表现存在形式不同 entity 交互如下



3. Request Method(请求方法)

请求方法主要有以下几种:

Options
Get
Head
Post
Put
Delete
Trace
Connect

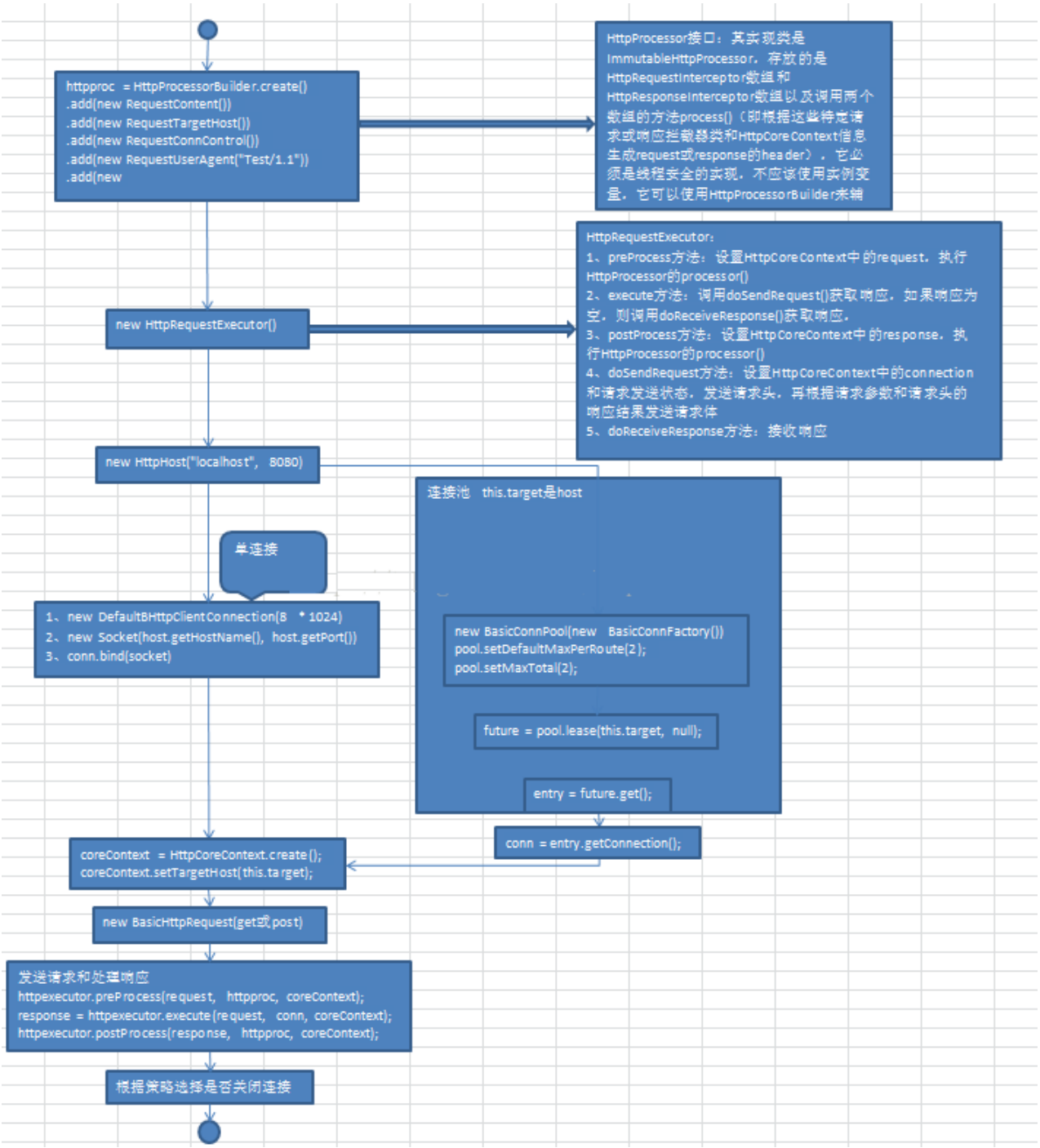
其中 Get、Post 是最常见的获取网页内容的请求形式。

下面是用 Get 请求获取一个 html 网页内容的代码:

```
//(1)创建HttpGet实例..
HttpGet get = new HttpGet("http://www.126.com");..
..
//(2)使用HttpClient发送get请求，获得返回结果HttpRes
HttpClient http = new DefaultHttpClient();..
HttpResponse response = http.execute(get);..
..
//(3)读取返回结果..
if (response.getStatusLine().getStatusCode() == 200)
....HttpEntity entity = response.getEntity();..
..
....InputStream in = entity.getContent();..
....readResponse(in);..
}..
```

三．高级设计分析

（一）一个 reques 在 http 中整个流程如下:



主要步骤解析:

1、阻塞式连接的使用

可以利用 HttpConnection 的对象拿到连接的一些信息（本地及远程主机地址和端口号），同时可以拿到一个封装了连接使用过程中的统计信息的 HttpConnectionMetrics 的对象，从而获取与连接相关的一些统计信息（请求和响应数量，接受和发送的数据字节数等）。

```java
public void httpConnectionBindSocketTest() throws Exception {
        //定义一个Socket
        Socket socket = new Socket("122.0.0.2", 8080);
        //定义一个有固定缓冲区的Http连接对象
        DefaultBHttpClientConnection connection = new DefaultBHttpClientConnection(8*1024);
        //将http连接对象绑定到套接字上,执行此方法后，这个套接字将被连接用来发送和接受数据，
        //连接的状态会变为打开，isOpen()将返回true
        connection.bind(socket);
        //获取连接的一些相关信息，例如：连接是否打开、连接的本地及远程主机地址和端口号等
        System.out.println(connection.isOpen());
        System.out.println(connection.getLocalPort());
        System.out.println(connection.getRemoteAddress());

        //HttpConnectionMetrics封装了使用该连接的过程中的一些统计信息，例如：请求和响应数量，发送和接收到的字节数量等
        HttpConnectionMetrics metrics = connection.getMetrics();

        System.out.println(metrics.getRequestCount());
        System.out.println(metrics.getResponseCount());
        System.out.println(metrics.getReceivedBytesCount());
        System.out.println(metrics.getSentBytesCount());
    }
```

对于 core 服务端，当接受到请求后，处理大致如下：

```java
public void serverConnectionTest() throws Exception {
        Socket socket = new Socket("122.0.0.2", 8081);

        DefaultBHttpServerConnection connection = new DefaultBHttpServerConnection(8*1024);
        //将特定的Socket绑定到服务端的Connection中
        connection.bind(socket);
        //接收客户端请求的请求头，并将其封装到一个请求对象中
        HttpRequest request = connection.receiveRequestHeader();

        if(request instanceof HttpEntityEnclosingRequest) {
            //接收客户端请求的请求体
            connection.receiveRequestEntity((HttpEntityEnclosingRequest) request);
            HttpEntity entity = ((HttpEntityEnclosingRequest) request).getEntity();
            if(entity!=null) {
                //处理请求中的实体，拿到需要的信息
                //关闭底层流
                EntityUtils.consume(entity);
            }
        }

        HttpResponse response = new BasicHttpResponse(HttpVersion.HTTP_1_1, 200, "OK");
        response.setEntity(new StringEntity("Get it"));

        //发送响应头
        connection.sendResponseHeader(response);
        //发送响应体
        connection.sendResponseEntity(response);
    }
```

2.阻塞式 http 协议的处理：

（1）Http Service

HttpService 是一个基于阻塞式 I/O 模型，满足 Http 协议对服务端消息处理的基本要求的服务端协议处理器。

a. http 请求处理器

HttpRequestHandler 接口代表处理一组特定 Http 请求的过程，主要目的是为了响应给定的请求，而生成一个带有内容实体的响应对象，以便于将其发送给客户端。

示例：

```
public void httpServiceTest() {
        HttpProcessor processor = HttpProcessorBuilder.create()
                                        .add(new ResponseDate())
                                    .add(new ResponseServer("MyServer-HTTP/1.1"))
                                .add(new ResponseContent())
                                    .add(new ResponseConnControl())
                                            .build();
        //构造方法中需要传入一个HttpProcessorde对象
        HttpService service = new HttpService(processor, null);
        System.out.println(service);
    }
```

b. 请求处理程序解析器

Http 请求处理器通常由 HttpRequestHandlerResolver 来管理, 它能将一个 Http 请求的 URI 与 HttpRequestHandler 匹配。

HttpCore 包含一个基于简单模式匹配算法的 HttpRequestHandlerResolver 的实现。

示例:

```
public void httpRequestHandlerResolverTest() {
    HttpProcessor processor = <...>

    HttpRequestHandler mHandler1 = <...>
    HttpRequestHandler mHandler2 = <...>
    HttpRequestHandler mHandler3 = <...>

    UriHttpRequestHandlerMapper handlerMapper = new UriHttpRequestHandlerMapper();
    handlerMapper.register("/service/*", mHandler1);
    handlerMapper.register("*.do", mHandler2);
    handlerMapper.register("*", mHandler3);

    HttpService service = new HttpService(processor, handlerMapper);
}
```

(2) Http 请求执行器

HttpRequestExecutor 是一个客户端的协议处理器，它是基于阻塞式 I/O 模型，实现了 Http 协议对客户端消息处理所提出的必要的要求。

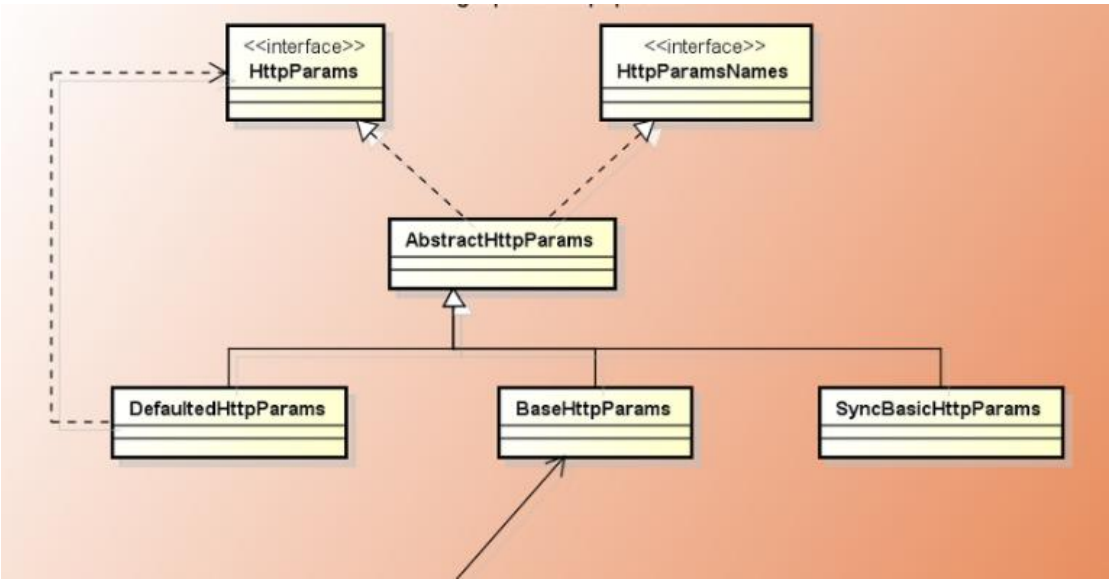
HttpRequestExecutor 依赖 HttpProcessor 对所有传入的消息产生强制性的协议头,对于所有传入和传出的消息应用通用交叉消息转换。

一旦执行请求并收到响应，应用程序就能实现 HttpRequestExecutor 之外的特定处理

(二)高级设计

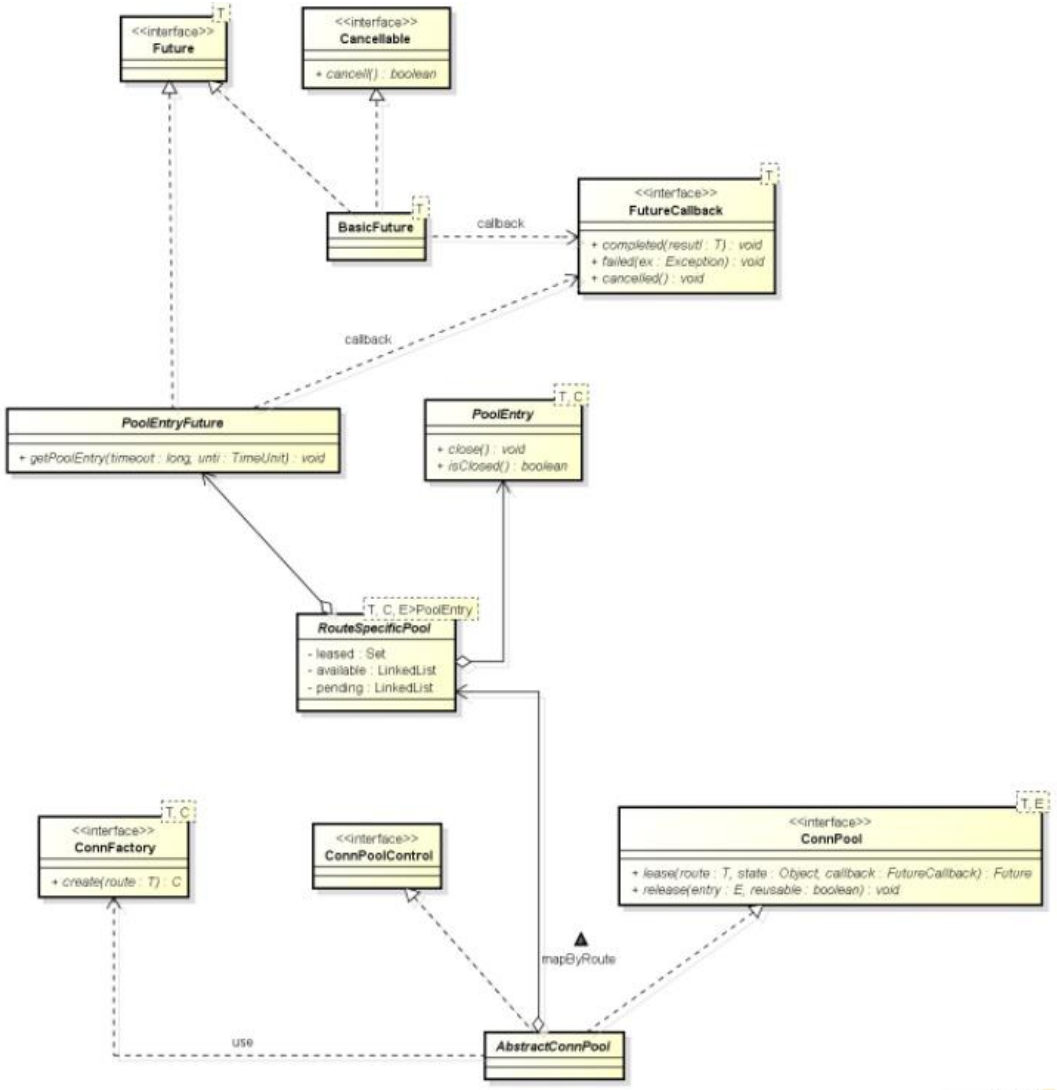
1. HttpParams

在 Http 各组件运行中，都要用到各种运行时的参数

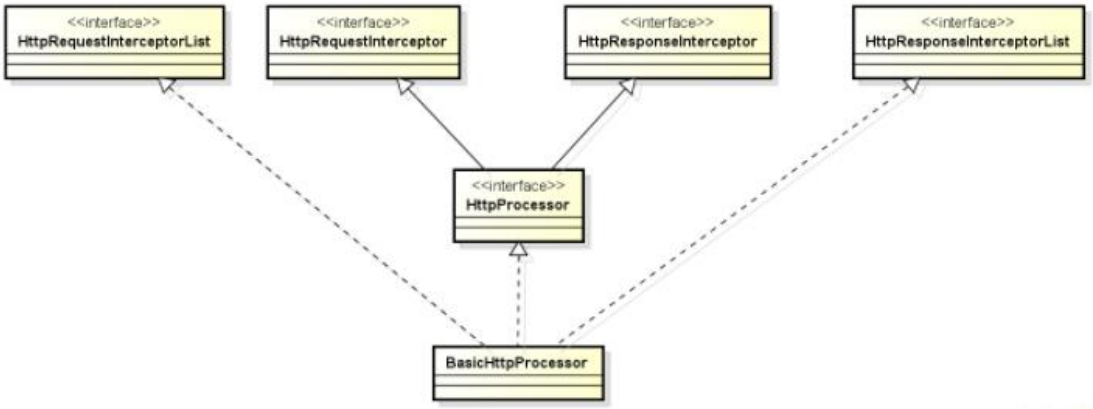


2.连接池

针对 Connection(网络连接资源，如 Socket)进行“池”管理。主要是对 URI 中相同的 Host 建立的 Socket 连接实现 reuseable 化，提高效率和优化性能，满足 Http1.1 中的 Connection 头描述（值为 keep-alive 时不关闭每个当前连接，以供下次请求使用）



3. AOP 架构模式，拦截处理 request 和 response 消息的各种头，内容体处理



四．总结和声明

以上报告仅代表我个人在学习 http core 项目的拙见，如有问题，欢迎通过邮箱等联系方式指正或与本人共同探讨。谢谢!

项目源码 github：https://github.com/apache/httpcomponents-core

五．参考文献

1. HttpComponents 入门解析，https://blog.csdn.net/zmken497300/article/details/52845936
2. httpcore 和 httpclient 的源码一点点，blog.csdn.net/u010681276/article/details/49555961
3. HttpCore 和 HttpClient，blog.csdn.net/lzp158869557/article/details/78204758
4. HttpCore 教程（四），blog.csdn.net/venus14/article/details/79734666
5. Apache HttpCore 官方教程笔记，blog.csdn.net/a5987995329/article/details/76467673
6. Apache HttpCore 学习，www.cnblogs.com/jcli/archive/2012/10/17/2727632.html