HTTPCORE 项目源码分析报告:

报告人：方相
学号 ：2016K8009906033
项目名: httpcomponents
主要功能:HTTP 请求处理

一．项目总体了解

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．HttpCore 项目大致了解

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．HttpCore 源码阅读分析

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。

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

高级主题

自定义 Http 连接

目前主要阅读目录:

C:\Users\FangXiang\Desktop\httpcomponents-core-master\httpcore5\src\main\java\org\apache\hc\core5\http

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

目前大致是分析基础的阻塞 http 连接功能部分。

（一）下面是一些基础内容的了解。

1. 请求报文和响应报文

在文件 Message.java 中定义了类: Message, 在文件夹 Message 则定义了各种 Message 的属性和一般方法。

Message 结构大致如下：

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

一个 http message 包含许多描述 Message 属性的 header（content length, content type, cookie, user-agent 等 ）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 则分别对应请求报文和响应报文，
   请求报文包含应用于资源的方法，资源标识符，使用的协议版本等。
   响应报文包含协议版本，数字状态码，相关文本段等。
   Request 结构如下：

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

   Response 结构如下：

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

3. HttpEntity（Http 实体）(HttpEntity.java 和文件夹 io/Entity)
   Http 报文可以携带，请求和响应相关的内容实体。
   HttpCore 根据内容来源分成三种实体:
   (1).streamed，从流中接收，或者在运行中产生，不可重复读。
   (2).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。
   (3).wrapping，从另外一个实体获得。

   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()

4. 连接池(Connection Pool)
连接池可以用来提高连接的持久化重用效率。

（二）接下来对阻塞 IO 模型的一些简要分析:

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）。

```
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:

```
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。

(1)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);
    }
```

(2)、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