Httpcore主要功能代码阅读和设计意图分析

2016K8009908053

1603班 孙越

1. **HTTP是什么**

根据RFC2616的定义，超文本传输协议（HTTP）是一种为共有的超文本信息传输系统定制的应用层协议。HTTP是一个客户端和服务器端请求和应答的标准（Transmission Control Protocol, TCP）。客户端通过使用Web浏览器、网络爬虫或者其它的工具发起一个到服务器上指定端口的HTTP请求。HTTP协议采用了请求/响应模型。整个基本的工作流程是客户端发送一个HTTP请求，说明客户端想要访问的资源和请求的动作，服务端收到请求之后开始处理，并根据请求做出相应的动作访问服务器资源，并以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。最后通过发送HTTP响应把结果返回给客户端。

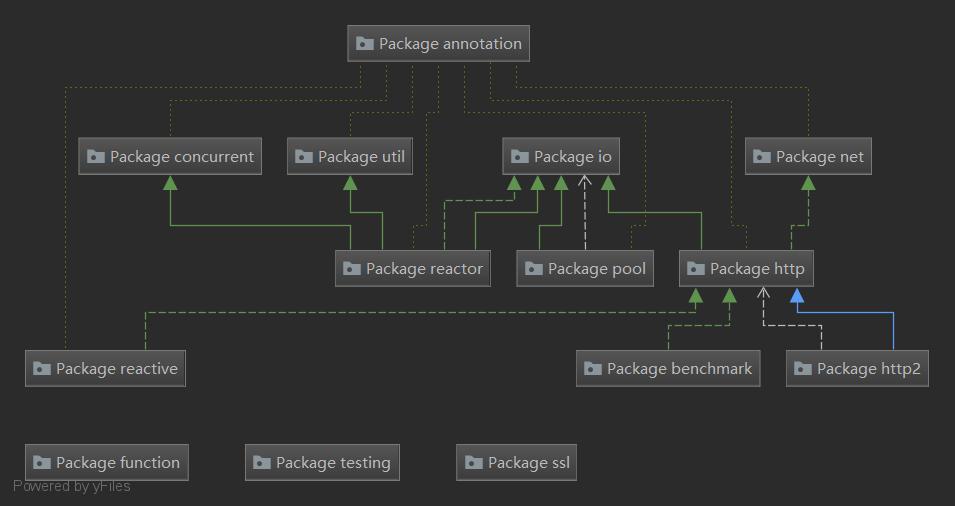
HTTP经历了多个版本，目前最常用的版本为HTTP1.1。版本号在每次HTTP请求的消息中是被要求的，一般处于消息头中。

1. **HttpCore的内容和主要功能**

HttpCore是基于超文本传输协议的一种实现方式，是一套实现了HTTP协议最基础功能的组件。HttpCore可以理解为一种人为编写的应用接口，这种接口实现了用户（应用端）与服务供应商（服务器端）还有中间的代理端之间信息交互的一致性，有效确保了信息传递的效率和安全性。

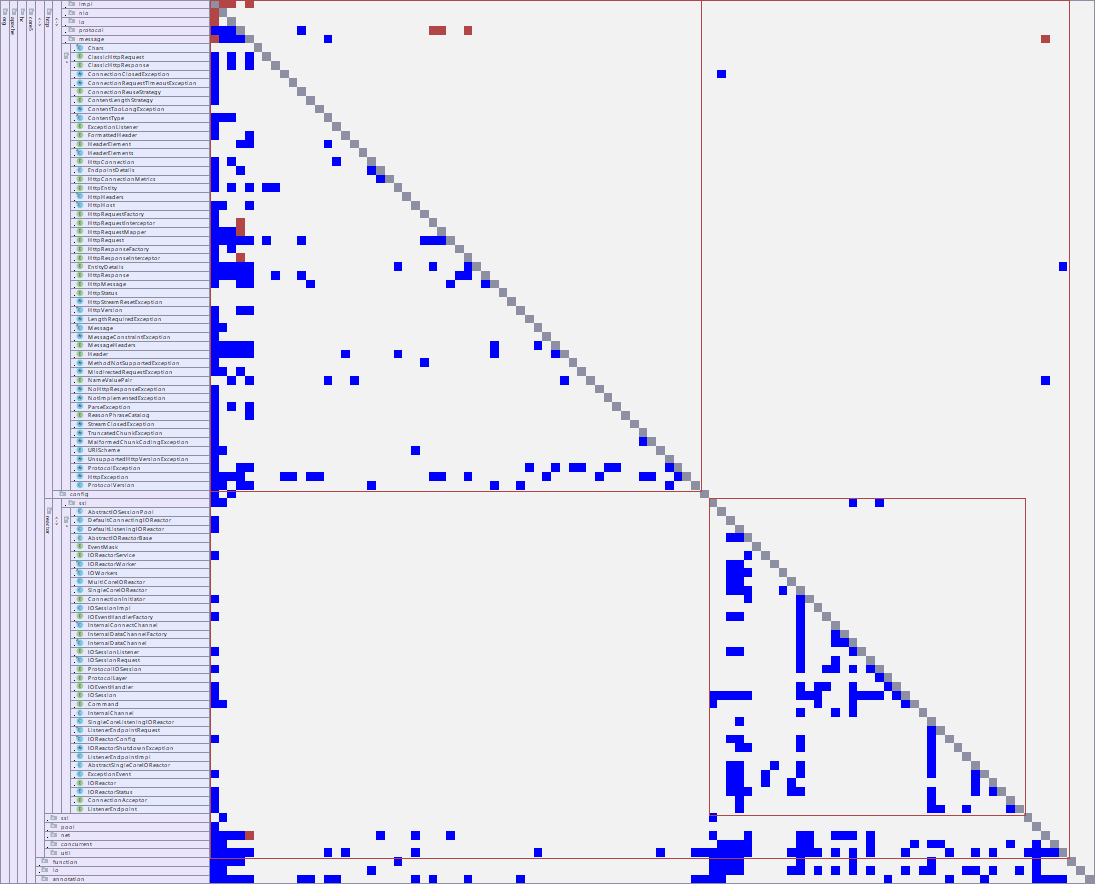
HttpCore需要实现最基本的HTTP传输，包括HttpRequest、HttpResponse等，以及具体方法比如GET和POST。这是最简单的模型，同时还要实现这些简单模型简单操作的复杂要求，包括阻塞与非阻塞的I/O模型等。

可以从代码的角度对整个HttpCore项目进行分析。本文只针对httpcomponents-core/src/main/java/org.apache.hc.core5/ 文件夹的代码进行分析，这些代码足够体现HttpCore的主体设计思想、主要需求实现和代码规范，其他文件夹中的代码与分析代码的相关性有限并且有较多重复，将不在本文报告中进行进一步阐述。



**Figure 1. Relation between components**

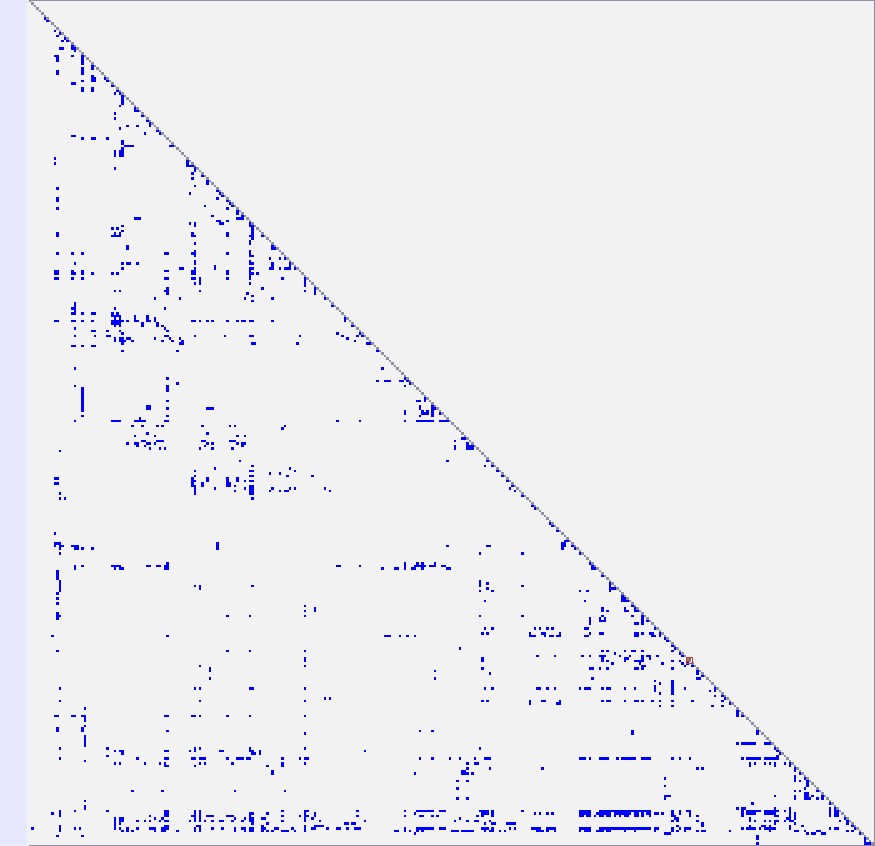
如图为本文分析的文件夹中包含的多个包之间的对应关系，不难发现其中的function、testing和ssl与代码主体的相关性有限。根据UML图对于包之间的描述，包reactor中有对包concurrent、util和io的继承，有对包io的实现；包pool中有对包io的继承和局部变量、方法或者静态方法的调用（依赖关系）；包http有对包io的继承和包net的实现。以上三个包构成了分析问题的核心。下面的三个包reactive、benchmark、http2是对包http的继承和依赖，所以包http是我们分析的重点。



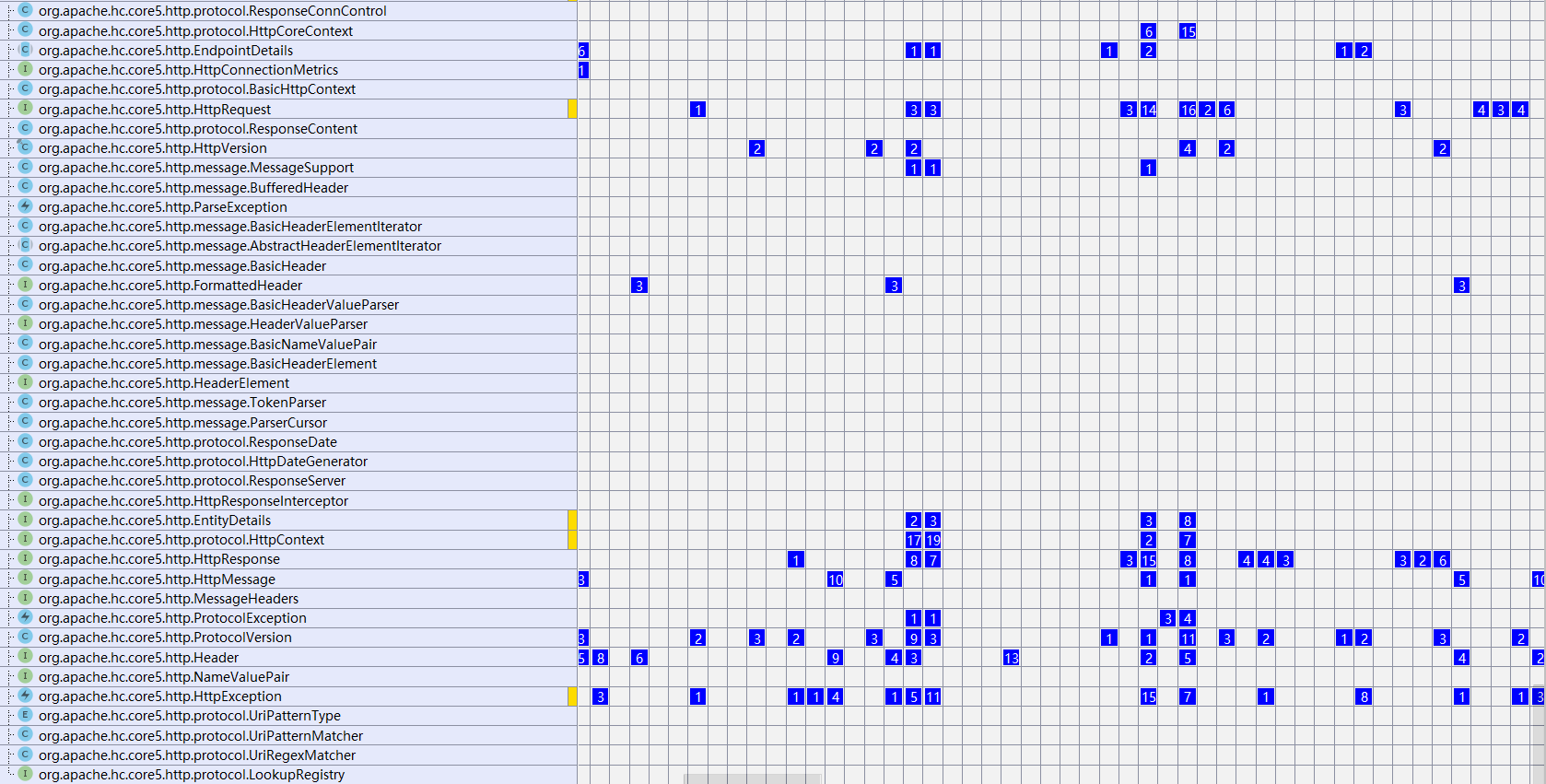
**Figure 2. DSM plot of the main frame----org.apache.hc.core5**

上图是对httpcomponents-core/src/main/java/org.apache.hc.core5/进行依赖分析(dependency structure matrix, DSM)后得到的一张DSM图像。DSM图像可以形象的展现类与类之间、类与接口之间的关系。对于大型复杂系统来说，相比起UML图而言，DSM图对于任务之间的信息交互的展示更加直观，可以明显的体现出依赖关系。比如上图中是将core5核心代码中的两个最主要的package文件夹http和reactor展开之后的结果，可以发现文件的架构呈现出高聚合性和低相关性的特点，即包与包之间的联系较少，而包内联系较为紧密。

下图是对package http的所有接口与类进行依赖矩阵分析之后得到的图。整个结构看上去较为清晰，没有明显的过于依赖其他模块的结构，说明程序设计的基本思路较为清晰，没有冗杂。同时不难发现底层的几行模块被大量其他模块使用。这说明文件整体（或者说HttpCore这个API整体）对于底层几行模块的依赖性较大。我们一般称这类模块为工具模块或者基础模块。这些模块也将成为之后对于程序分析的切入点。



**Figure 3. DSM plot of the main package----org.apache.hc.core5/http**



**Figure 4. The bottom of DSM plot of the main package----org.apache.hc.core5/http**

从上图可以发现那些基础模块包括：HttpMessage, HttpResponse, HttpRequest, HttpHeader, HttpException等，而这些恰好是HttpCore实现的基本功能。

1. **具体功能分析**



**Figure 5. The UML plot of the package of httpcore5**

以上是对于整个http包的UML分析图，其主要功能可以分为以下几块：HttpHeaders，即Http请求中的消息头部分，其最主要的实现为HttpMessage，包括HttpRequest和HttpResponse，是HttpCore最核心的内容，也是整个项目最需要实现的对象；HttpEntity，Http请求中的消息实体，包括其他附属内容；HttpConnection，包括网络服务器的连接状态；HttpException，其主要实现是协议例外。其他的包与整个功能主体的相互关联较小，本文就暂且不做过多的分析。将对每一个功能的具体实现和包之间的相互调用关系进行梳理。

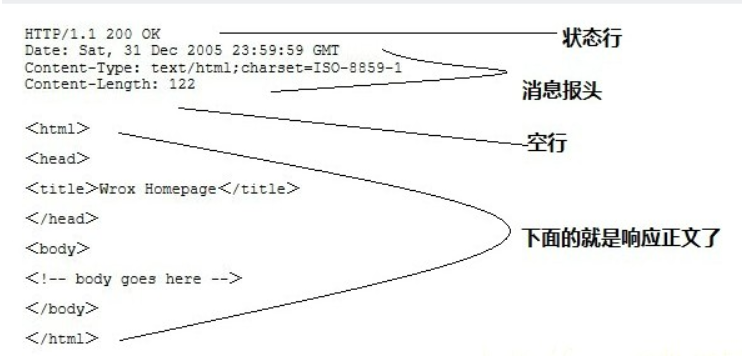
1. **HttpMessage的实现**

作为httpcomponents-core的最重要功能，HttpMessage实现了HTTP消息的有序传输，其中用到了多个接口、抽象类的设计，通过对于消息对象的抽象来达到信息隐藏和结构清晰的目的。我们先来回顾一下HttpMessage的大概组成：



**Figure 6. The basic structure of Http request message**

上图为请求消息的基本组成，期中请求行的内容是必须的，之后的内容可以视情况省略。



**Figure 7. The basic structure of Http response message**

上图为服务器端给出的响应消息的内容，状态行是必须的，之后的响应正文可以视情况省略。下面我们针对这一功能的实现进行项目的框架和源代码阅读。

在org.apache.http目录下，有一个HttpMessage接口，含有：

信息处理类方法getProtocolVersion协议版本获取, containsHeader返回消息头；

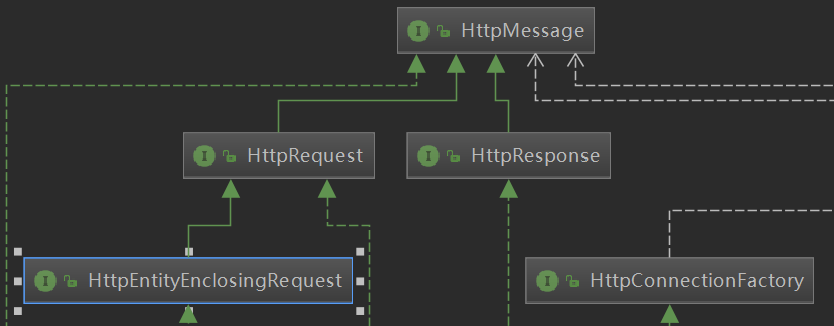
分析消息头信息方法 getHeaders, getFirstHeader, getLastHeader, getAllHeaders；

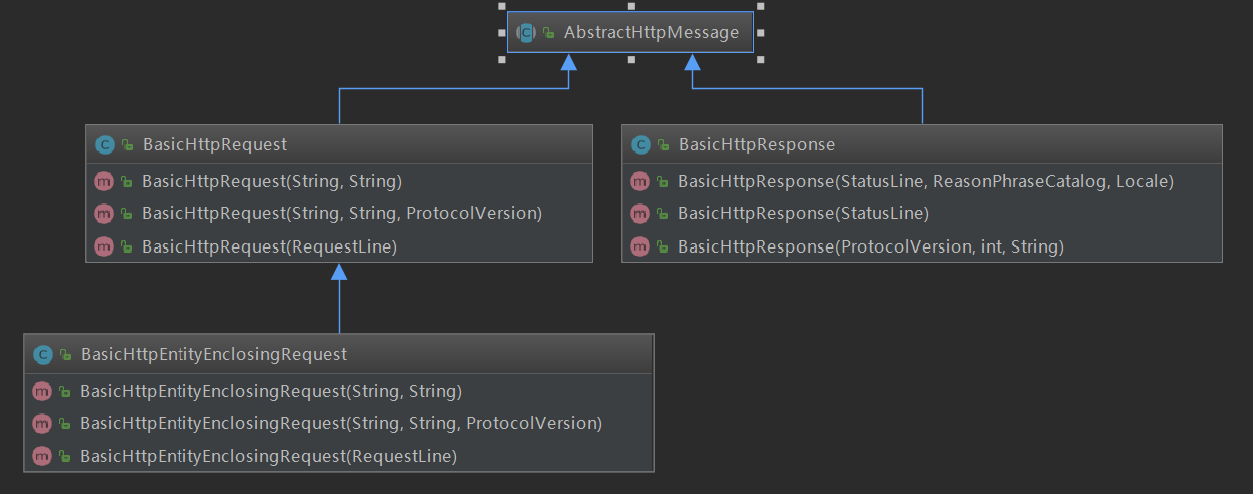
处理消息头方法 addHeader, setHeader, removeHeader, headerIterator。

这个接口体现出HttpMessage的一般操作方法，是对所有消息的抽象，特别是对所有消息头的处理。在这个基础上，有两个接口HttpRequest和HttpResponse继承了这个接口的设计，同时增加了相关消息的特殊方法。HttpRequest只加了一个getRequestLine方法，用于从消息中提取出Http请求行。HttpResponse相比请求消息多了很多其他的内容，包括getStatusLine获取状态行，setStatusLine编写状态行，setStatusCode设定状态代码，setReasonPhrase编写解释行，getEntity获取消息实体，setEntity将消息实体编写入消息，getLocale获取国际化地址，setLocale设定地区。作为一个响应消息的接口，HttpResponse需要从整个系统的其他部分去获取相关信息，完成对于HttpResponse消息的搭建。值得注意的是，接口中出现了三种对于setStatusLine的重载方法，以适用于不同的相应消息头。而考虑到Http的特殊性，代码中还对国际化进行了实现。

HttpRequest接口还被HttpEntityEnclosingRequest接口继承，出现了三个特殊方法：expectContinue用于判定是否采用expect-continue模式处理消息实体；setEntity用于将消息实体编入消息；getEntity用于获取消息实体。这个接口用于处理带有实体的请求消息。

这些接口的相互关系在package message中被非常好的继承和实现了，同时被继承的还有这些接口之间的继承关系，如下图所示：





**Figure 8. Relation between Interfaces and (Abstract) Classes**

抽象类AbstactHttpMessage实现了HttpMessage这个接口，并且对内部变量进行了严格的保护，防止外部对其进行修改，保护了数据和程序的安全。BasicHttpRequest和BasicHttpResponse继承了抽象类AbstactHttpMessage，并且分别实现了HttpRequest和HttpResponse接口，下面就分别分析一下相关代码。

BasicHttpResponse：

public class BasicHttpResponse extends AbstractHttpMessage implements HttpResponse {  
  
 private StatusLine statusline;//状态行  
 private ProtocolVersion ver;//协议版本  
 private int code;//协议编码  
 private String reasonPhrase;//原因解释  
 private HttpEntity entity;//消息实体  
 private final ReasonPhraseCatalog reasonCatalog;  
 private Locale locale;//国际化

所有内部变量都限定为private，仅能通过内部方法访问。

public BasicHttpResponse(final StatusLine statusline,  
 final ReasonPhraseCatalog catalog,  
 final Locale locale) {  
 super();  
 this.statusline = Args.*notNull*(statusline, "Status line");  
 this.ver = statusline.getProtocolVersion();  
 this.code = statusline.getStatusCode();  
 this.reasonPhrase = statusline.getReasonPhrase();  
 this.reasonCatalog = catalog;  
 this.locale = locale;  
}

public BasicHttpResponse(final StatusLine statusline) {  
 super();  
 this.statusline = Args.*notNull*(statusline, "Status line");  
 this.ver = statusline.getProtocolVersion();  
 this.code = statusline.getStatusCode();  
 this.reasonPhrase = statusline.getReasonPhrase();  
 this.reasonCatalog = null;  
 this.locale = null;  
}

public BasicHttpResponse(final ProtocolVersion ver,  
 final int code,  
 final String reason) {  
 super();  
 Args.*notNegative*(code, "Status code");  
 this.statusline = null;  
 this.ver = ver;  
 this.code = code;  
 this.reasonPhrase = reason;  
 this.reasonCatalog = null;  
 this.locale = null;  
}

HttpResponse消息的构建函数重载了三次，面向不同的Http相应消息的参数进行构建。一个消息中除了必须要有必要的参数以外（必要的参数是指协议版本、协议编码和原因字段，这些内容都包含于状态行中，所以当输入数据中有状态行时，直接调用状态行这个类的方法来调用这部分内容即可），其余部分可以选择性没有。

下面对于程序中较为值得分析的接口方法实现和内部方法实现进行一些分析。

@Override  
public StatusLine getStatusLine() {//该方法用于获取状态行  
 if (this.statusline == null) {//判定状态行是否为空，若空则构造一个然后返回  
 this.statusline = new BasicStatusLine(  
 this.ver != null ? this.ver : HttpVersion.*HTTP\_1\_1*,//默认版本号为HTTP 1.1  
 this.code,  
 this.reasonPhrase != null ? this.reasonPhrase : getReason(this.code));//如果原因字段为空，那么从编码中获取原因字段，方法见下  
 }  
 return this.statusline;  
}

protected String getReason(final int code) {//这是一个内部方法，用于从编码字段中获取原因字段  
 return this.reasonCatalog != null ? this.reasonCatalog.getReason(code,  
 this.locale != null ? this.locale : Locale.*getDefault*()) : null;

//返回原因字段，如果原因目录为空，那么就调用编码和国际化地区编码构建一个原因字段，其中国际化地区编码字段如果为空则采用系统默认编码  
}  
  
@Override  
public String toString() {//这个就比较厉害了，重写了java.lang的Object类，按照HttpRequestMessage的格式构建了这个方法  
 final StringBuilder sb = new StringBuilder();//创建一个字符串  
 sb.append(getStatusLine());//添加状态行  
 sb.append(' ');  
 sb.append(this.headergroup);//添加消息头部的包  
 if (this.entity != null) {//如果有实体的话添加实体  
 sb.append(' ');  
 sb.append(this.entity);  
 }  
 return sb.toString();//返回这整个字符串  
}

以上是对HTTP Message的主要框架的解析。实际上这个框架中还有大量的细节值得去注意，除了以上提到的对于客户端请求的包装处理，对服务器端响应的包装处理以外，还有对于header（请求头数据包）、entity（请求和响应实体）的构造和搭建。

在阅读源代码的过程中，不难发现源代码中有一定数量的对于Oracle提供的Java内核中的方法的重写，比如toString()，hasNext()等，这些方法的重写对于之后构建Http消息有益，可以在不冲突的情况下使代码变得更加容易编写和阅读。

**【参考文献】**

[1]<https://www.w3.org/Protocols/rfc2616/rfc2616.html> , last access 2018.10.31 原文如下：“The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.”