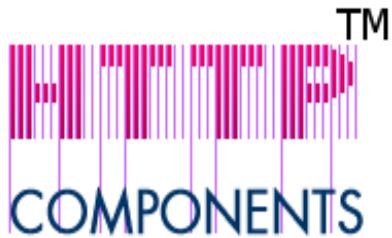


Report on Httpcomponents

张宇轩, 2017K8009908041



Contents

1 Preface	3
2 Related Introduction	4
2.1 http	4
2.2 https	4
2.3 Proxy Server	4
2.4 Cookies	4
2.5 Request Format	4
2.6 URL	5
2.7 Http Request Methods	5
3 About HttpComponents	6
3.1 Major Function	6
3.2 Component Structure	6
3.3 HttpComponents Core	7
3.3.1 Contents of HttpCore	8
3.4 BasicHttpRequest	9
3.4.1 Methods in BasicHttpRequest	9
3.4.2 Constructors	10

3.5	BasicHttpResponse	15
3.5.1	Methods in BasicHttpResponse	15
3.5.2	Constructors	17
4	Functions	19
4.1	Setting up Connections	20
4.2	Receiving Requests	20
4.3	Processing Requests	21
4.4	Accessing Resources	21
4.5	Building Response Message	22

1 Preface

由于先前的课程中并没有讲过关于网络与信息方面的知识，笔者之也并没有接触过。而这次的选题是HttpComponents，涉及到大量的网络方面的知识，所以在准备这篇报告的第一阶段笔者的主要精力是放在研究一些基本的网络方面的知识。要讲到HttpComponents，就先要讲一下关于http的各种知识。

2 Related Introduction

2.1 http

从万维网（World Wide Web）服务器传输超文本到本地浏览器，基于请求与响应，无状态的，应用层的协议。从本地角度去看http协议，就相当于浏览器访问一个url，然后就得到相应的web页面。从服务器角度看http协议，就相当于客户端（浏览器）链接远程http服务器，服务器返回数据，浏览器接收、解析数据之后显示出来。

2.2 https

利用SSL（此处：基于应用层的访问控制）进行传输层加密的http传输协议，更加安全。

$\text{https} = \text{http} + \text{SSL}$

2.3 Proxy Server

代理服务器：代理个人网络从互联网服务商那里获取信息（通过在二者间建立非直接的连接），可以保证安全。

2.4 Cookies

网站保存在用户端的含有用户信息以及行为的文本，用以弥补http协议的无状态性。举两个例子：

- 1. 网站上的记住密码。网站将用户的身份和密码经过加密cookies存在用户硬盘中，下次登录的时候就不用手动输密码了。
- 2. 上下文相关网址。如果上一个页面会对下一个页面产生影响，那次是就必须使用cookies。
比如一个购物网站，上一个页面选好商品，接下来的页面进行支付，支付页面就必须知道上一个页面中购买了什么东西，此时就需要cookies。

2.5 Request Format

客户端发送一个请求到服务器的请求消息格式如 Figure 1 所示（后面还会说到），请求头部的最后会有一个空行，表示请求头部结束，接下来为请求数据。



Figure 1: Request Format

2.6 URL

URL(Uniform Resource Locator, 统一资源定位符): 一种资源位置的抽象唯一识别方法。

URL组成: <协议>/<主机>/<端口>/<路径> (端口和路径可以省略, 端口默认80), 如 Figure 2 所示:

另外需要注意URI(Uniform Resource Identifier, 统一资源标识符), 用来表示web上每一种可用的资源, 与URL不一样。



Figure 2: Request Format

2.7 Http Request Methods

Http 协议共有9种请求方法, 如 Figure 3 所示, 其中最常用的两种方法是 GET 和 POST, 其对比如 Figure 4 所示。

GET - 从指定的资源请求数据。

POST - 向指定的资源提交要被处理的数据。

方法	描述
GET*	向特定资源发出请求 请求指定的页面信息, 并返回实体主体。
HEAD	类似于 GET 请求, 响应体不会返回, 获取包含在小消息头中的原信息 返回的响应中没有具体的内容, 用于获取报头。
POST	向指定资源提交数据进行处理请求 (例如提交表单或者上传文件), 数据被包含在请求体中。 可能会导致新的资源的建立或已有资源的修改。
PUT*	向指定资源位置上传最新内容 从客户端向服务器传送的数据取代指定文档的内容
PATCH	是对 PUT 方法的补充, 用来对已知资源进行局部更新。
DELETE	请求服务器删除指定的页面 (requestURL对应资源)。
CONNECT	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
OPTIONS	返回服务器针对特定资源所支持的HTML请求方法, 或向web服务器发送测试服务器功能 允许客户端查看服务器的性能。
TRACE	回显服务器收到的请求, 主要用于测试或诊断。

Figure 3: Request Format

	get	post
cache	请求可被缓存（主动cache）	请求不会被缓存
刷新	无影响	重新提交请求（浏览器提醒）
编码	只有url编码 application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data
	请求只应当用于取回数据	
数据长度	请求有长度限制	请求对数据长度没有要求
历史记录	请求保留在浏览器历史记录中	请求不会保留在浏览器历史记录中

Figure 4: Get and Post

本文要介绍的HttpComponents就是用于提供对于http服务器的访问功能。

3 About HttpComponents

HttpComponents: 用于提供对于http服务器的访问功能的超文本传输协议，其对HTTP底层协议进行了很好的封装。

在构建HTTP客户端或者服务器端应用中很常见，比如WEB浏览器、爬虫、HTTP代理、WEB服务库、基于调整或扩展HTTP协议的分布式通信系统 etc.

3.1 Major Function

实现http方法: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH

对https协议的支持 (https = http + SSL)

对代理服务器的支持

对cookies的支持

支持在特定的执行上下文（HTTP上下文）中执行HTTP消息——http不行（无状态、面向响应请求）

3.2 Component Structure

HttpComponents 的组件结构如 Figure 5 所示，包含 HttpComponents Core 和 HttpComponents AsyncClient，本次分析的是 HttpComponents Core。

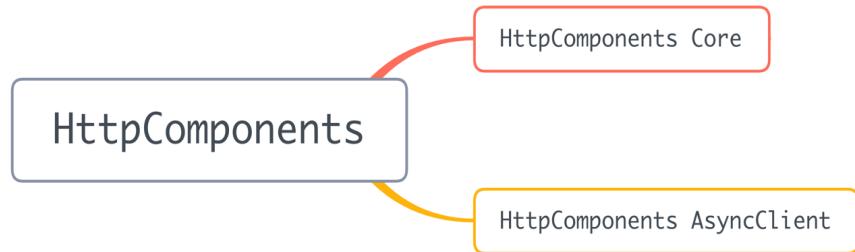


Figure 5: Component Structure of HttpComponents

3.3 HttpComponents Core

HttpComponents Core 简称 HttpCore，其实现基本http协议的组件，用于搭建客户端和服务器端的http服务。HttpCore 希望在实现基本的http功能的基础上提高性能与平衡性，如 Figure 6 所示：

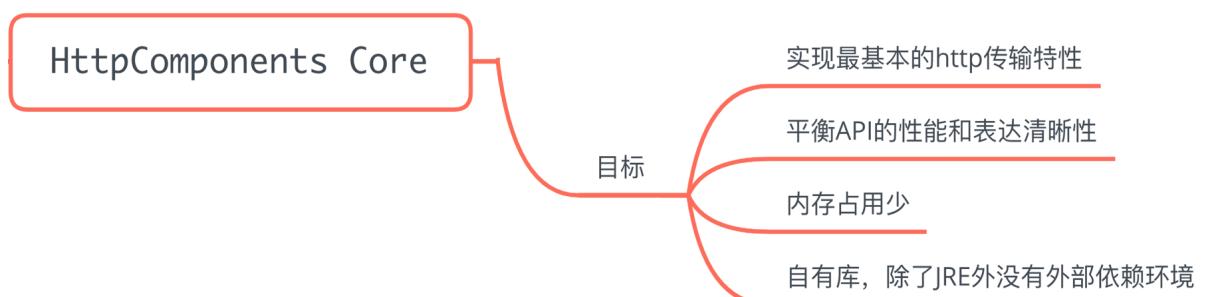


Figure 6: Goal of HttpCore



Figure 7: Guide Book

3.3.1 Contents of HttpCore

HttpCore 共有 241 个类, 如 Figure 8 所示。其中将会重点介绍 BasicHttpRequest 和 BasicHttpResponse 这两个类。

The figure consists of four tables labeled (a), (b), (c), and (d), each listing a set of Java classes. The classes are color-coded: blue for core interfaces and red for concrete implementations.

- Table (a):** Contains basic message-related classes.
 - AbstractConnPool
 - AbstractHttpClientConnection
 - AbstractHttpEntity
 - AbstractHttpMessage
 - AbstractHttpParams
 - AbstractHttpServerConnection
 - AbstractMessageParser
 - AbstractMessageWriter
 - AbstractSessionInputBuffer
 - AbstractSessionOutputBuffer
 - Args
 - Asserts
 - BasicConnFactory
 - BasicConnPool
 - BasicFuture
 - BasicHeader
 - BasicHeaderElement
 - BasicHeaderElementIterator
 - BasicHeaderIterator
 - BasicHeaderValueFormatter
- Table (b):** Contains header and buffer-related classes.
 - BasicHeaderValueParser
 - BasicHttpContext
 - BasicHttpEntity
 - BasicHttpEntityEnclosingRequest
 - BasicHttpParams
 - BasicHttpProcessor
 - BasicHttpRequest
 - BasicHttpResponse
 - BasicLineFormatter
 - BasicLineParser
 - BasicListHeaderIterator
 - BasicNameValuePair
 - BasicPoolEntry
 - BasicRequestLine
 - BasicStatusLine
 - BasicTokenIterator
 - BHttpConnectionBase
 - BufferedHeader
 - BufferedHttpEntity
 - BufferInfo
- Table (c):** Contains stream and connection-related classes.
 - ByteArrayBuffer
 - ByteArrayEntity
 - Cancellable
 - CharArrayBuffer
 - CharsetUtils
 - ChunkedInputStream
 - ChunkedOutputStream
 - ConnectionClosedException
 - ConnectionConfig
 - ConnectionConfig.Builder
 - ConnectionReuseStrategy
 - ConnFactory
 - ConnPool
 - ConnPoolControl
 - ConnSupport
 - Consts
 - ContentLengthInputStream
 - ContentLengthOutputStream
 - ContentLengthStrategy
 - ContentProducer
- Table (d):** Contains protocol and factory-related classes.
 - ContentTooLongException
 - ContentType
 - Contract
 - CoreConnectionPNames
 - CoreProtocolPNames
 - DefaultBHttpClientConnection
 - DefaultBHttpClientConnectionFactory
 - DefaultBHttpServerConnection
 - DefaultBHttpServerConnectionFactory
 - DefaultConnectionReuseStrategy
 - DefaultedHttpConnection
 - DefaultedHttpParams
 - DefaultHttpClientConnection
 - DefaultHttpRequestFactory
 - DefaultHttpRequestParser
 - DefaultHttpRequestParserFactory
 - DefaultHttpRequestWriter
 - DefaultHttpRequestWriterFactory
 - DefaultHttpResponseFactory
 - DefaultHttpResponseParser

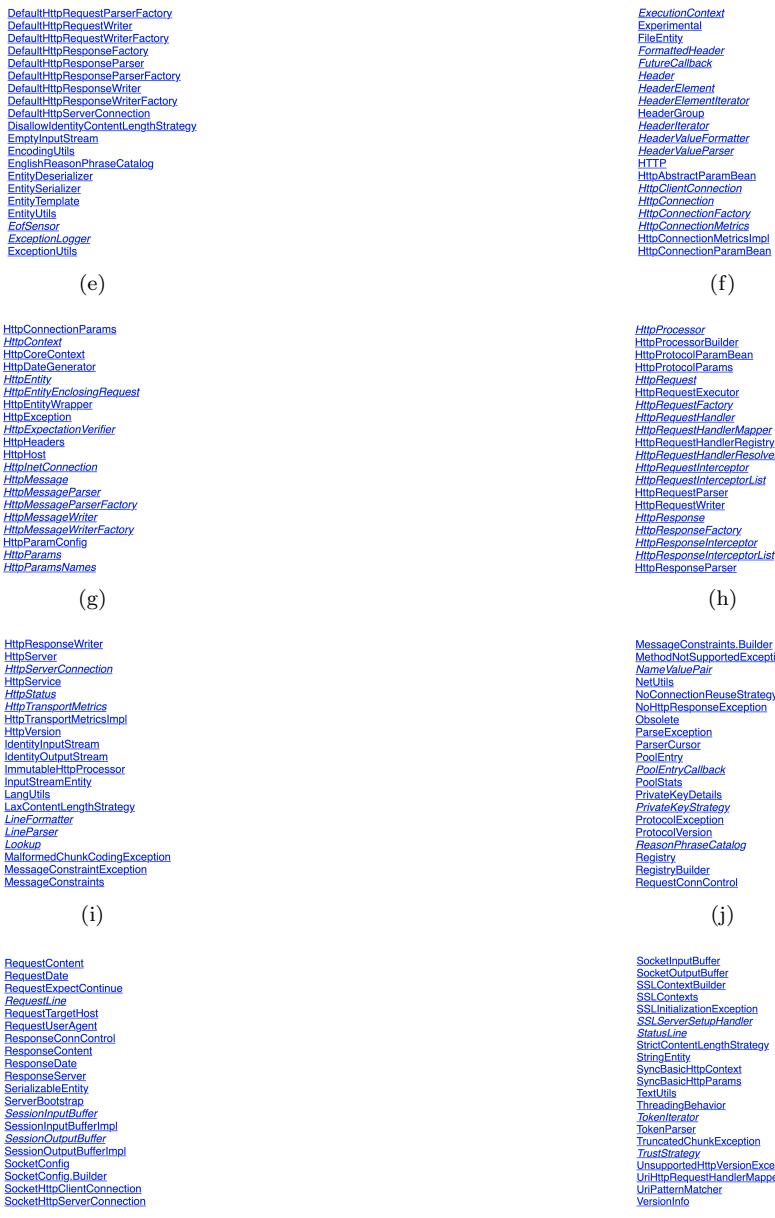


Figure 8: Classes of HttpCore

3.4 BasicHttpRequest

路径: org.apache.http.message

继承: AbstractHttpMessage

实现接口: HttpRequest

3.4.1 Methods in BasicHttpRequest

BasicHttpRequest 中包含了从别的类中继承来的方法, 以及其独有的, 其关系如 Figure 9 所示。

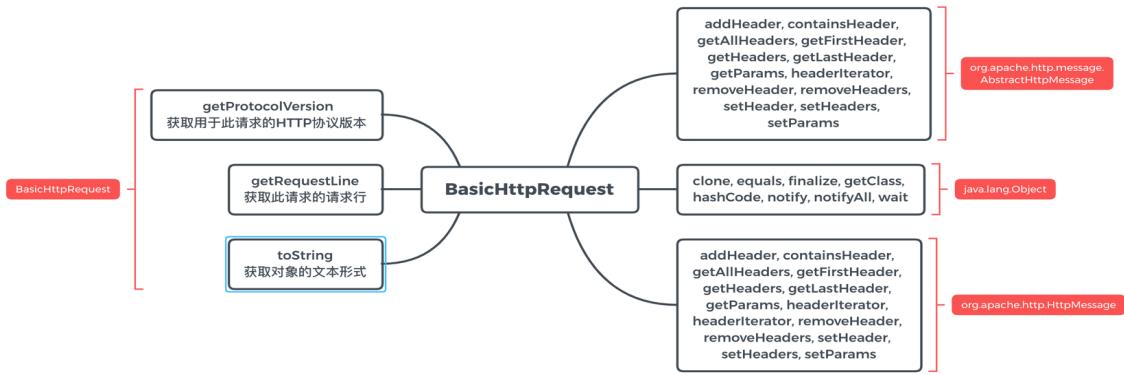


Figure 9: Methods in BasicHttpRequest

3.4.2 Constructors

构造器: 一个类里用于建立对象的方法，与类名相同，没有返回类型，不会被继承，且不会有范围修饰符。
 Java允许构造器重载（一个类被允许拥有多个接受不同参数种类的构造器同时存在，方法名称相同，参数列表不同）
 如果一个类中没有构造方法，那么编译器会为类加上一个默认的构造方法。构造器在创建对象的时候调用，为正在创建的对象的成员变量赋初值。

一个默认的构造方法的例子：

```

1  /* Default Constructor */
2  public ClassName() {
3
4 }
```

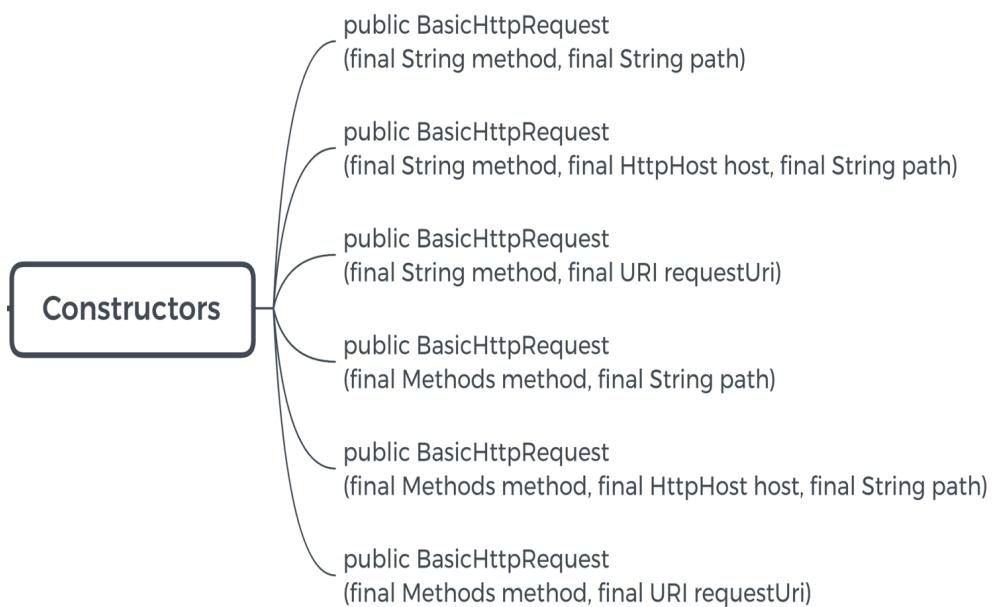


Figure 10: Constructors in BasicHttpRequest

接下来分别介绍 BasicHttpRequest 中的构造器

```

1  /**
2   * Creates request message with the given method and request path.
3   *
4   * @param method request method.
5   * @param path request path.
6   */
7 public BasicHttpRequest( final String method, final String path) {
8     super();
9     this.method = method;
10    if (path != null) {
11      try {
12        setUri(new URI(path));
13      } catch (final URISyntaxException ex) {
14        this.path = path;
15      }
16    }
17 }

```

参数	参数说明
method	请求方法
path	请求路径
功能：利用给定的方法和请求路径创建请求消息	

```

1 /**
2  * Creates request message with the given method, host and request path.
3  *
4  * @param method request method.
5  * @param host request host.
6  * @param path request path.
7  *
8  * @since 5.0

```

```

9  */
10 public BasicHttpRequest( final String method, final HttpHost host, final String path)
11 {
12     super();
13     this.method = Args.notNull(method, "Method.name");
14     this.scheme = host != null ? host.getSchemeName() : null;
15     this.authority = host != null ? new URIAuthority(host) : null;
16     this.path = path;
}

```

参数	参数说明
method	请求方法
host	请求主机
path	请求路径

功能：使用给定的方法、主机和请求路径创建请求消息

```

1 /**
2  * Creates request message with the given method, request URI.
3  *
4  * @param method request method.
5  * @param requestUri request URI.
6  *
7  * @since 5.0
8 */
9 public BasicHttpRequest( final String method, final URI requestUri) {
10
11     super();
12     this.method = Args.notNull(method, "Method.name");
13     setUri(Args.notNull(requestUri, "Request.URI"));
}

```

参数	参数说明
method	请求方法
requesturi	请求uri
功能：使用给定的方法和请求uri创建请求消息	

```

1  /**
2   * Creates request message with the given method and request path.
3   *
4   * @param method request method.
5   * @param path request path.
6   *
7   * @since 5.0
8   */
9 public BasicHttpRequest( final Methods method, final String path) {
10
11     super();
12
13     this.method = Args.notNull(method, "Method").name();
14
15     if (path != null) {
16
17         try {
18
19             setUri(new URI(path));
20
21         } catch (final URISyntaxException ex) {
22
23             this.path = path;
24
25         }
26
27     }
28
29 }
```

参数	参数说明
method	请求方法
path	请求路径
功能：利用给定的方法和请求路径创建请求消息	

```

1 /**
2  * Creates request message with the given method, host and request path.
```

```

3   *
4   * @param method request method.
5   * @param host request host.
6   * @param path request path.
7   *
8   * @since 5.0
9   */
10 public BasicHttpRequest( final Methods method, final HttpHost host, final String path)
11 {
12     super();
13     this.method = Args.notNull(method, "Method").name();
14     this.scheme = host != null ? host.getSchemeName() : null;
15     this.authority = host != null ? new URIAuthority(host) : null;
16     this.path = path;
}

```

参数	参数说明
method	请求方法
host	请求主机
path	请求路径
功能：使用给定的方法、主机和请求路径创建请求消息	

```

1 /**
2  * Creates request message with the given method, request URI.
3  *
4  * @param method request method.
5  * @param requestUri request URI.
6  *
7  * @since 5.0
8  */
9 public BasicHttpRequest( final Methods method, final URI requestUri) {
10     super();
11     this.method = Args.notNull(method, "Method").name();

```

```
12     setUri(Args.notNull(requestUri, "Request.URI"));  
13 }
```

参数	参数说明
method	请求方法
requesturi	请求uri
功能：使用给定的方法和请求uri创建请求消息	

3.5 BasicHttpResponse

路径: org.apache.http.message

继承: AbstractHttpMessage

实现接口: HttpResponse

3.5.1 Methods in BasicHttpResponse

BasicHttpResponse 中包含了从别的类中继承来的方法，以及其独有的，其关系如 Figure 11 所示。

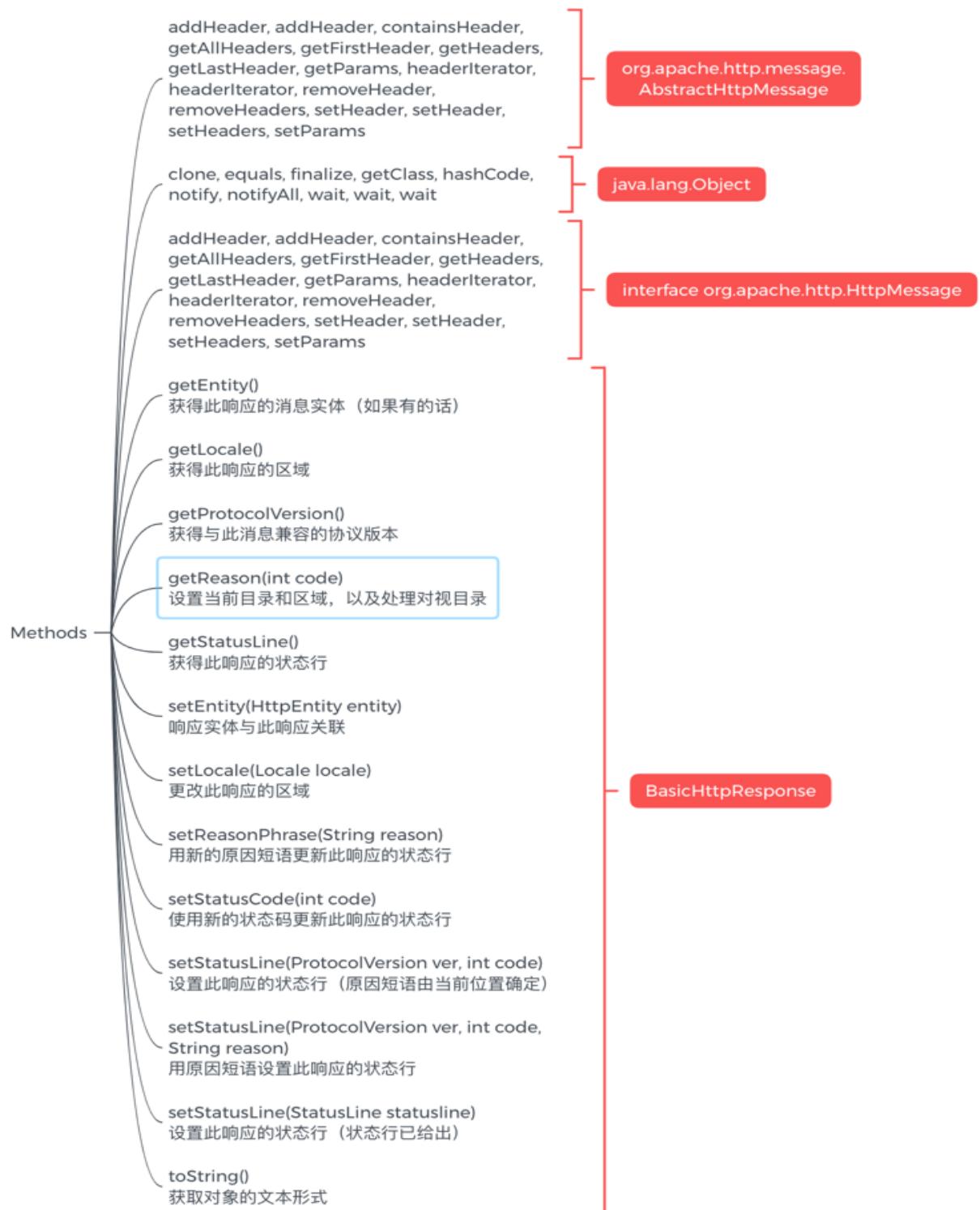


Figure 11: Methods in BasicHttpResponse

3.5.2 Constructors

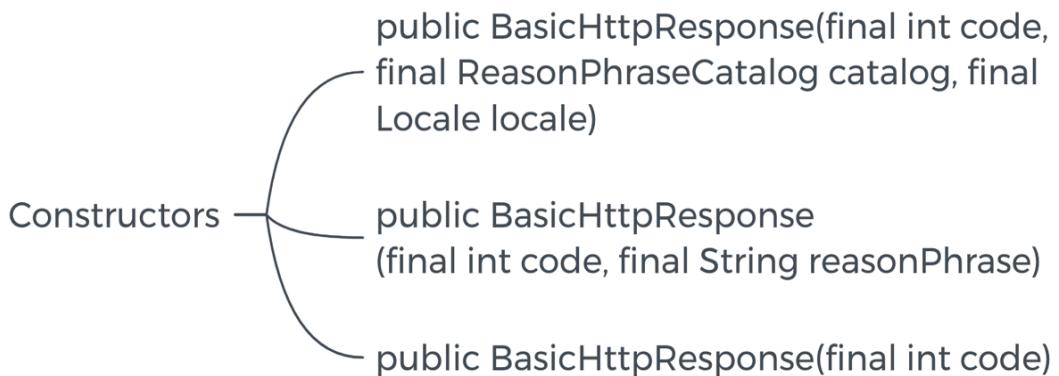


Figure 12: Constructors in `BasicHttpResponse`

接下来分别介绍 `BasicHttpResponse` 中的构造器

```
1  /**
2  * Creates a new response.
3  *
4  * @param code          the status code
5  * @param catalog       the reason phrase catalog, or
6  *                      {@code null} to disable automatic
7  *                      reason phrase lookup
8  * @param locale        the locale for looking up reason phrases, or
9  *                      {@code null} for the system locale
10 */
11 public BasicHttpResponse(
12     final int code,
13     final ReasonPhraseCatalog catalog,
14     final Locale locale) {
15
16     super();
17
18     this.code = Args.positive(code, "Status-code");
19
20     this.reasonCatalog = catalog != null ? catalog : EnglishReasonPhraseCatalog.
21
22     INSTANCE;
23
24     this.locale = locale;
25 }
```

参数	参数说明
code	响应的状态码*
catalog	原因短语目录，或用 null 来禁用自动查找原因短语
locale	查找原因短语的区域设置，或用 null 代表系统区域设置
功能：创建新的响应	

其中有一个新的概念：status code，状态码，这里介绍一下：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

Figure 13: Status Code

```

1 /**
2  * Creates a new response.
3 *
4  * @param code          the status code of the response
5  * @param reasonPhrase the reason phrase to the status code, or {@code null}
6 */
7 public BasicHttpResponse( final int code, final String reasonPhrase ) {
8     this .code = Args .positive( code, "Status_code" );
9     this .reasonPhrase = reasonPhrase;
10    this .reasonCatalog = EnglishReasonPhraseCatalog .INSTANCE;
11 }
```

参数	参数说明
code	响应的状态码
catalog	状态码的原因短语, 或空
功能 : 创建新的响应	

```

1  /**
2   * Creates a new response.
3   *
4   * @param code          the status code of the response
5   */
6 public BasicHttpResponse(final int code) {
7
8     this.code = Args.positive(code, "Status_code");
9
10    this.reasonPhrase = null;
11
12    this.reasonCatalog = EnglishReasonPhraseCatalog.INSTANCE;
13
14 }

```

参数	参数说明
code	响应的状态码
功能 : 创建新的响应	

4 Functions

Http请求处理的过程包括以下几个步骤:

建立连接 (Setting up Connections)

接收请求 (Receiving Requests)

处理请求 (Processing Requests)

访问资源 (Accessing Resources)

构建响应 (Building Response Message)

发送响应 (Sending out Response)

整个过程如 Figure 14 所示:

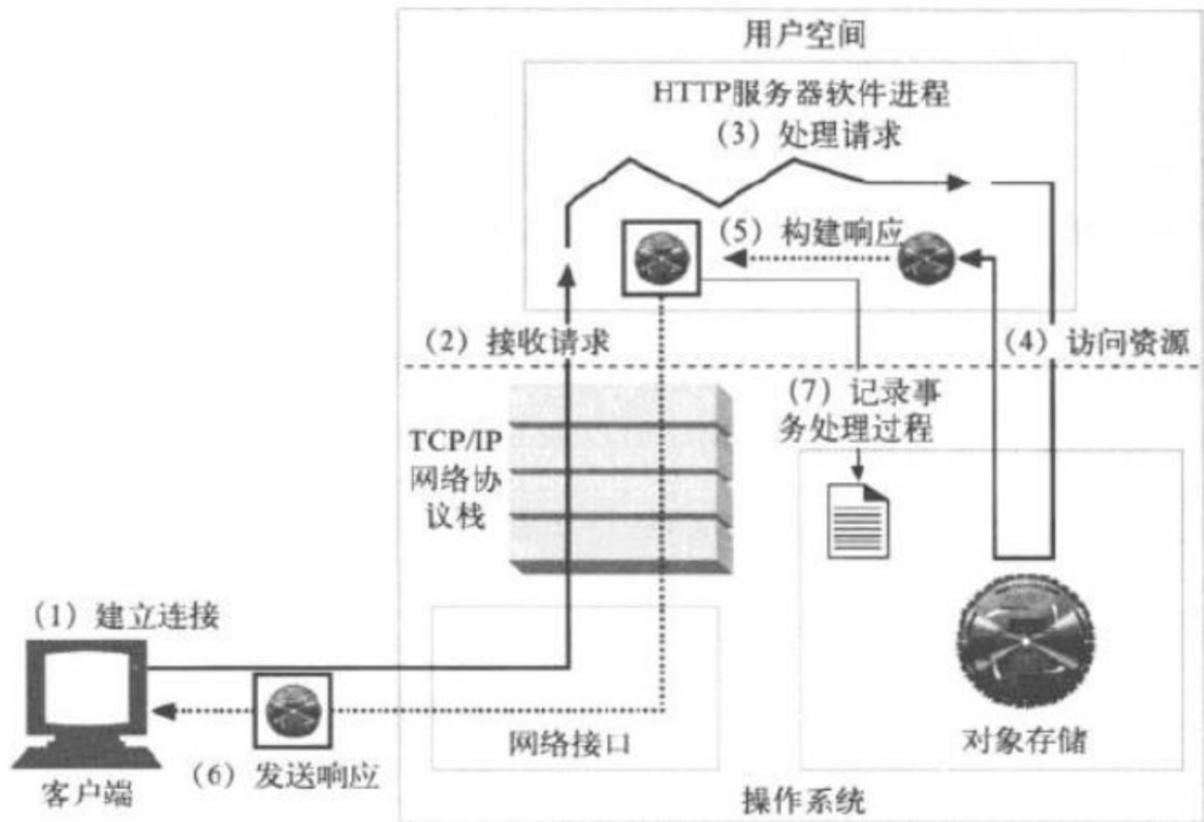


Figure 14: Http Request Processing

4.1 Setting up Connections

建立连接之前先要进行域名解析（把域名指向网站空间IP，DNS服务器把域名解析到IP地址）。

建立连接：接收或拒绝连接请求

4.2 Receiving Requests

HTTP请求报文：由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成，如 Figure 15 所示。

接收请求：接收客户端请求报文中对某资源的请求。



Figure 15: Request Format

4.3 Processing Requests

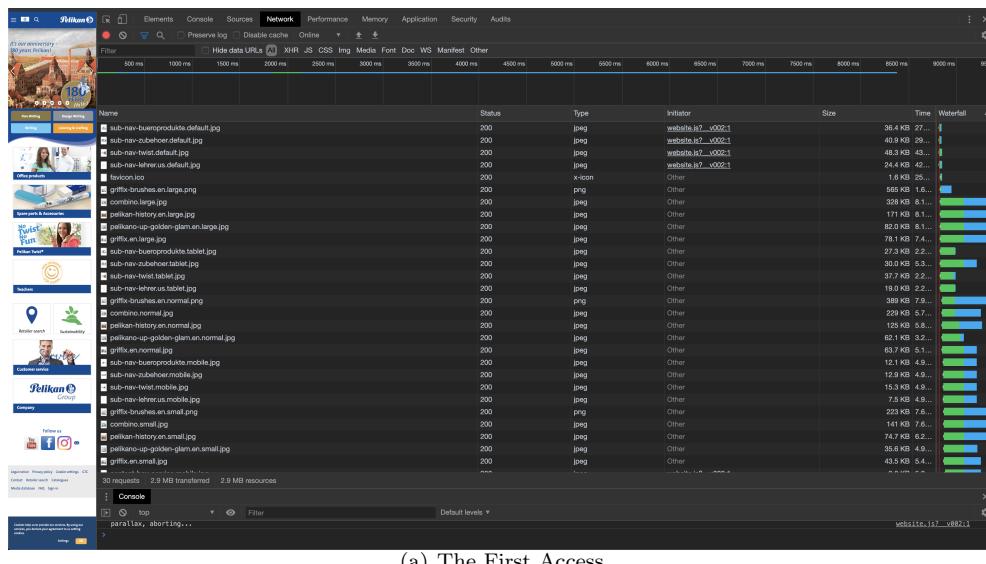
处理请求：服务器对请求报文进行解析，获取请求的资源及请求方法等相关信息。根据方法、资源、首部和可选的主体部分对请求进行处理。

(之前说过, http常用的请求方法: GET、POST、PUT、DELETE、PATCH、HEAD、OPTIONS、TRACE、CONNECT)

4.4 Accessing Resources

访问资源：服务器获取请求报文中请求的资源web服务器（存放了web资源的服务器），负责向请求者提供对方请求的静态资源或动态运行后生成的资源。

如果请求静态资源，客户端发起询问上一次修改到现在之间有没有再修改。若返回状态码为服务器端没修改（304），浏览器会直接读取本地的该资源的缓存文件（负载均衡）。如 Figure 16 所示，可以看到第二次访问同一对象的时候直接从 cache 里面拿数据。



(a) The First Access

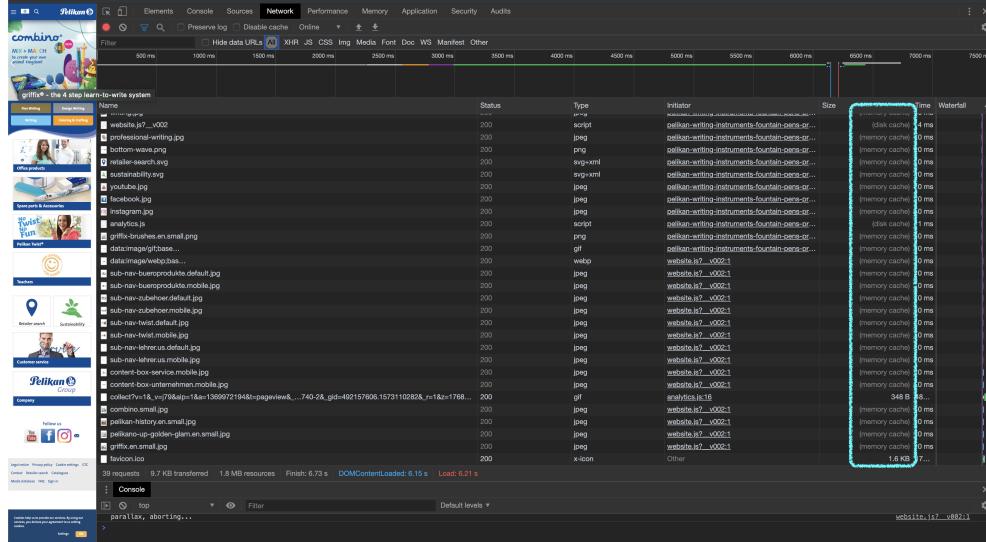


Figure 16. T = 10 K

4.5 Building Response Message

构建响应报文：一旦Web服务器识别除了资源，就执行请求方法中描述的动作，并返回响应报文。响应报文中包含有响应状态码、响应首部，如果生成了响应主体的话，还包括响应主体。

BasicHttpRequest

Request Message

method

host

path

requestUri

BasicHttpRequest(method, path)

BasicHttpRequest(method, host, path)

BasicHttpRequest(method, requestUri)

getEntity()

setEntity()

getProtocolVersion()

getRequestLine()

toString()

BasicHttpResponse

Response Message

code

catalog

locale

version

reason

Phrase

statusline

reason

BasicHttpResponse(code, catalog, locale)

BasicHttpResponse(code, reasonPhrase)

BasicHttpResponse(code)

getEntity()

getLocale()

getProtocolVersion()

getReason(int code)

getStatusLine()

setEntity(entity)

setLocale(locale)

setReasonPhrase(reason)

setStatusCode(code)

setStatusLine(version, code)

setStatusLine(version, code, reason)

setStatusLine(statusline)

toString()

```
HttpMessage  
BasicHttpResponse
```

```
name  
header  
value  
params  
addHeader(header)  
addHeader(name, value)  
containsHeader(name)  
getAllHeaders()  
getFirstHeader(name)  
getHeaders(name)  
getLastHeader(name)  
getParams()  
getProtocolVersion()  
headerIterator()  
headerIterator(name)  
removeHeader(header)  
removeHeaders(name)  
setHeader(header)  
setHeader(name, value)  
setHeaders(headers)  
setParams(params)
```