Http Components 阅读报告

吴承一 2017K8009908007

在浏览器的地址栏中输入一个 URL, 相应的 Web 页面就可以精准地呈现在我们的面前。这个看起来很直观的过程，其实背后包含了客户端与服务器之间一系列的交互操作。我们日常见到的网页基本都是以 http 开头的（当然后来越来越多的网页演变成了 https，不过这暂且不研究），也就是说客户端和服务器之间通过 http 协议进行信息的传递。超文本传输协议(http)是目前互联网上极其普遍的传输协议，它为构建功能丰富的网页提供了强大的支持。http 同样可以在其它的一些场景中使用，如游戏服务器和客户端的传输协议、网络爬虫、HTTP 代理、网站后台数据接口等。Apache 开源组织的 HttpComponents 项目，主要功能便是提供对 http 服务器的访问功能。它对 HTTP 底层协议进行了很好的封装，专门用来简化 HTTP 客户端与服务器的各种通讯编程。

现在的 HttpComponents 项目包含多个子项目，例如 HttpComponents Core 和 HttpComponents Client，其中 HttpCore 是一组底层 http 传输协议组件，后者则是建立在 HttpCore 之上的 Http 客户端管理组件，依赖 HttpCore 组件实现数据传输。重点关注的便是 HttpCore 部分，从名字上就能看出来，它也是整个项目的核心。

上面提到过，HttpCore 是对 HTTP 协议的基础封装的一套组件。根据 Apache 的官方教程，我们可以用它来建立客户端、代理、服务端 Http 服务，它支持同步异步服务，以及一系列基于阻塞和非阻塞 IO 模型。看起来似乎非常复杂，但是直观地来说，它所实现的主要功能无非就是两件事, 建立连接和传输数据。我所感兴趣的就是它传输数据这一部分的功能。当然，这里的传输数据是一个广义的概念，并不是像网盘一样上传下载数据才叫传输，在建立连接之后和服务器的任何交互行为本质上都是传输数据, 客户端发送包含请求的数据包到服务器，然后服务器返回包含响应的数据包。相关的一对发送请求和返回响应也构成了一次完整的 http 通信过程，这一点后面会再进行探讨。

关于 http 数据传输的探讨，要从 HttpMessage 开始。什么是 HttpMessage？字面上理解，就是遵循 HTTP 协议发送的消息，也就是客户端和服务器之间用于交互的数据。HttpMessage 包含从客户端到服务器的请求和服务器到客户端的响应，即：

HttpMessage = Request | Response

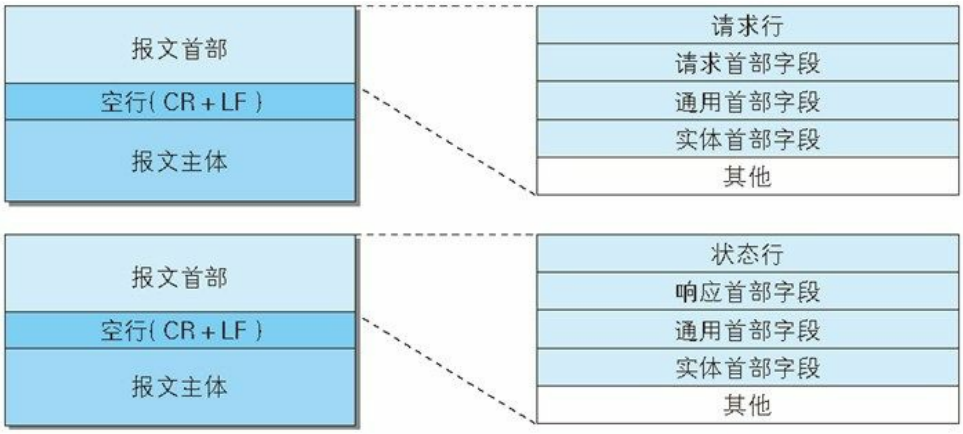
Request 和 Response 都采用一个通用的消息格式来传输消息实体。我们有必要详细了解一下这个通用消息格式。因为要传输一个消息，自然需要先了解如何表示它。

首先我们对这个用于 HTTP 协议交互的信息（有时我们也将其称作报文）有一个概括性的定义：

HttpMessage = Message Line + Message Header + Message Body(optional)

翻译一下，也就是说一个 HTTP 消息包含起始行（Message Line），若干消息头（Message Header）--也叫首部字段，以及可选的消息主体（Message Body），这是从内容上的划分。

从结构本身来看，HTTP 消息是由多行数据构成的字符串文本，可以大致将其分为首部和主体两个部分，之间用一个空行来进行分隔。首部就包含刚才提到的 Message Line 和 Message Header，主体就是 Message Body，自然，这一部分并不一定所有消息都有的。



请求消息和响应消息在消息头上有所不同。他们的起始行分别为请求行和状态行，也就是 RequestLine 和 StatusLine。RequestLine 的结构为：

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

StatusLine 的结构为：

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

请求行包含用于请求的方法，同时也要请求 URI 和 HTTP 版本。状态行包含表明响应结果的状态码，原因短语和 HTTP 版本。首部字段（也就是 header）则包含表示这条消息条件以及属性的一些相关信息。需要注意的是，请求行和状态行都必须包括 HTTP 版本信息，毕

竟如果客户端和服务器所使用的 HTTP 协议版本都不一样，那消息传输的正确性和可靠性就无从谈起了。

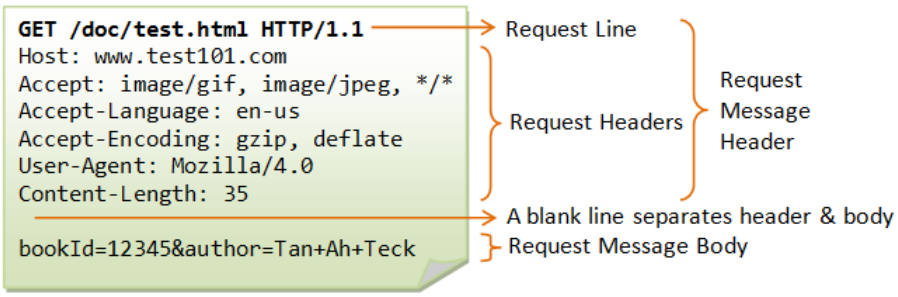
上面是对于某博客网站进行访问的请求和响应消息的具体实例（来自于互联网）。它们的内容比较复杂，但我们可以看出这两个例子是完全符合我们上面提到的通用消息格式的。

关于 HttpMessage，HttpCore 中为我们提供了相应的接口，下面是其内容：

```
public interface HttpMessage {

    ProtocolVersion getProtocolVersion();

    boolean containsHeader(String name);

    Header[] getHeaders(String name);

    Header getFirstHeader(String name);

    Header getLastHeader(String name);

    Header[] getAllHeaders();

    void addHeader(Header header);

    void addHeader(String name, String value);

    void setHeader(Header header);
```

```
    void setHeader(String name, String value);

    void setHeaders(Header[] headers);

    void removeHeader(Header header);

    void removeHeaders(String name);

    HeaderIterator headerIterator();

    HeaderIterator headerIterator(String name);

    @Deprecated
    HttpParams getParams();

    @Deprecated
    void setParams(HttpParams params);

}
```

可以看出来，除了一个获取协议版本的方法，其余基本都是对 header 的一些方法，可以用于取出、添加、移除、枚举这些 header 等。由此也可以看出 header 在一条消息中具有非常重要的作用。在下图的请求消息中，header 就包含了文件格式，语言，编码方式，内容长度等等一系列的关键信息。这也符合我们前面提到的，header 包含的是表示这条消息条件以及属性的一些相关信息。除了图示的内容之外，有关 cookie, user-agent，连接方式，host 等方面的信息也可以被包含在 header 中。



HttpMessage 接口是不直接区分 request 和 response，但它有两个子接口 HttpRequest 和 HttpResponse 。HttpRequest 很简单，只多了一个获取请求行的方法：

```
public interface HttpRequest extends HttpMessage {

    RequestLine getRequestLine();

}
```

而相比之下，HttpResponse 提供的方法就要丰富的多。因为是服务器的响应消息，所以其包含的内容更丰富一些也是很容易理解的：

```
public interface HttpResponse extends HttpMessage {

    StatusLine getStatusLine();

    void setStatusLine(StatusLine statusline);

    void setStatusLine(ProtocolVersion ver, int code);

    void setStatusLine(ProtocolVersion ver, int code, String reason);

    void setStatusCode(int code) throws IllegalStateException;

    void setReasonPhrase(String reason) throws IllegalStateException;

    HttpEntity getEntity();

    void setEntity(HttpEntity entity);

    Locale getLocale();

    void setLocale(Locale loc);

}
```

可以看到，除了与 request 相似的获取状态行之外，还涉及到了状态码、entity、locale 等很多部分的内容。

至此，我们对 request 和 response 的消息结构以及其对应的接口已经有了一个大概的了解，所以可以尝试探究一下从发送 request 到接收到 response 这一个完整的交互过程，来深究一些细节内容。

首先需要说明的是，在使用 http 协议的时候，必定是一端担任客户端，另一端担任服务器端。虽然两台计算机作为客户端和服务器端的角色可以互换，但就一次通信过程而言，客户端和服务器的角色是确定的，http 协议的通信需要对其明确区分。协议规定，请求从客户端发出，最后服务器端响应该请求并返回。换句话说，通信请求是由客户端发起，服务器端在没有接收到请求之前不会发送响应。

客户端需要创建一个请求消息，举一个简单的代码片段：

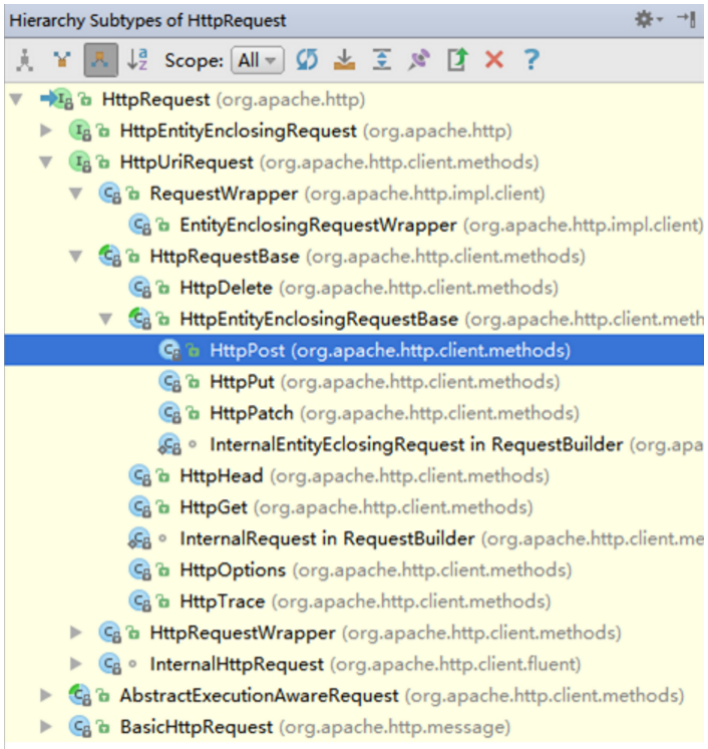
```
public void httpRequestTest() {
    HttpRequest httpRequest = new BasicHttpRequest("GET", "www.baidu.com", HttpVersion.HTTP_1_1);
}
```

我们创建了一个简单的起始行来发送请求，起始行开头的 GET 表示请求访问服务器的类型，随后的字符串 www.baidu.com 指明了请求访问的资源对象，这个字符串也叫作请求 URI（Uniform Resource Identifier，统一资源标识符），最后的 HTTP 1.1 就是我们之前所提到的

http 的版本号，用于告诉服务器客户端所使用的 HTTP 协议版本。所以这段请求内容的意思就是，请求访问 http 服务器 www.baidu.com 的页面资源。

方法	说明	支持的 HTTP 协议版本
GET	获取资源	1.0、1.1
POST	传输实体主体	1.0、1.1
PUT	传输文件	1.0、1.1
HEAD	获得报文首部	1.0、1.1
DELETE	删除文件	1.0、1.1
OPTIONS	询问支持的方法	1.1
TRACE	追踪路径	1.1
CONNECT	要求用隧道协议连接代理	1.1
LINK	建立和资源之间的联系	1.0
UNLINE	断开连接关系	1.0

这里的 GET 就是之前所提到的请求行格式里面的 method，其实这个 method 是 form 元素的属性。在 HTML 中，form 元素内的表单数据可以被提交到服务器，而所谓的提交包括了提交的地址、提交的内容与格式、希望服务器处理提交内容的方式。method 就是用来设置"希望服务器处理提交内容的方式"，包括 form 提交的内容放在什么地方，服务器怎么解析提交的内容等等。GET 正是 HTTP standard specification 中所规定的 method 之一，它的意思是请求由 `Request-URI` 所标识的信息。这个标准中规定的 method 还包括 POST、PUT、DELETE 等。事实上，最常用的只有 GET/POST 两种方式，所有的请求默认就是 GET。



上图是 HttpCore 中提供的常用的一些 HttpRequest 的实现类，可以看出与 HTTP standard specification 中规定的 method 一致的,从图中我们可以看到 HttpDelete、HttpPost、HttpPut、HttpPatch、HttpHead、HttpGet、HttpOptions、HttpTrace 等。当我们需要创建一个请求消息时，其实可以直接选择相应的类使用，非常方便。

由于 POST 需要将发送的内容放到 HTTP message body, 也就是 entity 中, 所以 HttpPost 类中还提供了 entity 的 GETTER/SETTER。而 GET 是不需要 HTTP message body 的，也就是没有 entity。

在创建好了消息之后，就需要将它发送给目标服务器。HTTP 协议使用 URI 定位互联网上的资源。URI 包含的域名通过 DNS 服务映射到 IP 地址，请求便会被发送到这个 IP 地址的服务器。在 URI 的定位帮助下，我们可以访问互联网上任意的开放资源。

服务器收到了请求消息之后，就会按照请求进行相应的处理，并生成一个对应的相应消息返回给客户端。一个简单的响应消息的代码片段:

```
public void httpResponseTest() {
    HttpResponse httpResponse = new BasicHttpResponse(HttpVersion.HTTP_1_1, HttpStatus.SC_OK, "OK");
    System.out.println(httpResponse.getStatusLine());
    //HTTP/1.1 200 OK
    System.out.println(httpResponse);
    //HTTP/1.1 200 OK []
}
```

我们创建了一个简单的状态行来进行相应。第一项信息仍然是 http 的版本，此外它包含了 Status-Code 和 Reason-Phrase。Status-Code 是一个 3 个数字的整数, 负责表示客户端 HTTP 请求的返回结果、标记服务器端的处理是否正常、通知出现的错误等工作。借助状态码，用户可以知道服务器端是正常处理了请求，还是出现了错误。Reason-Phrase 原因短语则是对 Status-Code 简短的文本描述。代码中所示的状态码 200 和原因短语 OK 就表示请求正常处理完毕，返回了正确的结果。

Status-Code 的第一个数字定义了响应的类型:

	类别	原因短语
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

有趣的是，状态码并没有具体的硬性规定，也就是说，只要符合上述类别的定义，服务器端甚至自行创建一些状态码，而这都是符合协议规定的。只不过我们常用的一些状态码已经约

定俗成了，所以在大部分访问请求中收到的状态码是有明确的含义的。下面是几种比较常用的状态码：

204 No Content：代表服务器接收的请求已成功处理，但在返回的响应消息中不含主体部分。浏览器收到返回 204 响应的时候，显示的页面不会发生更新。

301 Moved Permanently：该状态码表示请求的资源已被分配了新的 URI，服务器希望以后使用资源现在所指的 URI 来进行访问（也可能是你输入的地址出现了微小的错误，然后被机智的服务器发现了）。

404 Not Found：这几乎是我们最熟悉的一个状态码了。该状态码表明服务器上无法找到请求的资源。除此之外，也可以在服务器端拒绝请求且不想说明理由时使用（服务器拒绝了你的请求并向你扔了一个 404）。

503 Service Unavailable：表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求（忙着呢别来烦我）。

神奇的一点是，在我查阅资料的时候得知，不少返回的状态码响应都是错误的（迷惑），但是用户可能察觉不到这点。比如 Web 应用程序内部发生错误，状态码依然返回 200 OK，这种 情况也经常遇到。所以我们也许并不能百分之百信任状态码，有可能是服务器抽风了。

然后便是 header，头域部分。这部分允许服务器传递一些不能放在状态行(Status-Line)的关于响应的额外信息（状态行的信息量实在是有限）。这些头域给出了关于服务器和更多关于进由 Request-URI 标识的资源的一些咨询。

最后是响应消息的 body 了，包含请求的资源信息等，会随着请求的不同而发生一些变化。

由于之前请求消息所发送的是一个 GET 请求，也就是客户端要请求访问已被 URI 识别的资源。指定的资源经服务器端解析后，返回响应内容。也就是说，如果请求的资源是文本，那就原样返回；如果是像 CGI（Common Gateway Interface，通用网关接口）那样的程序，则返回经过执行后的输出结果。下面是一个 GET 的请求-返回的实例：

请求	GET /index.html HTTP/1.1 Host: www.hackr.jp If-Modified-Since: Thu, 12 Jul 2012 07:30:00 GMT
响应	仅返回2012年7 月12日7 点30分以后更新过的index.html页面资源。如果未有内容更新，则以状态码304 Not Modified作为响应返回

之前提到过最常用的是 GET/POST 请求，因此再提一下 POST。POST 方法用来传输实体的主体。虽然用 GET 方法也可以传输实体的主体，但一般是用 POST 方法。下面是一个 POST 的请求-返回的实例：

请求	POST /submit.cgi HTTP/1.1 Host: www.hackr.jp Content-Length: 1560（1560字节的数据）
响应	返回 submit.cgi 接收数据的处理结果

在创建完返回消息之后，这条消息便会从服务器出发，一路返回到来源的客户端上，由客户端的浏览器解析之后呈现在我们的面前。至此，一次完整的基于 http 协议的请求和响应过程就已经全部完成了（完结撒花）！

其实，在传输的过程中，除了 http 之外，还需要许多协议的帮助，包括我们提到的 DNS 协议和没提到过的 TCP/IP 协议等等。正是在这些协议的紧密配合之下，我们才有了愉快的网页浏览体验，只是这学期精力有限，只能对 http 协议有一个大概的了解，关于其他协议的内容以及这些协议之间的配合，希望以后有时间的时候能进行更加深入的了解。

关于 http 协议所体现的面向对象思想其实也是非常明显的。这里用一个我查到的反例来进行说明。在很久很久（好几年）之前，Apache 有一个 HTTP 协议的客户端实现 Commons HttpClient。通俗地讲，大家称它为基于 3.X 的或者老代码。这个项目当时的使用也是非常广泛，很多程序员都靠这个吃饭。但老代码在设计上有一些严重的限制。例如，有一个叫做 HttpMethodBase 的类，它同时表示一个请求和一个响应，它同时实现了两者的逻辑！这种将不同的功能放在一起的设计，让它很难维护，也很难进行扩展（也很难读）。因此，我们启动了它的后继项目，也就是今天我们所看到的 HttpComponents。基于老代码中经验和教训，我们使用了新的方法来实现 HTTP 协议。上面的问题得到了很好的解决，因为我们已经能非常明确的感受到，新项目关于模块化分离做的比老代码不知道高到哪里去了（起码不会吧请求和响应放到一起哈）。这便是面向对象思想一个非常好的体现。

当然，HTTP 协议本身仍然是有一些不足之处的。比如通信使用明文，没有身份验证等等，这都是对信息安全的考验，也就是说，我们传输的信息有被窃取，伪造的风险。这也是我们在一开始所说的 https 开始逐渐崭露头角的原因，正是为了数据安全的保障。任何项目都不会是完美的，漏洞是一定存在的，只有在不断的更新迭代中逐渐修复这些漏洞，才能不被飞速发展的互联网所抛弃。

一学期的面向对象课程很快就结束了，对老师和助教都印象深刻。由于这学期课程和实

验实在是太紧迫，对于这么一个著名的项目只是了解了一些皮毛，希望在以后能进行更加深入的挖掘。