http协议

HTTP协议的主要特点可概括如下

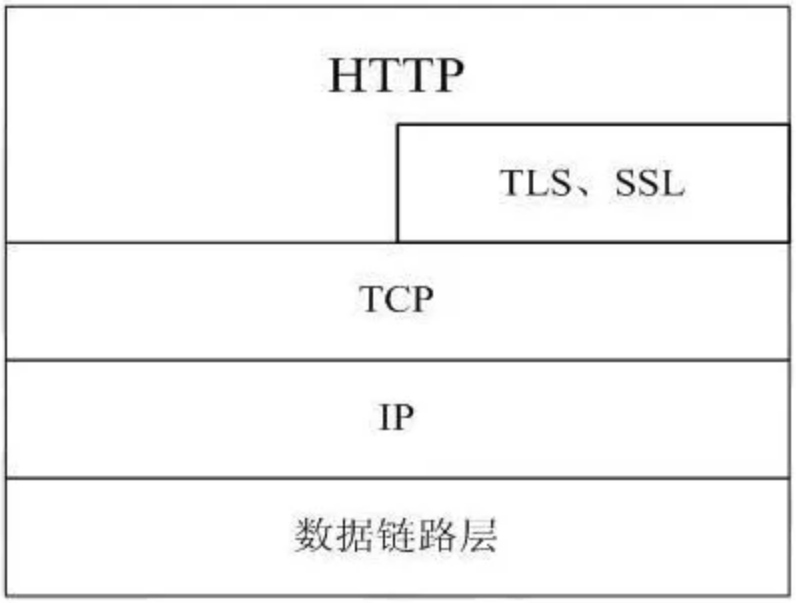
1.支持客户/服务器模式。
2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

3.灵活：这里主要指的是客户端可以通过http协议传输任意类型的数据。比如传输.jpg文件、.ppt文件等等，只需要设定content-type就可以进行传输。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以
节省传输时间。

5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

HTTP协议通常承载于TCP协议之上，有时也承载了TLS或SSL协议层之上，这个时候，就成了我们常说的HTTPS



> 浏览网页是HTTP的主要应用，但是这并不代表HTTP就只能应用于网页的浏览。HTTP是一种协议，只要通信的双方都遵守这个协议，HTTP就能有用武之地。比如咱们常用的QQ，迅雷这些软件，都会使用HTTP协议(还包括其他的协议)。

客户端发送一个HTTP请求到服务器的请求消息包括以下格式：**请求行（request line）**、**请求头部（header）**、**空行**和**请求正文**四个部分组成

Get请求例子，使用Charles抓取的request：

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
Host    img.mukewang.com
User-Agent    Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari/537.36
Accept    image/webp,image/*,*/*;q=0.8
Referer    http://www.imooc.com/
Accept-Encoding    gzip, deflate, sdch
Accept-Language    zh-CN,zh;q=0.8
```

第一部分：请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.

GET说明请求类型为GET,[/562f25980001b1b106000338.jpg]为要访问的资源，该行的最后一部分说明使用的是HTTP1.1版本。

第二部分：请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息

从第二行起为请求头部，HOST将指出请求的目的地.User-Agent,服务器端和客户端脚本都能访问它,它是浏览器类型检测逻辑的重要基础.该信息由你的浏览器来定义,并且在每个请求中自动发送等等

第三部分：空行，请求头部后面的空行是必须的

即使第四部分的请求数据为空，也必须有空行。

第四部分：请求数据也叫主体，可以添加任意的其他数据。

这个例子的请求数据为空。

POST请求例子，使用Charles抓取的request：

```
POST / HTTP1.1

Host:www.wrox.com
User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
Content-Type:application/x-www-form-urlencoded
Content-Length:40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

第一部分：请求行，第一行明了是post请求，以及http1.1版本。

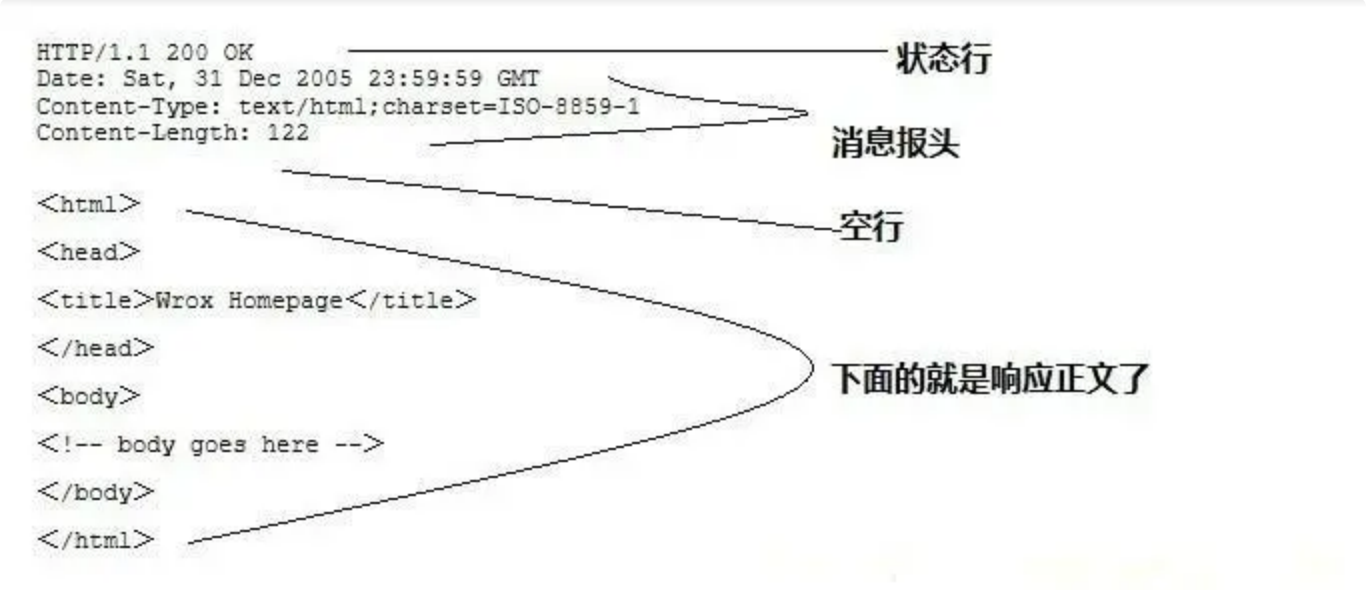
第二部分：请求头部，第二行至第六行。

第三部分：空行，第七行的空行。

第四部分：请求数据，第八行。

HTTP之响应消息Response

HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。



http响应消息格式.jpg

例子

```
1  HTTP/1.1 200 OK
2  Date: Fri, 22 May 2009 06:07:21 GMT
3  Content-Type: text/html; charset=UTF-8
4  <html>
5        <head></head>
6        <body>
7              <!--body goes here-->
8        </body>
9  </html>
```

第一部分：状态行，由HTTP协议版本号， 状态码， 状态消息 三部分组成。

第一行为状态行，（HTTP/1.1）表明HTTP版本为1.1版本，状态码为200，状态消息为（ok）

第二部分：消息报头，用来说明客户端要使用的一些附加信息

第二行和第三行为消息报头，
Date:生成响应的日期和时间；Content-Type:指定了MIME类型的HTML(text/html),编码类型是UTF-8

第三部分：空行，消息报头后面的空行是必须的

第四部分：响应正文，服务器返回给客户端的文本信息。

空行后面的html部分为响应正文。

HTTP请求方法

HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

方法	描述
GET	向Web服务器请求一个文件
POST	向Web服务器发送数据让Web服务器进行处理
PUT	向Web服务器发送数据并存储在Web服务器内部
HEAD	检查一个对象是否存在
DELETE	从Web服务器上删除一个文件
CONNECT	对通道提供支持
TRACE	跟踪到服务器的路径
OPTIONS	查询Web服务器的性能

GET和POST两种基本请求方法的区别

- GET在浏览器回退时是无害的，而POST会再次提交请求。
- GET产生的URL地址可以被Bookmark，而POST不可以。
- GET请求会被浏览器主动cache，而POST不会，除非手动设置。
- GET请求只能进行url编码，而POST支持多种编码方式。
- GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。
- GET请求在URL中传送的参数是有长度限制的，而POST么有。
- 对参数的数据类型，GET只接受ASCII字符，而POST没有限制。
- GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。
- GET参数通过URL传递，POST放在Request body中。

ASCII

在 HTTP 协议中，URL 是 HTTP 的一个首部。既然作为一个首部，那么根据约定，一定是 ASCII 字符的（毕竟计算机是美帝发明的嘛），而 GET 参数取的正是 URL 中的 QUERY STRING。最初 HTTP 也和计算机一样只是应用的很少，美帝也没想到会覆盖全球，于是他们就很 naive 地使用非 ASCII 的头部 URL 来传输 GET。后来有了 UTF-8，浏览器开发公司发现，编码都是十六进制的嘛。于是开始使用 % 加编码的一个字节来组成新编码，这就是 URL code 的由来，其实本质上传输的还是 ASCII，只不过对它硬转码了。

HTTP 是直接使用 MIME 的，它是为了收发非 ASCII 邮件而存在的一种扩展。最初电子邮件也是只支持 ASCII，可是由于邮件越来越复杂，美帝才对协议进行扩展，让它能够传输小电影、小图片或者其它的文件，然后就有了 MIME，它有一个 MIME type，来说明具体是什么类型的文件。后来，其他国家的人一想：诶，卧槽，我直接用 MIME 来传输我们国家的字符不就完了么。然后 MIME 就能传非 ASCII 字符了。

我们说的 POST，它其实就是 HTTP 的实体，通过 MIME，也就可以传输非 ASCII 字符了。

GIT提交

GET方式提交，浏览器会对URL进行URL encode，然后发送给服务器

1.对于中文IE,如果在高级选项中选中总以UTF-8发送(默认方式)，则PathInfo是URL Encode是按照UTF-8编码,QueryString是按照GBK编码。

http://localhost:8080/example/中 国 name=中国

实际上提交是：GET /example/%E4%B8%AD%E5%9B%BD

name=%D6%D0%B9%FA

2.对于中文IE,如果在高级选项中取消总以UTF-8发送，则PathInfo和QueryString是URL encode按照GBK编码。

实际上提交是：GET /example/%D6%D0%B9%FA name=%D6%D0%B9%FA

3.对于中文firefox，则pathInfo和queryString都是URL encode按照GBK编码。

实际上提交是：GET /example/%D6%D0%B9%FA name=%D6%D0%B9%FA

很显然，不同的浏览器以及同一浏览器的不同设置，会影响最终URL中PathInfo的编码。

对于中文的IE和FIREFOX都是采用GBK编码 QueryString。

小结，解决方案：

4.**URL中如果含有中文等非ASCII字符，则浏览器会对它们进行URLEncode。为了避免浏览器采用了我们不希望的编码，

所以最好不要在URL中直接使用非ASCII字符，而采用URL Encode编码过的字符串%.**

比如：URL：http://localhost:8080 /example/中国 name=中国

建议：URL：http://localhost:8080/example/%D6%D0 %B9%FA

name=%D6%D0%B9%FA

我们建议URL中PathInfo和QueryString采用相同的编码，这样对服务器端处理的时候会更加简单。

POST方式提交

对于POST方式，表单中的参数值对是通过request body发送给服务器，**此时浏览器会根据网页的ContentType(“text/html; charset=GBK”)
中指定的编码进行对表单中的数据进行编码**，然后发给服务器。
在服务器端的程序中我们**可以通过 Request.setCharacterEncoding() 设置编码**，然后通过request.getParameter获得正确的数据。
解决方案：
从最简单，所需代价最小来看，我们**对URL以及网页中的编码使用统一的编码**对我们来说是比较合适的。
如果不使用统一编码的话，我们就需要在程序中做一些编码转换的事情。这也是我们为什么看到有网络上大量的资料介绍如何对乱码进行处理，
其中很多解决方案都只是一时的权宜之计，没有从根本上解决问题。

GET和POST总结

(1)**URL中的PathInfo和QueryString字符串的编码和解码是由浏览器和应用服务器的配置决定的**，
我们的程序不能设置，不要期望用request.setCharacterEncoding()方法能设置URL中参数值解码时的字符集。
所以我们建议URL中不要使用中文等非ASCII字符，如果含有非ASCII字符的话要使用URLEncode编码一下，比如：
http://localhost:8080 /example1/example/中国
正确的写法：
http://localhost:8080/example1/example/%E4%B8%AD%E5%9B%BD
并且我们建议URL中不要在PathInfo和QueryString同时使用非ASCII字符，比如
http://localhost:8080 /example1/example/中国 name=中国
原因很简单：不同浏览器对URL中PathInfo和QueryString编码时采 用的字符集不同，但应用服务器对URL通常会
采用相同的字符集来解码。
(2)我们建议URL中的**URL Encode编码的字符集和网页的contentType的字符集采用相同的字符集**，这样程序的实现就很简单，
不用做复杂的编码转换。 JS中对中文(UTF-8格式保存的)进行编码(unicode编码)的函数常用有：
encodeURI(str); encodeURIComponent(str);

HTTP状态码

- 1xx：指示信息--表示请求已接收，继续处理。

- 2xx：成功--表示请求已被成功接收、理解、接受。
- 3xx：重定向--要完成请求必须进行更进一步的操作。
- 4xx：客户端错误--请求有语法错误或请求无法实现。
- 5xx：服务器端错误--服务器未能实现合法的请求。

常见的HTTP相应状态码

200：请求被正常处理
204：请求被受理但没有资源可以返回
206：客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源(在播放视屏和音频地址的时候会出现)。
301：永久性重定向（所请求的页面已经转移新的url）
302：临时重定向
303：与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上
304：客户端有缓存了文档并发出了一个条件性的请求，服务器告诉客户，原来缓冲的文档还阔以继续使用
307：临时重定向，与302类似，只是强制要求使用POST方法
400：请求报文语法有误(客户端语法错误)，服务器无法识别
401：请求需要认证
403：请求的对应资源禁止被访问
404：服务器无法找到对应资源
500：服务器内部错误
503：服务器正忙(服务器宕机或者过载)

http协议详解及htt面试题目，常见的http状态码:https://blog.csdn.net/xiaoninvhuang/article/details/70257189
HTTP协议：持久连接、非持久连接https://blog.csdn.net/neninee/article/details/79634187

http管线化

HTTP协议类 管线化

- 管线化机制通过持久连接完成，仅 HTTP/1.1 支持此技术
- 只有 GET 和 HEAD 请求可以进行管线化，而 POST 则有所限制
- 初次创建连接时不应启动管线机制，因为对方（服务器）不一定支持 HTTP/1.1 版本的协议
- 管线化不会影响响应到来的顺序，如上面的例子所示，响应返回的顺序并未改变
- HTTP /1.1 要求服务器端支持管线化，但并不要求服务器端也对响应进行管线化处理，只是要求对于管线化的请求不失败即可
- 由于上面提到的服务器端问题，开启管线化很可能并不会带来大幅度的性能提升，而且很多服务器端和代理程序对管线化的支持并不好，因此现代浏览器如 Chrome 和 Firefox 默认并未开启管线化支持