## **简介**

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。。

HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

## **HTTP 工作原理**

HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。

Web服务器有：Apache服务器，IIS服务器（Internet Information Services）等。

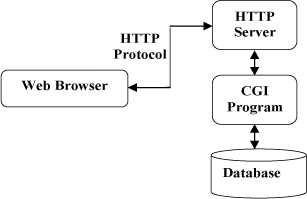
Web服务器根据接收到的请求后，向客户端发送响应信息。

HTTP默认端口号为80，但是你也可以改为8080或者其他端口。

****HTTP三点注意事项：****

* HTTP是无连接：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。
* HTTP是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过HTTP发送。客户端以及服务器指定使用适合的MIME-type内容类型。
* HTTP是无状态：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。

以下图表展示了HTTP协议通信流程：



HTTP 为一个通信协议。

HTTP 客户端发起请求并创建端口。

HTTP 服务器在端口监听客户端的请求。

HTTP 服务器在收到请求后则返回状态和所请求的内容。

## **消息结构**

HTTP是基于客户端/服务端（C/S）的架构模型，通过一个可靠的链接来交换信息，是一个无状态的请求/响应协议。

一个HTTP"客户端"是一个应用程序（Web浏览器或其他任何客户端），通过连接到服务器达到向服务器发送一个或多个HTTP的请求的目的。

一个HTTP"服务器"同样也是一个应用程序（通常是一个Web服务，如Apache Web服务器或IIS服务器等），通过接收客户端的请求并向客户端发送HTTP响应数据。

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。

一旦建立连接后，数据消息就通过类似Internet邮件所使用的格式[RFC5322]和多用途Internet邮件扩展（MIME）[RFC2045]来传送。

## **客户端请求消息**

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）、空行和请求数据四个部分组成，下图给出了请求报文的一般格式。

****

## **服务器响应消息**

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。



## **实例**

下面实例是一点典型的使用GET来传递数据的实例：

客户端请求：

GET /hello.txt HTTP/1.1 User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3 Host: www.example.com Accept-Language: en, mi

服务端响应:

HTTP/1.1 200 OK Date: Mon, 27 Jul 2009 12:28:53 GMT Server: Apache Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT ETag: "34aa387-d-1568eb00" Accept-Ranges: bytes Content-Length: 51 Vary: Accept-Encoding Content-Type: text/plain

输出结果：

Hello World! My payload includes a trailing CRLF.

实例：

<!DOCTYPE html>

<html>

<head>

<script src="/jquery/jquery-1.11.1.min.js"></script>

<script>

$(document).ready(function(){

$("button").click(function(){

$.get("/jquery/demo\_test.php",function(data,status){

alert("数据：" + data + "\n状态：" + status);

});

});

});

</script>

</head>

<body>

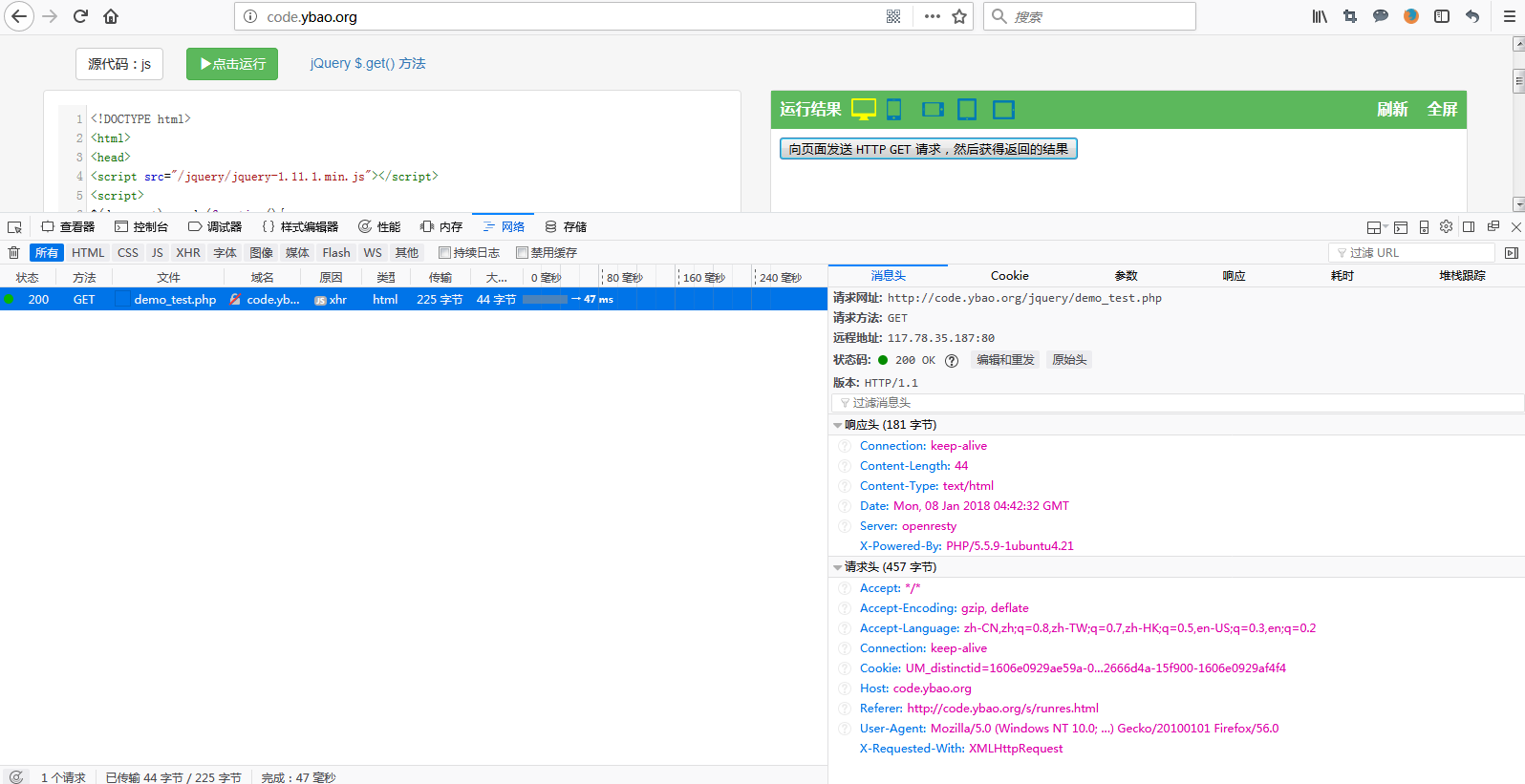
<button>向页面发送 HTTP GET 请求，然后获得返回的结果</button>

</body>

</html>

[亲自试一试](http://www.ybao.org/article-getsc-1955" \t "http://www.ybao.org/book/http/_blank)

使用firefox的Web开发者工具（F12），可查看具体情况：



## **常用方法**

## ****常用方法****

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **是否包含主题** |
| GET | 从服务器获取一份文档 | 否 |
| POST | 向服务器发送需要处理的数据 | 是 |
| PUT | 将请求的主题部分储存在服务器上 | 是 |
| DELETE | 从服务器删除一份文档 | 否 |

****不常用方法****

|  |  |  |
| --- | --- | --- |
| **方法** | **描述** | **是否包含主题** |
| HEAD | 只从服务器获取头文档的首部 | 否 |
| TRACE | 对可能经过代理服务器传送到服务器上的报文进行追踪 | 否 |
| OPTIONS | 决定可以在服务器上执行的方法 | 否 |

## **GET 对比 POST**

****两种最常用的 HTTP 方法是：GET 和 POST。****

## **什么是 HTTP？**

超文本传输协议（HTTP）的设计目的是保证客户机与服务器之间的通信。

HTTP 的工作方式是客户机与服务器之间的请求-应答协议。

web 浏览器可能是客户端，而计算机上的网络应用程序也可能作为服务器端。

举例：客户端（浏览器）向服务器提交 HTTP 请求；服务器向客户端返回响应。响应包含关于请求的状态信息以及可能被请求的内容。

## **两种 HTTP 请求方法：GET 和 POST**

在客户机和服务器之间进行请求-响应时，两种最常被用到的方法是：GET 和 POST。

* 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

## **GET 方法**

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

<!DOCTYPE html>

<html>

<head>

<script src="/jquery/jquery-1.11.1.min.js">

</script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $.get("/jquery/demo\_test\_get.php",

    {

      name:"Donald Duck",

      city:"Duckburg"

    },

    function(data,status){

      alert("数据：" + data + "\n状态：" + status);

    });

  });

});

</script>

</head>

<body>

<button>向页面发送 HTTP GET 请求，并获得返回的结果</button>

</body>

</html>

## [尝试一下 »](http://www.ybao.org/article-getsc-1960" \t "http://www.ybao.org/book/http/_blank)

服务端文件demo\_test\_get.php：

<?php

$fname = $\_GET["name"];

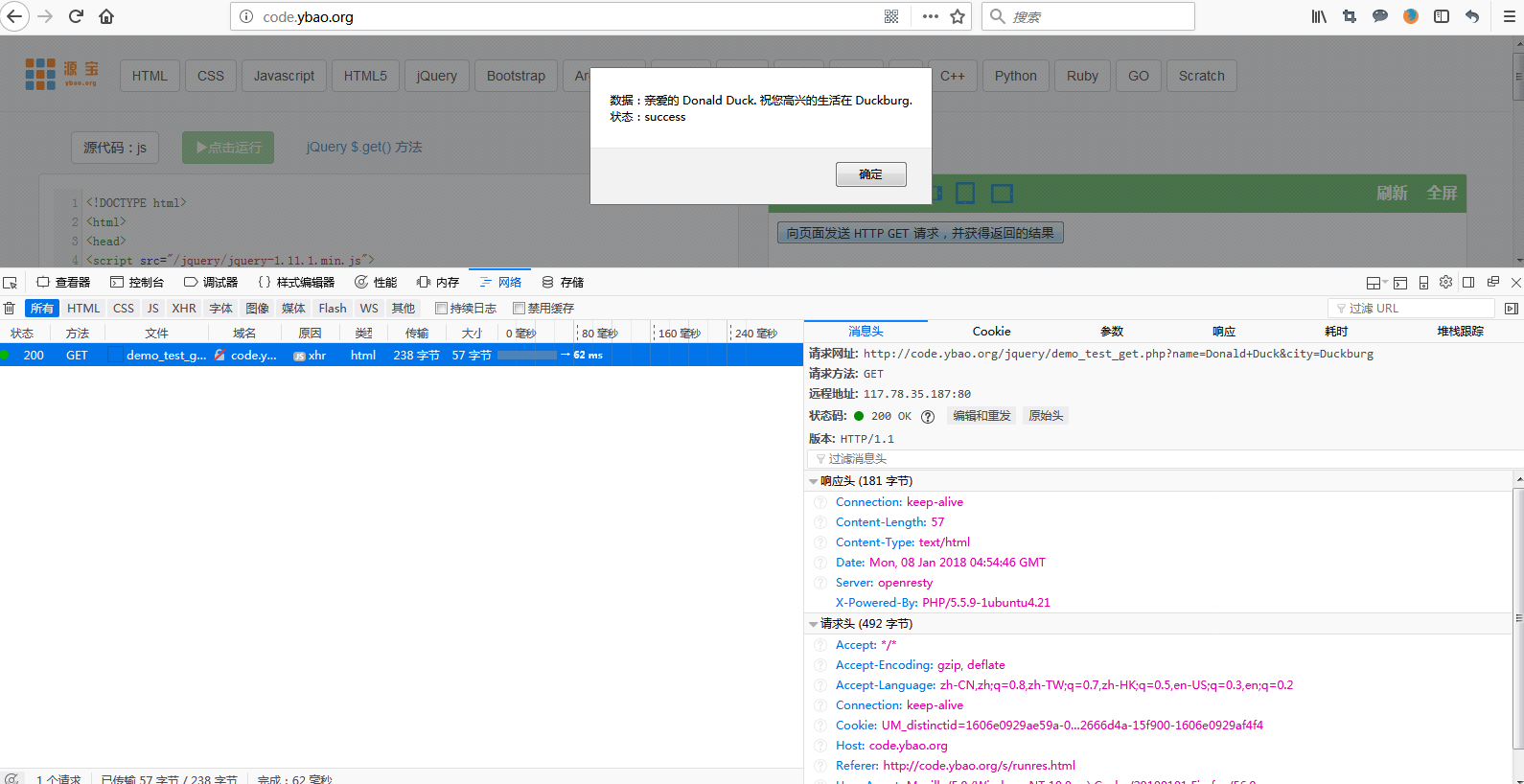
$city = $\_GET["city"];

echo "亲爱的 " . $fname . ". ";

echo "祝您高兴的生活在 " . $city . ".";

?>

使用Firefox Web开发者工具（F12）



有关 GET 请求的其他一些注释：

* GET 请求可被缓存
* GET 请求保留在浏览器历史记录中
* GET 请求可被收藏为书签
* GET 请求不应在处理敏感数据时使用
* GET 请求有长度限制
* GET 请求只应当用于取回数据

## **POST 方法**

请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

<!DOCTYPE html>

<html>

<head>

<script src="/jquery/jquery-1.11.1.min.js">

</script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $.post("/jquery/demo\_test\_post.php",

    {

      name:"Donald Duck",

      city:"Duckburg"

    },

    function(data,status){

      alert("数据：" + data + "\n状态：" + status);

    });

  });

});

</script>

</head>

<body>

<button>向页面发送 HTTP POST 请求，并获得返回的结果</button>

</body>

</html>

## [尝试一下 »](http://www.ybao.org/article-getsc-1956" \t "http://www.ybao.org/book/http/_blank)

服务端文件demo\_test\_post.php：

<?php

$fname = $\_POST["name"];

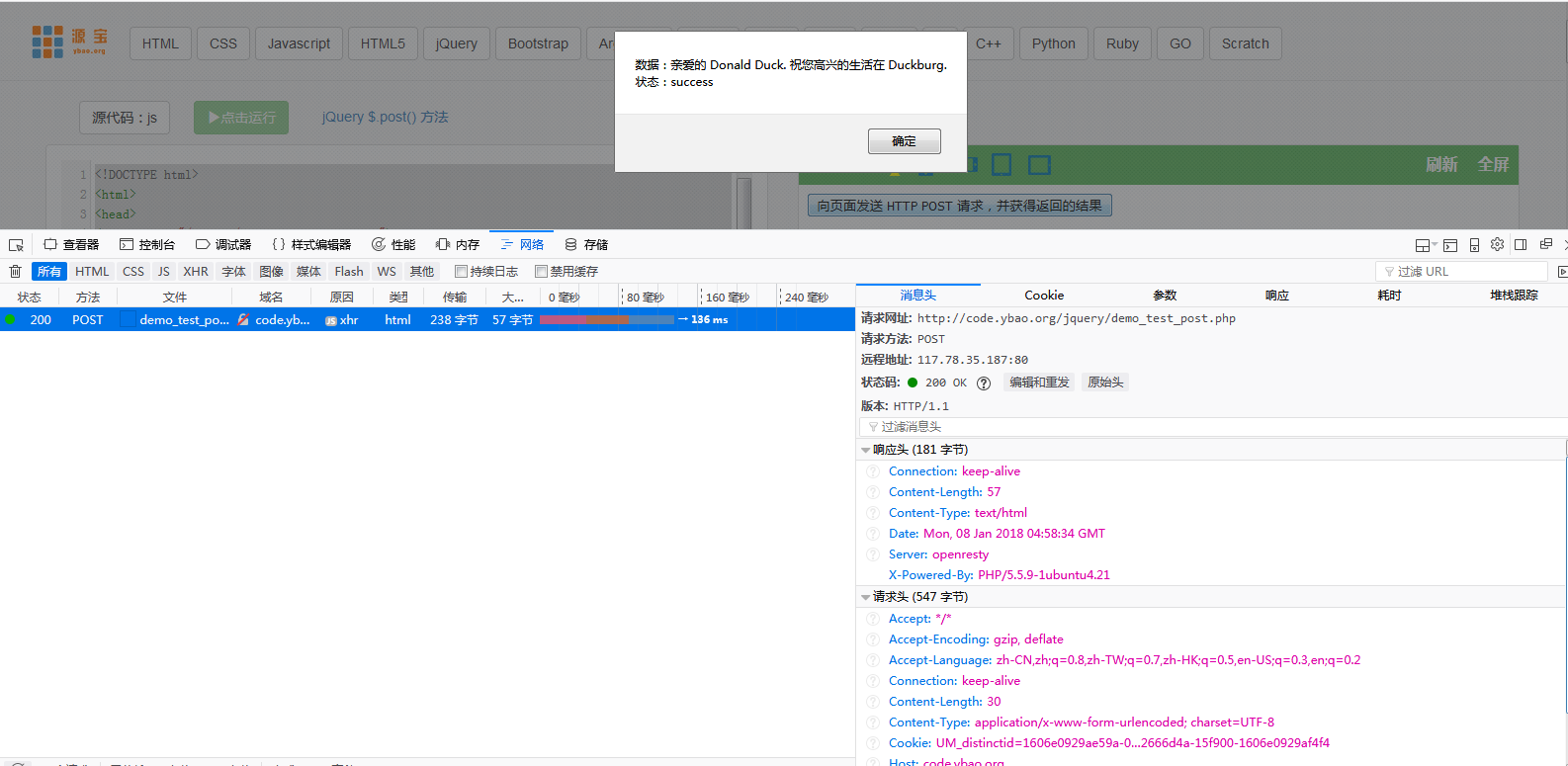
$city = $\_POST["city"];

echo "亲爱的 " . $fname . ". ";

echo "祝您高兴的生活在 " . $city . ".";

?>

使用Firefox Web开发者工具（F12）



有关 POST 请求的其他一些注释：

* POST 请求不会被缓存
* POST 请求不会保留在浏览器历史记录中
* POST 不能被收藏为书签
* POST 请求对数据长度没有要求

## **比较 GET 与 POST**

下面的表格比较了两种 HTTP 方法：GET 和 POST。

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| 后退按钮/刷新 | 无害 | 数据会被重新提交（浏览器应该告知用户数据会被重新提交）。 |
| 书签 | 可收藏为书签 | 不可收藏为书签 |
| 缓存 | 能被缓存 | 不能缓存 |
| 编码类型 | application/x-www-form-urlencoded | application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。 |
| 历史 | 参数保留在浏览器历史中。 | 参数不会保存在浏览器历史中。 |
| 对数据长度的限制 | 是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。 | 无限制。 |
| 对数据类型的限制 | 只允许 ASCII 字符。 | 没有限制。也允许二进制数据。 |
| 安全性 | 与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。  在发送密码或其他敏感信息时绝不要使用 GET ！ | POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。 |
| 可见性 | 数据在 URL 中对所有人都是可见的。 | 数据不会显示在 URL 中。 |

## **其他 HTTP 请求方法**

下面的表格列出了其他一些 HTTP 请求方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| HEAD | 与 GET 相同，但只返回 HTTP 报头，不返回文档主体。 |
| PUT | 上传指定的 URI 表示。 |
| DELETE | 删除指定资源。 |
| OPTIONS | 返回服务器支持的 HTTP 方法。 |
| CONNECT | 把请求连接转换到透明的 TCP/IP 通道。 |

## **响应头信息**

HTTP请求头提供了关于请求，响应或者其他的发送实体的信息。

在本章节中我们将具体来介绍HTTP响应头信息。

|  |  |
| --- | --- |
| **应答头** | **说明** |
| Allow | 服务器支持哪些请求方法（如GET、POST等）。 |
| Content-Encoding | 文档的编码（Encode）方法。只有在解码之后才可以得到Content-Type头指定的内容类型。利用gzip压缩文档能够显著地减少HTML文档的下载时间。Java的GZIPOutputStream可以很方便地进行gzip压缩，但只有Unix上的Netscape和Windows上的IE 4、IE 5才支持它。因此，Servlet应该通过查看Accept-Encoding头（即request.getHeader("Accept-Encoding")）检查浏览器是否支持gzip，为支持gzip的浏览器返回经gzip压缩的HTML页面，为其他浏览器返回普通页面。 |
| Content-Length | 表示内容长度。只有当浏览器使用持久HTTP连接时才需要这个数据。如果你想要利用持久连接的优势，可以把输出文档写入 ByteArrayOutputStream，完成后查看其大小，然后把该值放入Content-Length头，最后通过byteArrayStream.writeTo(response.getOutputStream()发送内容。 |
| Content-Type | 表示后面的文档属于什么MIME类型。Servlet默认为text/plain，但通常需要显式地指定为text/html。由于经常要设置Content-Type，因此HttpServletResponse提供了一个专用的方法setContentType。 |
| Date | 当前的GMT时间。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦。 |
| Expires | 应该在什么时候认为文档已经过期，从而不再缓存它？ |
| Last-Modified | 文档的最后改动时间。客户可以通过If-Modified-Since请求头提供一个日期，该请求将被视为一个条件GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个304（Not Modified）状态。Last-Modified也可用setDateHeader方法来设置。 |
| Location | 表示客户应当到哪里去提取文档。Location通常不是直接设置的，而是通过HttpServletResponse的sendRedirect方法，该方法同时设置状态代码为302。 |
| Refresh | 表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。  注意这种功能通常是通过设置HTML页面HEAD区的＜META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://host/path"＞实现，这是因为，自动刷新或重定向对于那些不能使用CGI或Servlet的HTML编写者十分重要。但是，对于Servlet来说，直接设置Refresh头更加方便。   注意Refresh的意义是"N秒之后刷新本页面或访问指定页面"，而不是"每隔N秒刷新本页面或访问指定页面"。因此，连续刷新要求每次都发送一个Refresh头，而发送204状态代码则可以阻止浏览器继续刷新，不管是使用Refresh头还是＜META HTTP-EQUIV="Refresh" ...＞。   注意Refresh头不属于HTTP 1.1正式规范的一部分，而是一个扩展，但Netscape和IE都支持它。 |
| Server | 服务器名字。Servlet一般不设置这个值，而是由Web服务器自己设置。 |
| Set-Cookie | 设置和页面关联的Cookie。Servlet不应使用response.setHeader("Set-Cookie", ...)，而是应使用HttpServletResponse提供的专用方法addCookie。参见下文有关Cookie设置的讨论。 |
| WWW-Authenticate | 客户应该在Authorization头中提供什么类型的授权信息？在包含401（Unauthorized）状态行的应答中这个头是必需的。例如，response.setHeader("WWW-Authenticate", "BASIC realm=＼"executives＼"")。  注意Servlet一般不进行这方面的处理，而是让Web服务器的专门机制来控制受密码保护页面的访问（例如.htaccess）。 |

读取响应头部信息：

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>读取相应的响应头部</title>

<script type="text/javascript">

var xmlHttp;

var requestType = "";

function createXMLHttpRequest() {

if (window.ActiveXObject) {

xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

}

else if (window.XMLHttpRequest) {

xmlHttp = new XMLHttpRequest();

}

}

function doHeadRequest(request, url) {

requestType = request;

createXMLHttpRequest();

xmlHttp.onreadystatechange = handleStateChange;

xmlHttp.open("HEAD", url, true);

xmlHttp.send(null);

}

function handleStateChange() {

if(xmlHttp.readyState == 4) {

if(requestType == "allResponseHeaders") {

getAllResponseHeaders();

}

else if(requestType == "lastModified") {

getLastModified();

}

else if(requestType == "isResourceAvailable") {

getIsResourceAvailable();

}

}

}

function getAllResponseHeaders() {

alert(xmlHttp.getAllResponseHeaders());

}

function getLastModified() {

alert("最后被修改的时间：" + xmlHttp.getResponseHeader("Last-Modified"));

}

function getIsResourceAvailable() {

if(xmlHttp.status == 200) {

alert("成功响应");

}

else if(xmlHttp.status == 404) {

alert("资源页无效");

}

else {

alert("异常反应状态：" + xmlHttp.status);

}

}

</script>

</head>

<body>

<h1>读取相应的响应头部</h1>

<a href="j a v a s c r i p t :doHeadRequest('allResponseHeaders', '/demo/src/readingResponseHeaders.xml');">读所有响应头部</a>

<br/>

<a href="j a v a s c r i p t :doHeadRequest('lastModified', '/demo/src/readingResponseHeaders.xml');">获得最后修改的时间</a>

<br/>

<a href="j a v a s c r i p t :doHeadRequest('isResourceAvailable', '/demo/src/readingResponseHeaders.xml');">读取有效资源</a>

<br/>

<a href="j a v a s c r i p t :doHeadRequest('isResourceAvailable', '/demo/src/not-available.xml');">读取无效资源</a>

</body>

</html>

## **URL构成**

http://www.github.com:8080/index.html?user=li-xinyang&lang=zh-CN#home

| | | | | |

protocol | | | | |

hostname port | | |

\ / pathname search hash

host

****可选部分包括****：

* port
* pathname
* search
* hash

NOTE：上面提供的 URL 地址仅为参考所用。

## **HTTP版本**

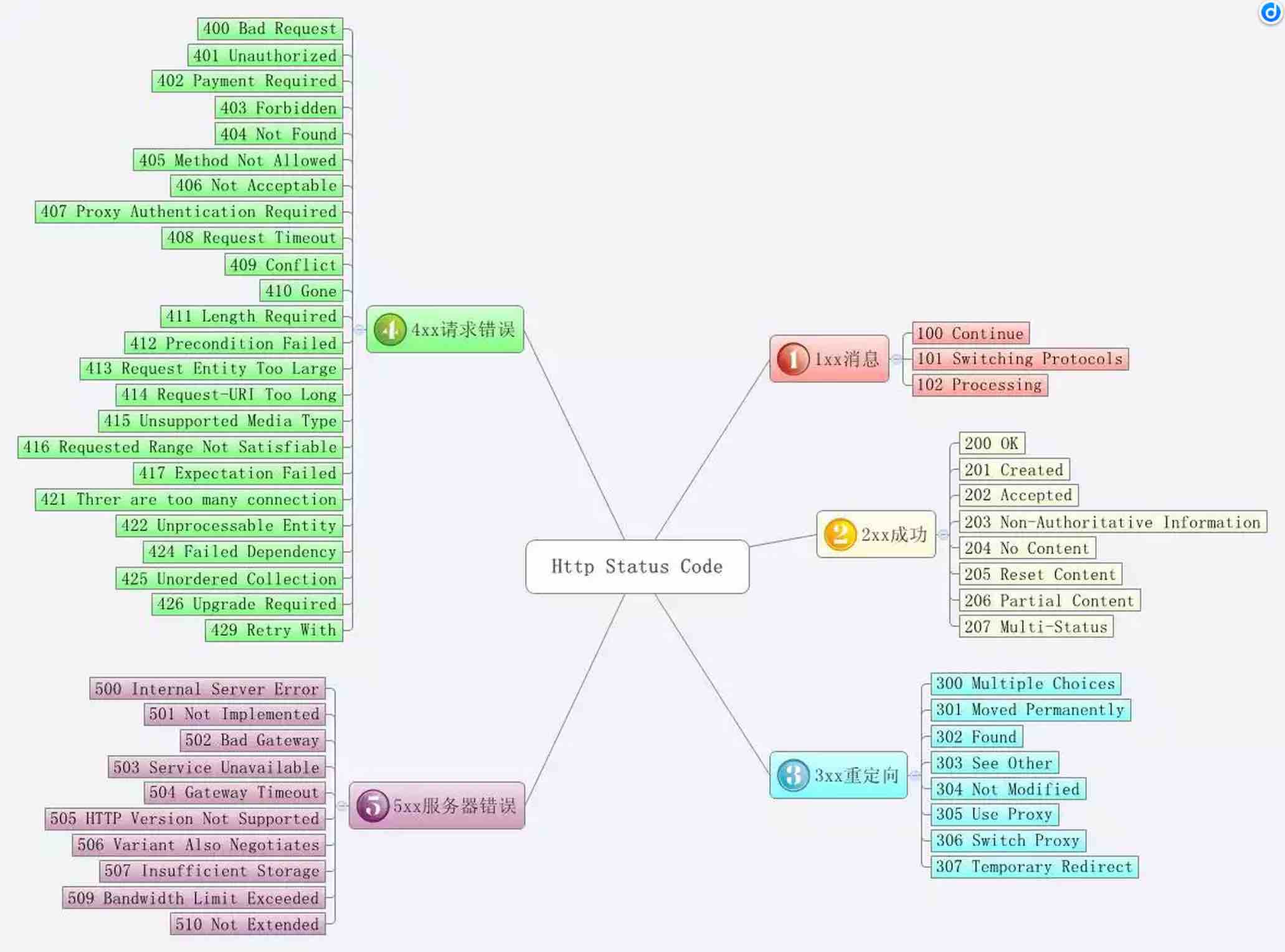
#### HTTP 版本

* HTTP/0.9 1991年 HTTP 原型，存在设计缺陷
* HTTP/1.0 第一个广泛应用版本
* HTTP/1.0+ 添加持久的 keep-alive 链接，虚拟主机支持，代理连接支持，成为非官方的事实版本
* HTTP/1.1 校正结构性缺陷，明确语义，引入重要的新能优化措施，删除不好的特性（****当前使用版本****）

## **状态码**

#### 常见 HTTP 状态码

|  |  |  |
| --- | --- | --- |
| **状态码** | **描述** | **代码描述** |
| 200 | 请求成功，一般用于 GET 和 POST 方法 | OK |
| 301 | 资源移动。所请求资源自动到新的 URL，浏览器自动跳转至新的 URL | Moved Permanently |
| 304 | 未修改。所请求资源未修改，浏览器读取缓存数据 | Not Modified |
| 400 | 请求语法错误，服务器无法解析 | Bad Request |
| 404 | 未找到资源，可以设置个性“404页面” | Not Found |
| 500 | 服务器内部错误 | Internal Server Error |



## **HTTP状态码分类**

HTTP状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。HTTP状态码共分为5种类型：

HTTP状态码分类

|  |  |
| --- | --- |
| **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 |

HTTP状态码列表:

HTTP状态码列表

|  |  |  |
| --- | --- | --- |
| **状态码** | **状态码英文名称** | **中文描述** |
| 100 | Continue | 继续。[客户端](http://www.dreamdu.com/webbuild/client_vs_server/)应继续其请求 |
| 101 | Switching Protocols | 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议 |
|  | | |
| 200 | OK | 请求成功。一般用于GET与POST请求 |
| 201 | Created | 已创建。成功请求并创建了新的资源 |
| 202 | Accepted | 已接受。已经接受请求，但未处理完成 |
| 203 | Non-Authoritative Information | 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本 |
| 204 | No Content | 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档 |
| 205 | Reset Content | 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域 |
| 206 | Partial Content | 部分内容。服务器成功处理了部分GET请求 |
|  | | |
| 300 | Multiple Choices | 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择 |
| 301 | Moved Permanently | 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替 |
| 302 | Found | 临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI |
| 303 | See Other | 查看其它地址。与301类似。使用GET和POST请求查看 |
| 304 | Not Modified | 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源 |
| 305 | Use Proxy | 使用代理。所请求的资源必须通过代理访问 |
| 306 | Unused | 已经被废弃的HTTP状态码 |
| 307 | Temporary Redirect | 临时重定向。与302类似。使用GET请求重定向 |
|  | | |
| 400 | Bad Request | 客户端请求的语法错误，服务器无法理解 |
| 401 | Unauthorized | 请求要求用户的身份认证 |
| 402 | Payment Required | 保留，将来使用 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但是拒绝执行此请求 |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面 |
| 405 | Method Not Allowed | 客户端请求中的方法被禁止 |
| 406 | Not Acceptable | 服务器无法根据客户端请求的内容特性完成请求 |
| 407 | Proxy Authentication Required | 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权 |
| 408 | Request Time-out | 服务器等待客户端发送的请求时间过长，超时 |
| 409 | Conflict | 服务器完成客户端的PUT请求是可能返回此代码，服务器处理请求时发生了冲突 |
| 410 | Gone | 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置 |
| 411 | Length Required | 服务器无法处理客户端发送的不带Content-Length的请求信息 |
| 412 | Precondition Failed | 客户端请求信息的先决条件错误 |
| 413 | Request Entity Too Large | 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息 |
| 414 | Request-URI Too Large | 请求的URI过长（URI通常为网址），服务器无法处理 |
| 415 | Unsupported Media Type | 服务器无法处理请求附带的媒体格式 |
| 416 | Requested range not satisfiable | 客户端请求的范围无效 |
| 417 | Expectation Failed | 服务器无法满足Expect的请求头信息 |
|  | | |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 501 | Not Implemented | 服务器不支持请求的功能，无法完成请求 |
| 502 | Bad Gateway | 充当网关或代理的服务器，从远端服务器接收到了一个无效的请求 |
| 503 | Service Unavailable | 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中 |
| 504 | Gateway Time-out | 充当网关或代理的服务器，未及时从远端服务器获取请求 |
| 505 | HTTP Version not supported | 服务器不支持请求的HTTP协议的版本，无法完成处理 |