

1.1 生成HTTP请求信息

1.1.1 探索之旅从输入网址开始

1.1.2 浏览器先要解析URL

1.1.3 省略文件名的情况

1.1.4 HTTP的基本思路

- HTTP协议定义了客户端和服务端之间交互的消息内容和步骤
 - 基本思路：
 - 客户端向服务器发送请求消息
 - 请求消息内容：“对什么”+“进行怎样的操作”
 - 对什么：URI，存放网页数据的文件名或者是一个CGI程序的文件名
 - 怎么做：方法
 - GET：访问服务器获取网页数据
 - POST：将网页输入框中的信息发送给Web服务器
 - 收到请求消息之后，Web服务器会对其中的内容进行解析，通过URI和方法来判断“对什么”“进行怎样的操作”，并根据这些要求来完成自己的工作，然后将结果存放在响应消息中。
 - 响应消息的开头有一个状态码，它用来表示操作的执行结果是成功还是发生了错误
 - 访问Web服务器时，遇到找不到的文件就会显示出404 Not Found的错误信息
 - 响应消息会被发回客户端
 - 客户端收到之后，浏览器会从消息中读出所需的数据并显示在屏幕上

1.1.5 客户端生成HTTP请求信息 (http基本思路第一步)

- 浏览器会按照规定的格式来生成请求消息

```
<方法><空格><URL><空格><HTTP版本>    /// 第一行称为请求行，通过这一行可以大致了解请求的内容
<字段名>:<字段值>    /// 这一部分称为消息头，每行包含一个头字段，用于表示请求的附加信息。消息头的行数根据具体情况可变，一直延伸到空行为止
...
...
...
<空行>
<消息体>    /// 消息体包含客户端向服务器发送的数据，例如用POST方法向Web服务器发送的网页表单数据
```

- 第一行是请求行
 - 最开头的方法：告诉Web服务器它应该进行怎样的操作（很多种，需要判断）
 - 判断取决于浏览器的**工作状态**：
 - eg. 向Web服务器发送请求消息：浏览器顶部的地址栏输入网址，网页中的超级链接，表单中填写信息后点击的“提交”按钮
 - 场景出发浏览器的工作，选用哪种方法由场景决定
 - 场景：在地址栏输入网址并显示网页 ==> 方法：GET
 - 场景：表单 ==> 方法：**根据HTML源代码中表单的属性中指定**使用哪种方法来发送请求，可能是GET也可能是POST
 - Tip：GET方法能够发送的数据只有几百个字节，如果表单中的数据超过这一长度，则必须用POST方法来发送。

```

表单HTML源代码：
<form method="GET" action="/cgi/sample.cgi"> // 对方法的指定（“GET”）和接受表单数据的程序文件名
  <input type="text" name="Field1" size="20"> //输入的内容 ==> ABCDEFG
  <input type="submit" value="SEND" name="SendButton"> //提交键
  <input type="reset" value="RESET" name="ResetButton"> //重置键
</form>

请求行【当method="GET"时的消息】：
GET /cgi/sample.cgi?field1=ABCDEFG&SendButton=SEND HTTP/1.1 ---- <方法><空格><URL><空格><HTTP版本>
OR 【当method="POST"时的消息】
POST /cgi/sample.cgi HTTP/1.1
（若干行头字段）
Field1=ABCDEFG&SendButton=SEND // POST方法时，表单填写的信息（ABCDEFG）写在消息体中

```

- 第二行开始为消息头：存放额外的详细信息
 - 消息头的规格中定义了很多项目的细节信息：
 - 日期、客户端支持的数据类型、语言、压缩格式、客户端和服务器的软件名称和版本、数据有效期和最后更新时间等
 - 要想了解这些头字段信息的意思，就需要对**HTTP协议**有非常深入的了解
 - 消息头内容随着浏览器类型、版本号、设置等的不同而不同，大多数情况下消息头的长度为几行到十几行不等
- 当使用POST方法时，需要将表单中填写的信息写在**消息体**中。

```

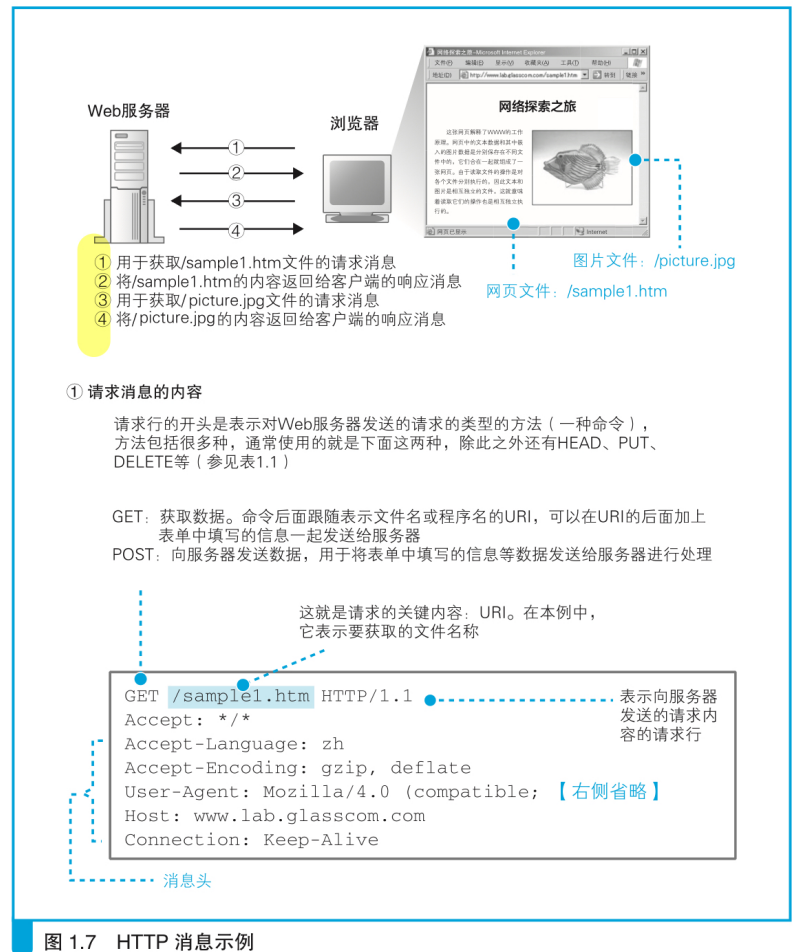
>响应消息
<HTTP版本><空格><状态码><空格><响应短语>    //状态行，用来解释状态码的短语
<字段名>:<字段值>    //消息头
...
...
...
<空行>
<消息体>    //消息体包含服务器向客户端发送的数据，例如从文件中读取的数据，或者CGI应用程序输出的数据等。消息体的内容作为二进制数据来处理

```

1.1.6 发送请求后会收到响应

- 将请求消息发送出去后，Web服务器会返回响应消息。关于响应消息我们将在第6章详细介绍
- 第一行的内容为状态码和响应短语（表示的内容一致，用途不同），用来表示请求的执行结果是成功还是出错
 - 状态码是一个数字，主要用来向程序告知执行的结果
 - 响应短语是一段文字，用来向人们告知执行的结果
- 返回响应消息之后，浏览器会将数据提取出来并显示在屏幕上，我们就能够看到网页的样子了
 - 如果网页只有文字，浏览器请求&服务器响应到此为止
 - 网页中还包括图片等资源，会在相应位置嵌入形如的标签，浏览器会在显示文字时搜索相应的标签，当遇到图片相关的标签时，会在屏幕上留出用来显示图片的空间，再次访问Web服务器请求获取相应的图片并显示在预留的空间中。
 - 比如一个网页中包含三张图片，那么获取网页加上获取图片，一共需要向Web服务器发送4条请求
- 浏览器需要判断所需的文件，然后获取这些文件并显示在屏幕上，这一系列工作的整体指挥也是浏览器的任务之一，而Web服务器却毫不知情
- 服务器的任务是对每一条单独的请求返回1条响应

1.1.7 实例



再分析sample1.html的内容返回给客户端的响应消息

服务器程序类型 状态信息。"200 OK"表示请求成功完成

```
HTTP/1.1 200 OK
Date: Wed, 21 Feb 2007 09:19:14 GMT
Server: Apache
Last-Modified: Mon, 19 Feb 2007 12:24:51 GMT
ETag: "5a9da-279-3c726b61"
Accept-Ranges: bytes
Content-Length: 432
Connection: close
Content-Type: text/html

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>网络探索之旅</title>
</head>
<body>
  align="center">网络探索之旅</h1>
  

```

这张网页解释了WWW的工作原理，网页中的文本数据和其中嵌入的图片数据是分别存在不同文件中的，它们存在一起就组成了一个网页。由于读取网页的操作是对各个文件分别执行的，因此文本和图片是相互独立的文件，这就意味着读取它们的操作也是相互独立执行的。

以MIME规范表示的数据格式。text/html表示HTML文档，如果是JPEG格式的图片，这里应该表示image/

这是嵌入的图片文件的名字。在接下来的请求中向服务器器获取这个文件（如下👇）

3 用于获取 picture.jpg 文件的请求消息

```
GET /picture.jpg HTTP/1.1
Accept: */*
Referer: http://www.lab.glasscom.com/sample1.htm
Accept-Language: zh
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; 【右侧省略】)
Host: www.lab.glasscom.com
Connection: Keep-Alive
```

4 将 picture.jpg 的内容返回给客户端的响应消息

```
HTTP/1.1 200 OK
Date: Wed, 21 Feb 2007 09:19:14 GMT
Server: Apache
Last-Modified: Mon, 19 Feb 2007 13:50:32 GMT
ETag: "5a9d1-1913-3aefa236"
Accept-Ranges: bytes
Content-Length: 6419
Connection: close
Content-Type: image/jpeg
```

image/jpeg表示JPEG格式的图片数据

【下面就是图片数据，因为这些数据都是二进制的，所以我们在右侧省略】