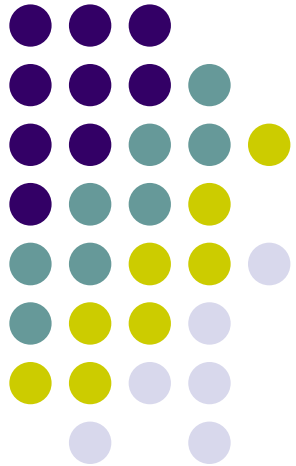
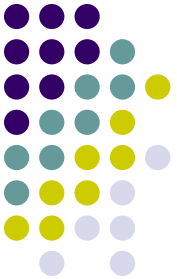
第四章 HTTP 客户/服务器通信协议



主要内容



- HTTP协议概述
- Web应用部署模式
- HTTP协议的基本内容
- 网络抓包分析HTTP协议

HTTP协议概述

Web采用的传输协议是HTTP，即超文本传输协议(Hypertext Transfer Protocol)，它是一个应用级协议，它构成了我们从浏览器发出的所有指令的基础。

它是一个请求/应答式的无状态协议，是Internet上大部分通信的基础，它具有对于具体的网络的无关性，就像Java对于平台无关一样。

此外，HTTP还是一个基于消息的协议，即在客户和服务器之间的通信由可读的文本组成，无需象二进制的协议（如java RMI等）需要将指令进行编码成为二进制形式。例如：

Http://www.example.com/index.html的一个

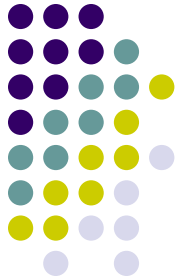
Web页面的请求示例：

GET /index.html/1.1

HOST:www.example.com

应答的示例为：

```
HTTP/1.1 200
Date:Sun,01 Jul 2001 12:00:01 GMT
Content_Type:text/html
<HTML>
    <HEAD></HEAD>
    <BODY>
        Hello World!
    </BODY>
</HTML>
```



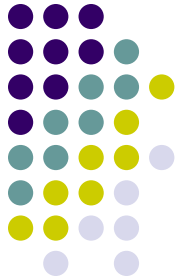
当然，与二进制协议相比，基于消息的协议有一个明显的缺点，即消息要大一些。由于各字符都要转换成为二进制的0或者1才能在传输线路上进行传输，所以产生的消息就要大一些。

例如，要表示GET则需要用3*8=24位，而在二进制协议中需要的位数要少许多，当然消息的协议一个主要的优点就是比较简单，可以直接看出来发生了什么，易于调试。

刚才我们看了一个简单的HTTP工作的例子，那么HTTP这种请求/应答式的协议在工作时具体包括那些内容呢？

请求包括的部分：请求方法、通用资源标识符(URI)、协议版本以及MIME消息(服务器信息、一些元信息和消息体组件)

服务器作出的响应部分：状态行、协议版本、一个成功或错误码以及MIME消息



MIME(Multipurpose Internet Mail Extension, 多用途Internet邮件扩展），它原来的基础是RFC(请求注解文档Requests for Comments)，MIME RFC主要说明文本消息和消息体是如何格式化的。

HTTP是一个应用级的协议，根据计算机网络的OSI模型，它位于TCP等会话级协议之上，其部署是可选的。

当前存在两个版本的HTTP：1.0和1.1，我们一般的情况下使用1.1多一些。

Web应用部署模式

Web应用有两种基本的部署模式

1、以Web浏览器作为应用客户

浏览器的主要工作一是基于用户的指令与Web服务器通信，二是接受由做出响应的服务器输出的内容。

2、不使用浏览器的其他应用客户

这类客户通常是无线PDA，集成的语音响应系统，甚至是运行在远程主机上的软件代理，这时可能会是使用到其他的传输协议。

HTTP协议的主要内容

- 整体协议语义
- HTTP URL
- HTTP请求
- HTTP响应
- HTTP消息报头
- 持续连接及缓存支持

整体协议语义

HTTP是部署在TCP/IP之上的，其请求和应答将通过TCP通道发送，而这个TCP通道是有客户和服务器所建立的，使用的是TCP协议而不是UDP协议，数据包的发送是可以得到保证的

那么HTTP的整个请求/响应的过程为：

(1)客户向服务器请求建立一个TCP连接

(2)服务器同意建立连接，并向客户做出应答

HTTP是部署在TCP/IP之上的，其请求和应答将通过TCP通道发送，而这个TCP通道是有客户和服务器所建立的，使用的是TCP协议而不是UDP协议，数据包的发送是可以得到保证的

那么HTTP的整个请求/响应的过程为：

(1)客户向服务器请求建立一个TCP连接

(2)服务器同意建立连接，并向客户做出应答

(3)客户在这个打开的通道上发送一个HTTP请求

(4)服务器接收到此请求，并对它做出响应

上面的步骤说明需要两次客户/服务器的通信往返：一个往返是为了初始化连接的建立，另外一个单向通信是为了完成客户的HTTP请求，还有一个单向通信则是返回数据的开始部分。

整个传输过程为：(2*RT)+Tc

RT=客户/服务器之间的往返时间

Tc=传输实际内容字节的时间

HTTP URL

HTTP URL（Uniform Resource Locator）的格式如下：

http://host [“:” port] [abs_path]

其中http表示要通过HTTP协议来定位网络资源；host表示合法的Internet主机域名或IP地址（以点分十进制的格式表示）；port用于指定一个端口号，拥有被请求资源的服务器主机监听该端口的TCP连接，如果port是空，或者没有给出，则使用缺省的端口80。

如果URL没有给出abs_path，那么当它作为请求URI（Uniform Resource Identifier，统一资源标识符）时，必须以“/”的形式给出。通常这个工作浏览器帮我们完成了。我们在浏览器中输入www.sina.com.cn，然后回车，浏览器会自动将我们输入的地址转换为http://sina.com.cn/。

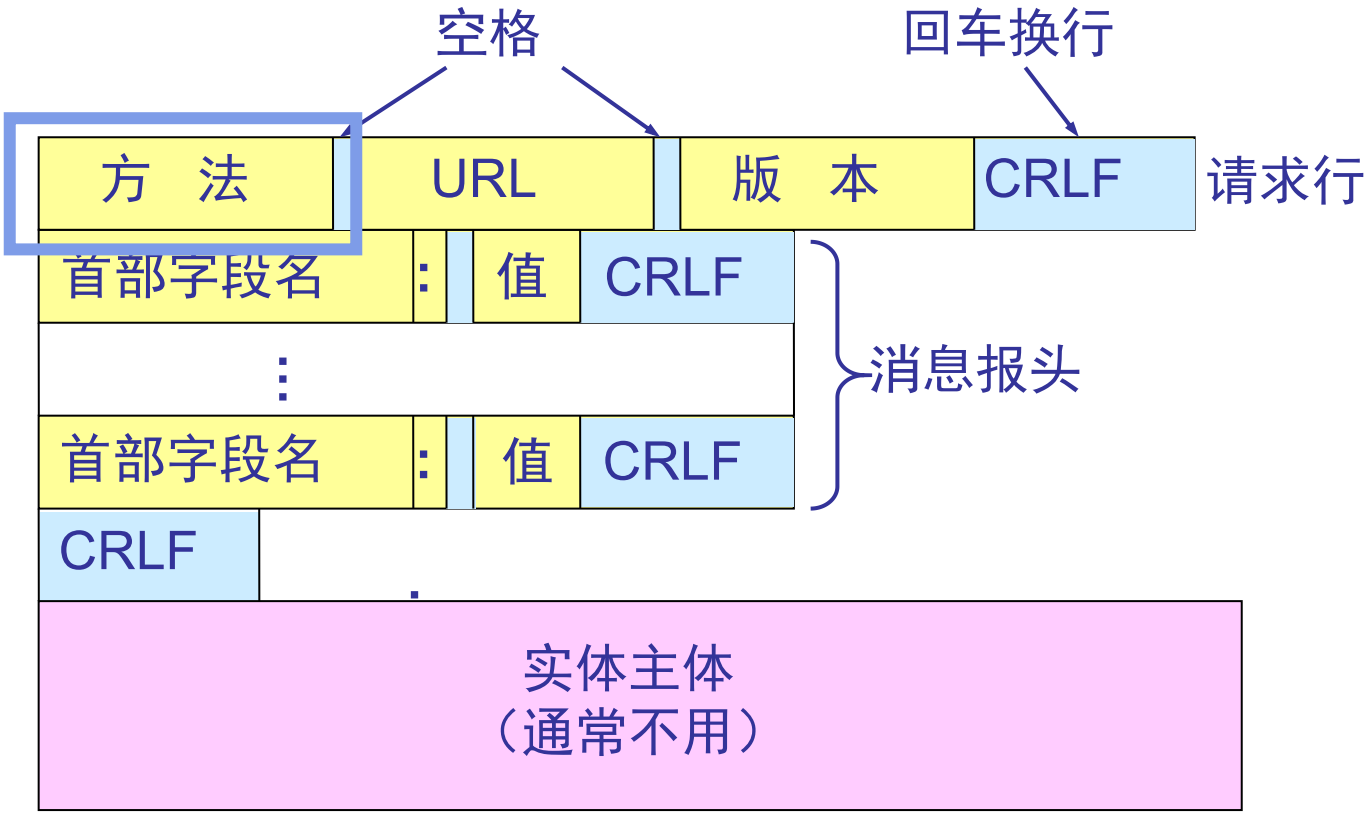
注意URI与URL的区别：

URI:纯粹是一个符号结构，用于指定构成Web资源的字符串的各个不同部分。

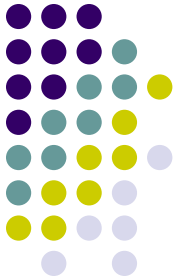
URL:是一种特殊类型的URI，它包含了用于查找某个资源的足够的信息。例如mailto:zhangsan@sina.com.cn则不属于URL，因为它里面不存在根据该标识符来查找的任何数据。这种URI成为URN（通用资源名）

HTTP请求

客户端通过发送HTTP请求向服务器请求对资源的访问；HTTP请求报文由三部分组成，分别是：请求行，消息报头和请求正文。



请求报文的类型是由它所采用的方法决定的。

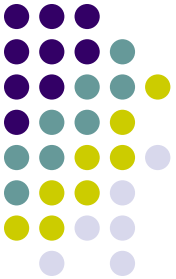


请求行以一个方法符号开头，后面跟着请求URI和协议的版本，以CRLF作为结尾。请求行以空格分隔，除了作为结尾的CRLF外，不允许出现单独的CR或LF字符。格式如下：

Method Request-URI HTTP-Version CRLF

Method表示请求的方法，Request-URI是同一资源标识符，标识了要请求的资源，HTTP-Version表示请求的HTTP协议版本，CRLF表示回车换行。例如：

GET /form.html HTTP/1.1（CRLF）



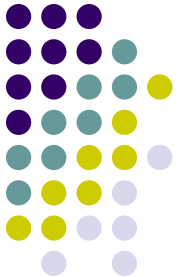
http请求方法

方法	作用
GET	请求获取由Request-URI所标识的资源。
POST	在Request-URI所标识的资源后附加新的数据。
HEAD	请求获取由Request-URI所标识的资源的响应消息报头。
PUT	请求服务器存储一个资源，并用Request-URI作为其标识。
DELETE	请求服务器删除由Request-URI所标识的资源。
TRACE	请求服务器回送收到的请求信息，主要用于测试或诊断。
CONNECT	保留将来使用。
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求。

（1）GET方法

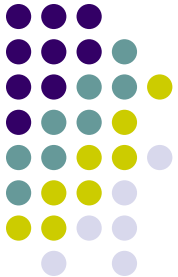
GET方法即获取由Request-URI标识的任何信息（以实体的形式）。如果Request-URI引用某个数据处理过程，则应该以它产生的数据作为在响应中的实体，而不是该过程的源代码文本。

如果请求消息包括If-Modified-Since、If-Unmodified-Since、If-Match、If-None-Match或者If-Range头部域，则GET方法的语义变为“条件GET”。条件GET方法请求只传输在条件头部域描述情形下的实体。条件GET方法试图通过允许刷新缓存的实体而不需要多次请求或传输客户端已经拥有的数据来减少非必要的网络使用。



如果请求消息包括Range头部域，则GET方法的语义变为“局部GET”。局部GET请求只需传输实体的某部分。局部GET方法试图通过允许部分获取实体来完成而不需要传输客户端已经拥有的数据来减少非必要的网路使用。

缓存静态GET请求：通过条件GET或局部GET请求静态网页，例如： GET /citys.html HTTP/1.1 If-Modified-Since:Mon,3 Apr 2001 18:00:00 GMT这个请求的对象就是从上述日期以后已经被改动了的，而那些未改动的对象服务器就不返回了。



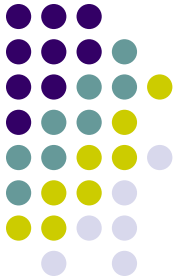
缓存动态GET请求：带参数的URL请求

这种情况下，客户向服务器传送参数，主要采用的机制就是HTML FROM元素，它支持客户的输入发送给服务器，给予所请求的URI和客户端参数就能生成一个定制的HTTP应答（Web页面）。

例如：一个带参数的GET请求

GET /cityquery.cgi?city=paris%2C+France

HTTP/1.1



（2）POST方法

POST是HTTP协议中更为通用的请求对象方法，它用来请求原始服务器接受请求中封装的实体作为从属请求行中的Request-URI标识的附属。POST设计允许完成下列功能的统一方法：

- ✓ 注解存在的资源
- ✓ 上传消息到论坛、新闻组或相似的讨论组
- ✓ 向数据处理过程提供数据库，如提交表单的结果
- ✓ 通过追加操作来扩展数据库

基于POST请求，服务器可以简单地利用一个确认做出响应，不提供任何其他数据，也可以指示该实体已得到创建，并提供此创建相对应的信息。

（3）HEAD方法

除了服务器不能在响应中返回消息体，HEAD方法与GET相同。所以HEAD方法只是请求消息报头，而不是完整的内容。对于HEAD请求的回应部分来说，他的HTTP头部中包含的信息与通过GET请求所得到的信息是相同的。

利用这个方法，不必传输整个资源内容，就可以得到Request-URI所标识的资源的信息。这个方法通常被用于测试超链接的有效性，是否可以访问，以及最近是否更新等。

注意：

当我们在HTML中提交表单时，浏览器会根据你的提交方法是get还是post，采用响应的在HTTP协议中的GET或POST方法，向服务器发出请求。这里需要注意的是：在HTML文档中，书写get和post，大小写都可以，但HTTP协议中的GET和POST只能是大写形式。

HTTP请求的消息报头和请求正文

- 消息报头在后面讲述
- 请求正文中可以包含提交表单的数据

POST方法提交表单的HTTP请求示例：

```
POST /reg.jsp HTTP/ (CRLF)

Accept:image/gif,image/x-xbit,... (CRLF)
...
HOST:www.guet.edu.cn (CRLF)
Content-Length:22 (CRLF)
Connection:Keep-Alive (CRLF)
Cache-Control:no-cache (CRLF)
(CRLF) //该CRLF 表示消息报头已经结束，在此之前为消息报头

user=jeffrey&pwd=1234 //此行以下为提交的数据
```

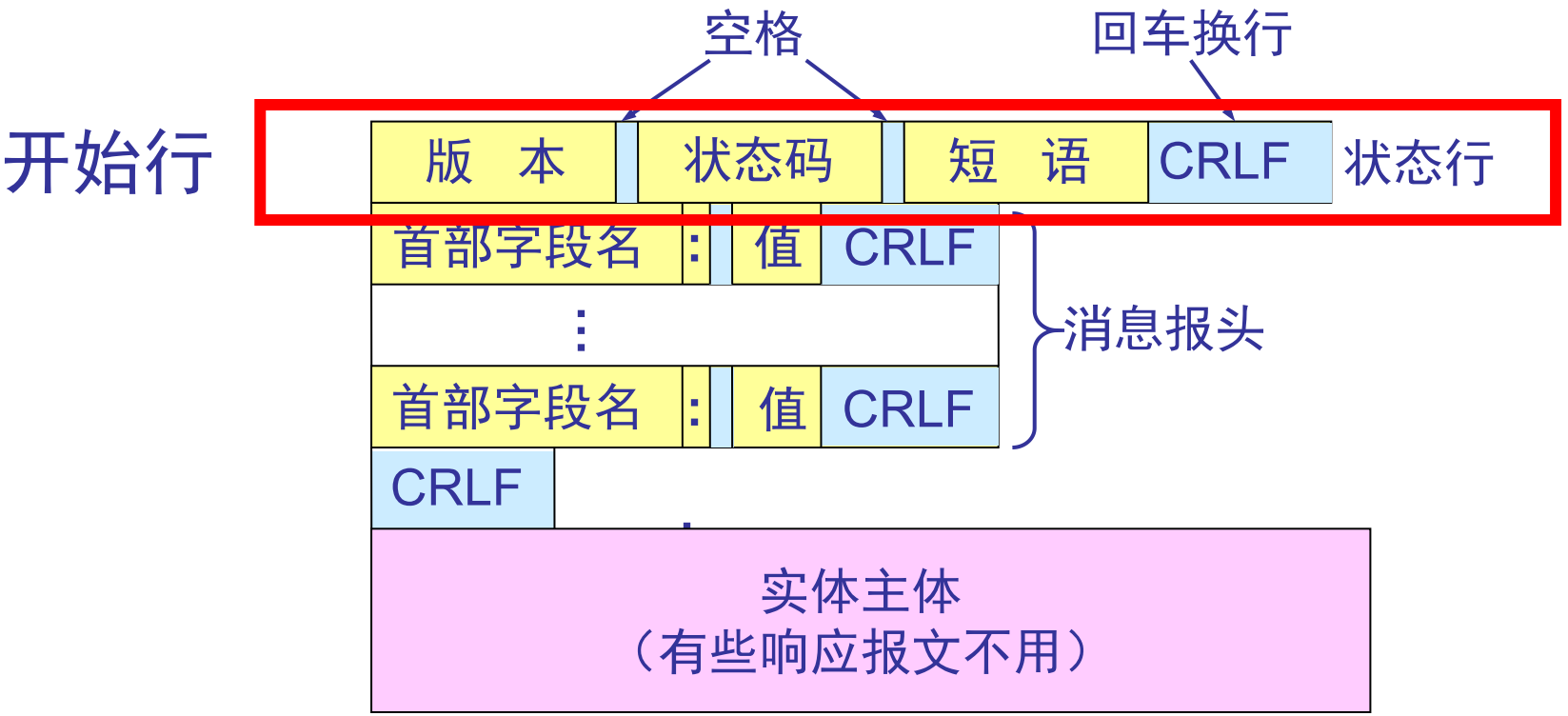
请求行

消息报头

请求正文

HTTP响应

在接收和解释请求消息后，服务器返回一个HTTP 响应消息，HTTP 响应也是由三个部分组成，分别是：状态行、消息报头、响应正文。



状态行由协议版本、数字形式的状态代码、及相应的状态描述组成，各元素之间以空格分隔，除了结尾的CRLF（回车换行）序列外，不允许出现CR或LF字符。格式如下：

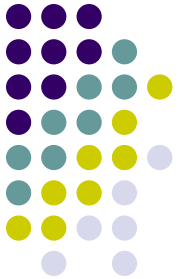
HTTP-Version Status-Code Reason-Phrase CRLF

HTTP-Version表示服务器HTTP协议的版本，status-Code标识服务器发回的响应代码，Reason-Phrase表示状态代码的文本描述，CRLF标识回车换行。例如：

HTTP/1.1 200 OK（CRLF）

状态代码由3位数字组成，标识请求是否被理解或被满足，状态描述给出了关于状态代码的简短的文本描述。状态代码的第一个数字定义了相应的类别，后面两位数字没有具体的分类。第一个数字有五种可能的取值：

- ✓ 1xx：指示信息-表示请求已接受，继续处理。
- ✓ 2xx：成功-表示请求已经被成功接收、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。



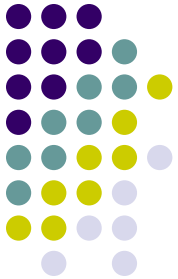
常见状态代码以及文本描述

状态代码	状态描述	说明
200	OK	客户端请求成功
400	Bad Request	由于客户端请求有语法错误，不能被服务器所理解
401	Unauthorized	请求未经授权。这个状态代码必须和www-Authenticate报头域一起使用。
403	Forbidden	服务器收到请求，但是拒绝提供服务。服务器通常会在响应正文中给出不提供服务的原因。
404	Not Found	请求的资源不存在，例如，输入错误的URL。
500	Internal Server Error	服务器发生不可预期的错误，导致无法完成客户端的请求。
503	Service Unavailable	服务器当前不能够处理客户端的请求，在一段时间之后服务器可能会恢复正常。

HTTP消息报头

HTTP 消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行，消息报头（可选），空行（只有CRLF 的行），消息正文（可选）组成。对于请求消息，开始行就是请求行，对于响应消息，开始行就是状态行。

HTTP 消息报头包括普通报头、请求报头、响应报头、实体报头。每一个报头域都是由名字+“：”+空格+值组成，消息报头域的名字是大小写无关的。



请求消息示例：

GET /form.html HTTP/1.1 (CRLF)

Accept:image/gif,image/x-xbitmap,image/jpeg,application/x-shockwave-flash,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,*/* (CRLF)

Accept-Language:zh-cn (CRLF)

Accept-Encoding:gzip,deflate (CRLF)

If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF)

If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF)

User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF)

Host:www.guet.edu.cn (CRLF)

Connection:Keep-Alive (CRLF)

(CRLF)

响应消息示例：

HTTP/1.1 200 OK（CRLF）
Content-Length 2218（CRLF）
Content-Type text/html（CRLF）
Last-Modified Wed 05 Jan 2005 11:21:51 GMT（CRLF）
Accept-Ranges bytes（CRLF）
ETag vw”80b1a4c018t3c41:88td”（CRLF）
Server Microsoft-IIS/6.0（CRLF）
Date Wed,05 Jan 2005 05:48:54 GMT（CRLF）
CRLF（表示一个空行）

```
<html>
    <head>
        <title>
            This is the form page.
        </title>
    </head>
    <body>
        ……(省略)
    </body>
</html>
```

（1）普通报头

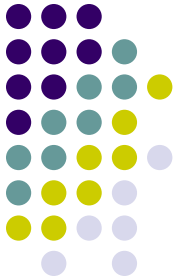
在普通报头中，有少数报头域用于所有的请求和响应消息，但并不用于被传输的实体，只用于传输的消息。

举例：

Cache-Control 用于指定缓存指令，缓存指令是单向的（响应中出现的缓存指令在请求中未必会出现），且是独立的（一个消息的缓存指令不会影响另一个消息处理的缓存机制），HTTP1.0 使用的类似的报头域为 Pragma。

请求时的缓存指令包括：no-cache（用于指示请求或响应消息不能缓存）、no-store、max-age、max-stale、min-fresh、only-if-cached;

响应时的缓存指令包括： public 、private 、no-cache 、no-store 、no-transform 、 must-revalidate、 proxy-revalidate、 max-age、s-maxage.



举例：

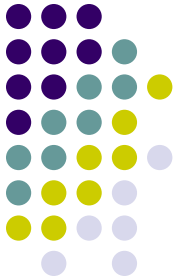
为了指示IE 浏览器（ 客户端） 不要缓存页面， 服务器端的JSP 程序可以编写如下：

response.sehHeader("Cache-Control","no-cache");

//response.setHeader("Pragma","no-cache");作用相当于上述代码，通常两者//合用这句代码将在发送的响应消息中设置普通报头域：Cache-Control:no-cache

Date 普通报头域表示消息产生的日期和时间

Connection 普通报头域允许发送指定连接的选项。例如指定连接是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接



（2）请求报头

请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。常用的请求报头：

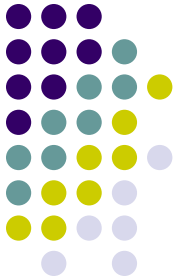
- Accept 请求报头域用于指定客户端接受哪些类型的信息。

  例如：Accept：image/gif，表明客户端希望接受GIF 图象格式的资源；Accept：text/html，表明客户端希望接受html 文本。

- Accept-Charset 请求报头域用于指定客户端接受的字符集。例如 ：Accept-Charset:iso-8859-1,gb2312.如果在请求消息中没有设置这个域，缺省是任何字符集都可以接受。

- Accept-Encoding 请求报头域类似于Accept，但是它是用于指定可接受的内容编码。例如：Accept-Encoding:gzip.deflate.如果请求消息中没有设置这个域服务器假定客户端对各种内容编码都可以接受。

- Accept-Language 请求报头域类似于Accept ， 但是它是用于指定一种自然语言。例如：Accept-Language:zh-cn.如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。



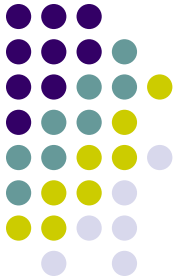
- Authorization 请求报头域主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为401（未授权），可以发送一个包含Authorization 请求报头域的请求，要求服务器对其进行验证。
- Host （发送请求时，该报头域是必需的）请求报头域主要用于指定被请求资源的Internet 主机和端口号， 它通常从HTTP URL 中提取出来的，例如：我们在浏览器中输入：http://www.guet.edu.cn/index.html浏览器发送的请求消息中，就会包含Host 请求报头域，如下：Host：www.guet.edu.cn此处使用缺省端口号80，若指定了端口号，则变成：Host：www.guet.edu.cn:指定端口号
- User-Agent 请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。不过，这个报头域不是必需的，如果我们自己编写一个浏览器，不使用User-Agent请求报头域，那么服务器端就无法得知我们的信息了。

（3）响应报头

响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对Request-URI 所标识的资源进行下一步访问的信息。常用的响应报头有：

- Location 响应报头域用于重定向接受者到一个新的位置。Location 响应报头域常用在更换域名的时候。
- Server 响应报头域包含了服务器用来处理请求的软件信息。与User-Agent 请求报头域是相对应的。下面是Server 响应报头域的一个例子：

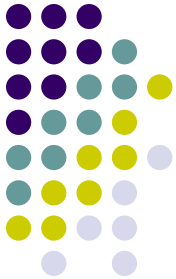
  Server：Apache-Coyote/1.1
- WWW-Authenticate 响应报头域必须被包含在401（未授权的）响应消息中，客户端收到401 响应消息时候，并发送Authorization 报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。例如：WWW-Authenticate:Basic realm="Basic Auth Test!" //可以看出服务器对请求资源采用的是基本验证机制。



（4）实体报头

请求和响应消息都可以传送一个实体。一个实体由实体报头域和实体正文组成，但并不是说实体报头域和实体正文要在一起发送，可以只发送实体报头域。实体报头定义了关于实体正文和请求所标识的资源的元信息。常用的实体报头有：

- Content-Encoding 实体报头域被用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码，因而要获得Content-Type 报头域中所引用的媒体类型，必须采用相应的解码机制。Content-Encoding 这样用于记录文档的压缩方法，例如：Content-Encoding：gzip
- Content-Language 实体报头域描述了资源所用的自然语言。没有设置该域则认为实体内容将提供给所有的语言阅读者。例如：Content-Language:da



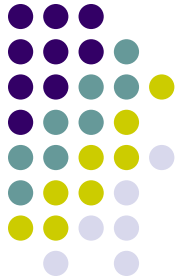
- Content-Length 实体报头域用于指明实体正文的长度，以字节方式存储的十进制数字来表示。
- Content-Type 实体报头域用语指明发送给接收者的实体正文的媒体类型。例如：

  Content-Type:text/html;charset=ISO-8859-1

  Content-Type:text/html;charset=GB2312
- Last-Modified 实体报头域用于指示资源的最后修改日期和时间
- Expires 实体报头域给出响应过期的日期和时间。为了让代理服务器或浏览器在一段时间以后更新缓存中(再次访问曾访问过的页面时，直接从缓存中加载，缩短响应时间和降低服务器负载)的页面，我们可以使用Expires 实体报头域指定页面过期的时间。eg：Expires：Thu，15 Sep 2006 16:23:12GMT

  HTTP1.1 的客户端和缓存必须将其他非法的日期格式（包括0）看作已经过期。例如：为了让浏览器不要缓存页面， 我们也可以利用Expires 实体报头域， 设置为0 ， jsp 中程序如下：response.setDateHeader("Expires","0");

持续连接及缓存支持



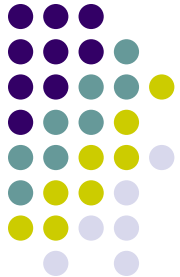
- **持续链接**

  HTTP的语义告诉我们，一个使用TCP的简单的HTTP连接需要两个通信往返：一个用于建立连接，另一个用于实现内容传输。

  随着Web群体的不断扩大、Web网页和Web网站变得越来越复杂，那么HTTP连接就将面临很大的挑战，另一方面在传输Web对象的过程中存在着很大的浪费。

为了弥补HTTP的缺陷，协议设计人员提出了一种持续连接的概念，基本思想就是利用已建立的与服务器的TCP连接，来传输与此网站或页面相关的其他对象，这样可以大大提高传输的效率。

另外，持续连接还提供了一个重要的特性：请求管道，它允许客户通过一个HTTP连接来传输一组请求，而不是在分发下一个请求前等待前一个请求的响应。



- **缓存支持**

前面讨论了静态和动态的GET响应可以得到缓存，即根据GET请求，为对象在本地建立一个副本。缓存机制使得客户和服务器都需要对请求/应答过程有所影响。

对于服务器在其响应中可以指示关于内容的以下信息：

(1) 与内容过期相关的日期

(2) 确保缓存一致性的标签

(3)响应是否需要显示地缓存

(4) 需要用那一种类型的缓存来对响应进行缓存

而客户端对应可以请求某个资源的整个内容

(1)有条件的，如果此内容比客户指定的日期更接近

(2)有条件的，如果内容标签与客户所制定的不同

(3)无条件的，忽略所有基于服务器的缓存建议

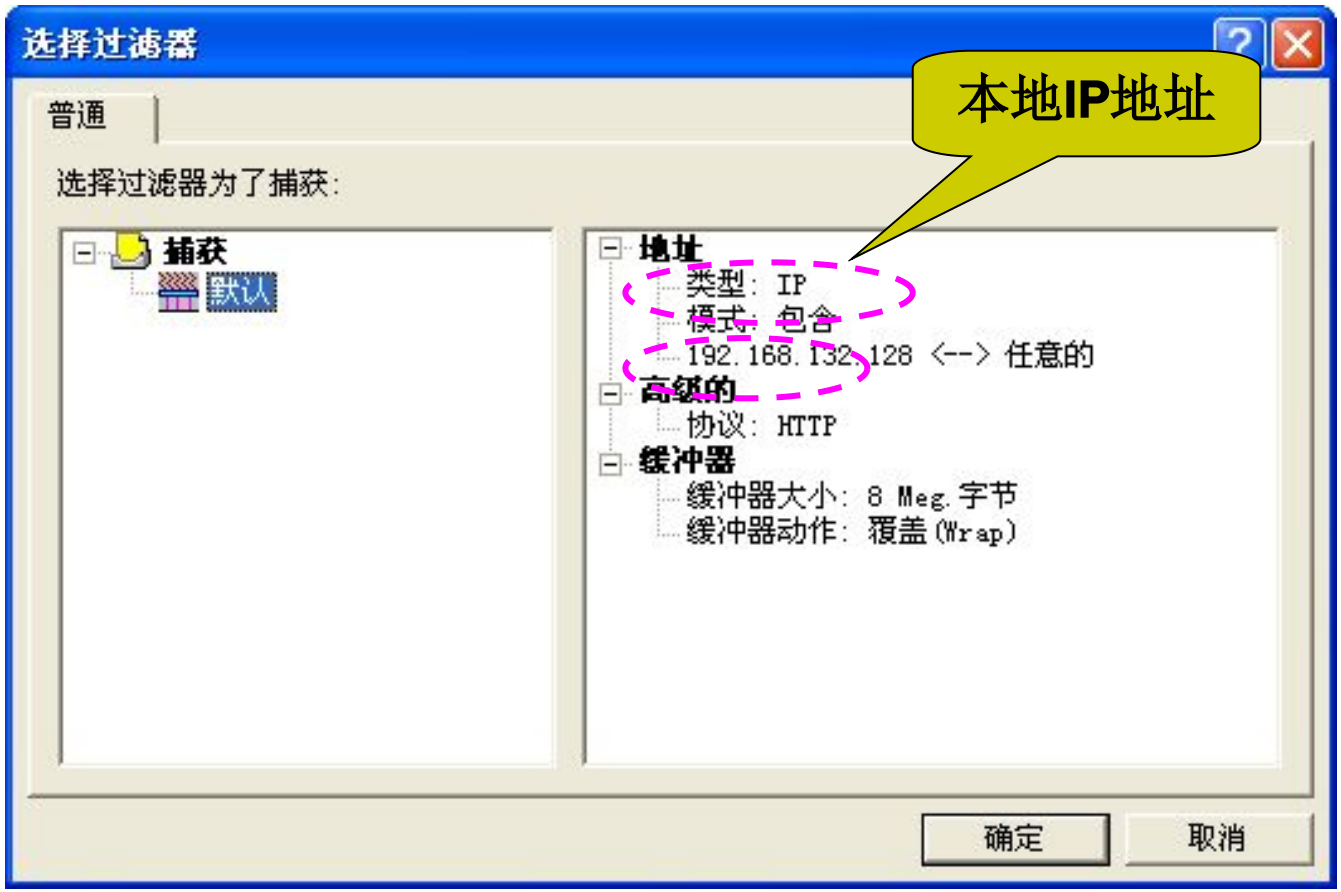
浏览器一般使用基于日期和标签的验证方法来验证缓存是否刷新。

网络抓包分析HTTP协议

- 通过网络抓包分析工具分析HTTP协议连接，请求/响应及关闭过程。
- 常用的网络抓包分析软件有： Sniffer、WireShark及Solarwinds等。
- 使用Sniffer抓包分析HTTP协议

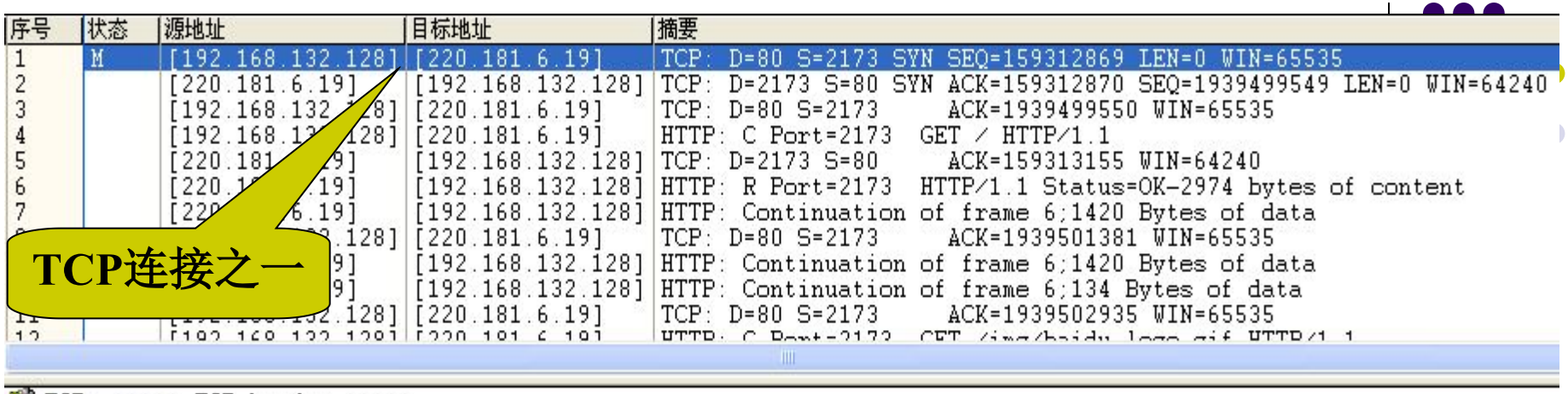
HTTP抓包分析

- 设置Sniffer 4.7 抓包过滤器



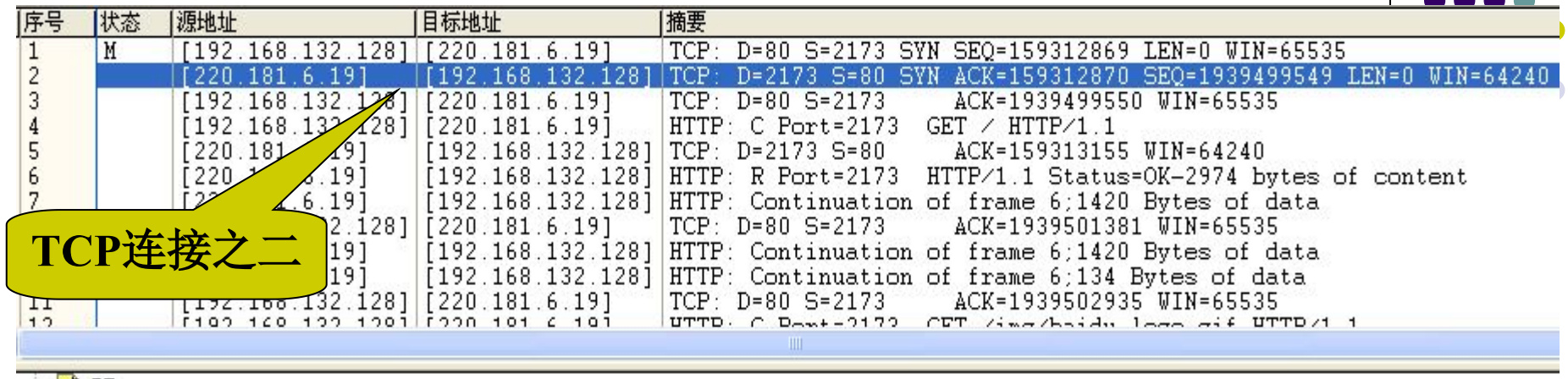
HTTP抓包分析

- 抓包的对象为百度
- 抓包的目标是查看 HTTP连接、请求/响应 和 关闭过程




```
TCP: ----- TCP header -----
TCP:
TCP: Source port              =  2173
TCP: Destination port         =    80 (WWW/WWW-HTTP/HTTP)
TCP: Initial sequence number = 159312869
TCP: Next expected Seq number= 159312870
TCP: Data offset              = 28 bytes
TCP: Reserved Bits: Reserved for Future Use (Not shown in the Hex Dump)
TCP: Flags                    = 02
TCP:                 ..0. .... = (No urgent pointer)
TCP:                 ...0 .... = (No acknowledgment)
TCP:                 .... 0... = (No push)
TCP:                 .... .0.. = (No reset)
TCP:                 .... ..1. = SYN
TCP:                 .... ...0 = (No FIN)
TCP: Window                   = 65535
TCP: Checksum                 = 5CFD (correct)
TCP: Urgent pointer           = 0
TCP:
TCP: Options follow
TCP: Maximum segment size = 1460
TCP: No-Operation
TCP: No-Operation
TCP: SACK-Permitted Option
```

TCP连接请求信号



IP:

TCP: ----- TCP header -----

TCP:

TCP: Source port = 80 (WWW/WWW-HTTP/HTTP)

TCP: Destination port = 2173

TCP: Initial sequence number = 1939499549

TCP: Next expected Seq number= 1939499550

TCP: Acknowledgment number = 159312870

TCP: Data offset = 24 bytes

TCP: Reserved Bits: Reserved for Future Use (Not shown in the

TCP: Flags = 12

TCP: ..0. .... = (No urgent pointer)

TCP: ...1 .... = Acknowledgment

TCP: .... 0... = (No push)

TCP: .... .0.. = (No reset)

TCP: .... ..1. = SYN

TCP: ....0 = (No FIN)

TCP: Window = 64240

TCP: Checksum = 994A (correct)

TCP: Urgent pointer = 0

TCP:

TCP: Options follow

TCP: Maximum segment size = 1460

服务器确认

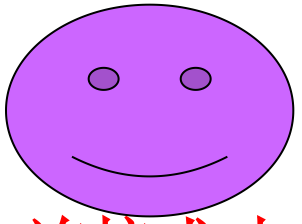
服务器同时也发出连接请求

序号	状态	源地址	目标地址	摘要
1	M	[192.168.132.128]	[220.181.6.19]	TCP: D=80 S=2173 SYN SEQ=159312869 LEN=0 WIN=65535
2		[220.181.6.19]	[192.168.132.128]	TCP: D=2173 S=80 SYN ACK=159312870 SEQ=1939499549 LEN=0 WIN=64240
3		[192.168.132.128]	[220.181.6.19]	TCP: D=80 S=2173 ACK=1939499550 WIN=65535
4		[192.168.132.128]	[220.181.6.19]	HTTP: C Port=2173 GET / HTTP/1.1
5		[220.181.6.19]	[192.168.132.128]	TCP: D=2173 S=80 ACK=159313155 WIN=64240
6		[220.181.6.19]	[192.168.132.128]	HTTP: R Port=2173 HTTP/1.1 Status=OK-2974 bytes of content
7		[220.181.6.19]	[192.168.132.128]	HTTP: Continuation of frame 6;1420 Bytes of data
8		[192.168.132.128]	[220.181.6.19]	TCP: D=80 S=2173 ACK=1939501381 WIN=65535
		[220.181.6.19]	[192.168.132.128]	HTTP: Continuation of frame 6;1420 Bytes of data
		[220.181.6.19]	[192.168.132.128]	HTTP: Continuation of frame 6;134 Bytes of data
		[192.168.132.128]	[220.181.6.19]	TCP: D=80 S=2173 ACK=1939502935 WIN=65535

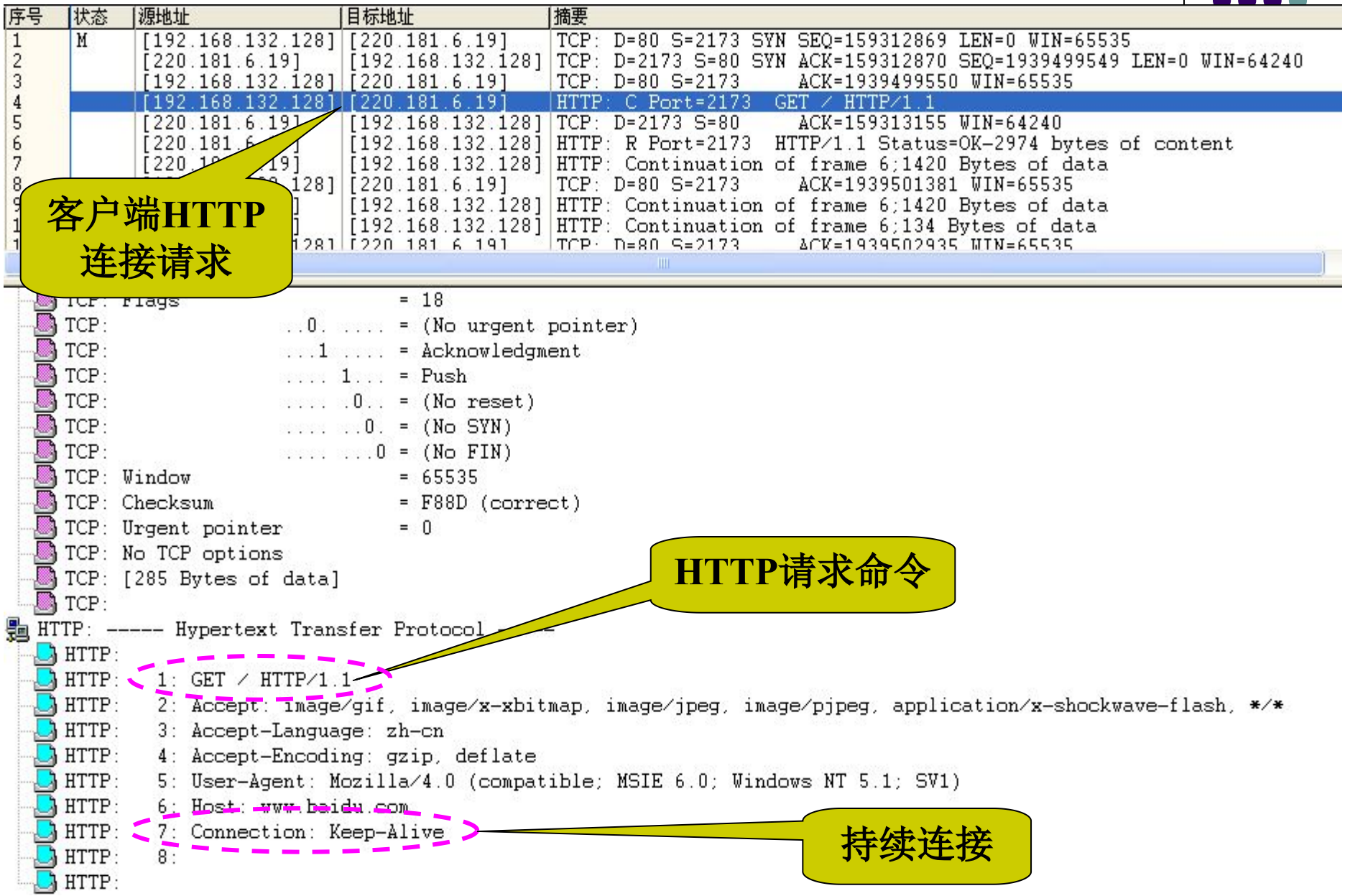
TCP连接之三

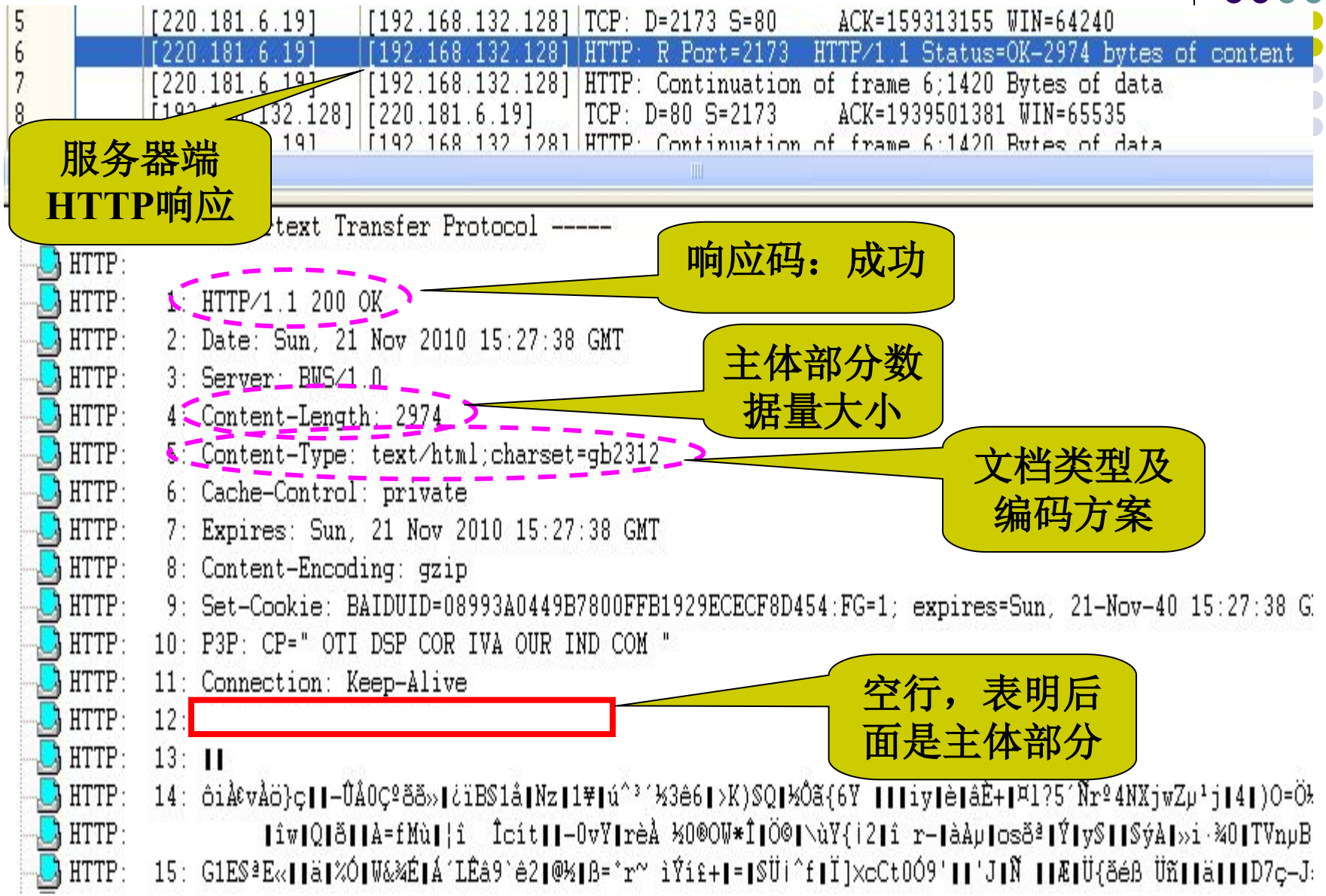
```
IP: No options
IP:
TCP: ----- TCP header -----
TCP:
TCP: Source port              =  2173
TCP: Destination port         =    80 (WWW/WWW-HTTP/HTTP)
TCP: Sequence number          = 159312870
TCP: Next expected Seq number= 159312870
TCP: Acknowledgment number    = 1939499550
TCP: Data offset              = 20 bytes
TCP: Reserved Bits: Reserved for Future Use (Not shown in the Hex Dump)
TCP: Flags                    = 10
TCP:                  ..0. .... = (No urgent pointer)
TCP:                  ...1 .... = Acknowledgment
TCP:                  .... 0... = (No push)
TCP:                  .... .0.. = (No reset)
TCP:                  .... ..0. = (No SYN)
TCP:                  .... ...0 = (No FIN)
TCP: Window                   = 65535
TCP: Checksum                 = ABF8 (correct)
TCP: Urgent pointer           = 0
TCP: No TCP options
```

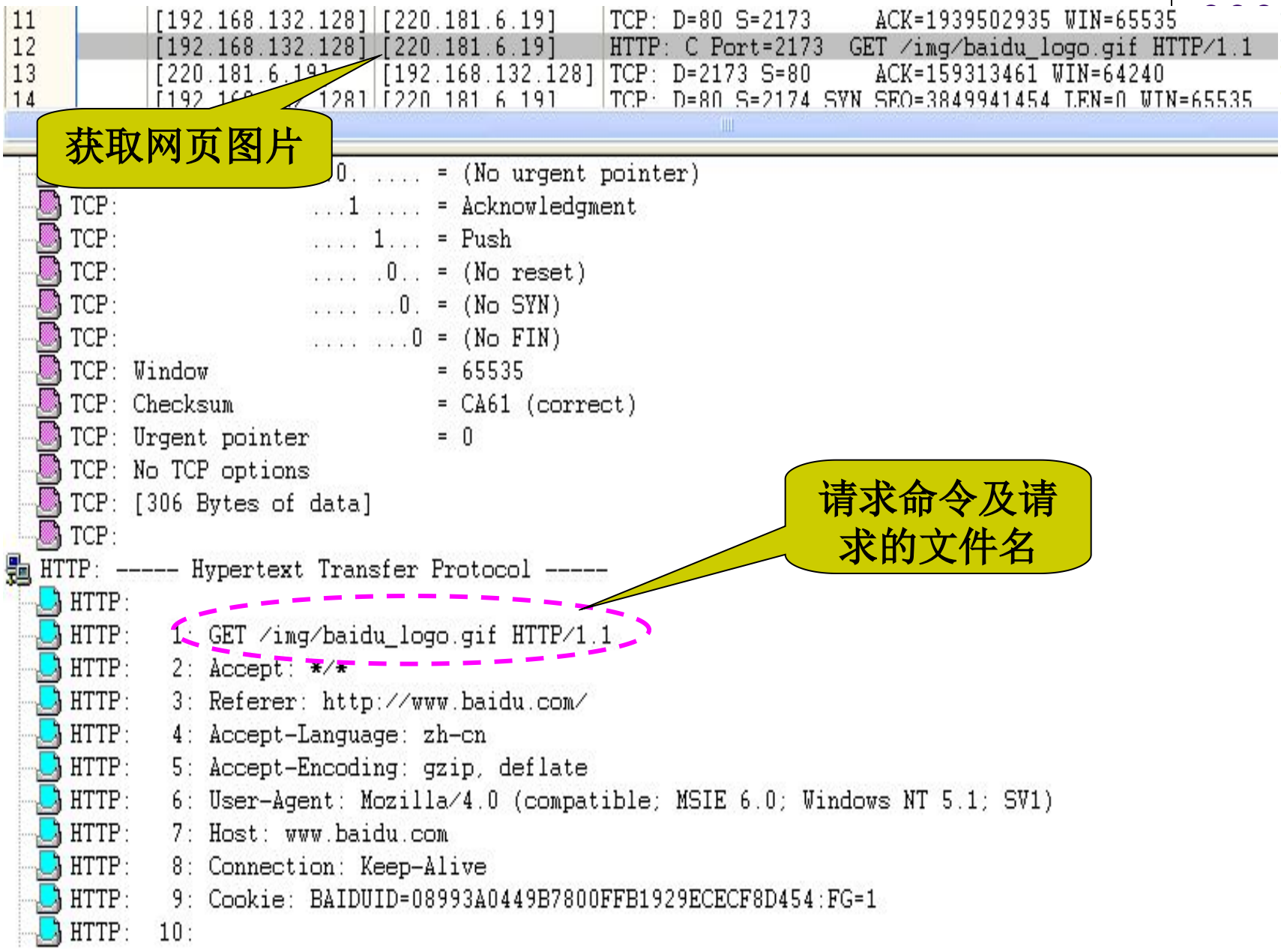
客户端确认

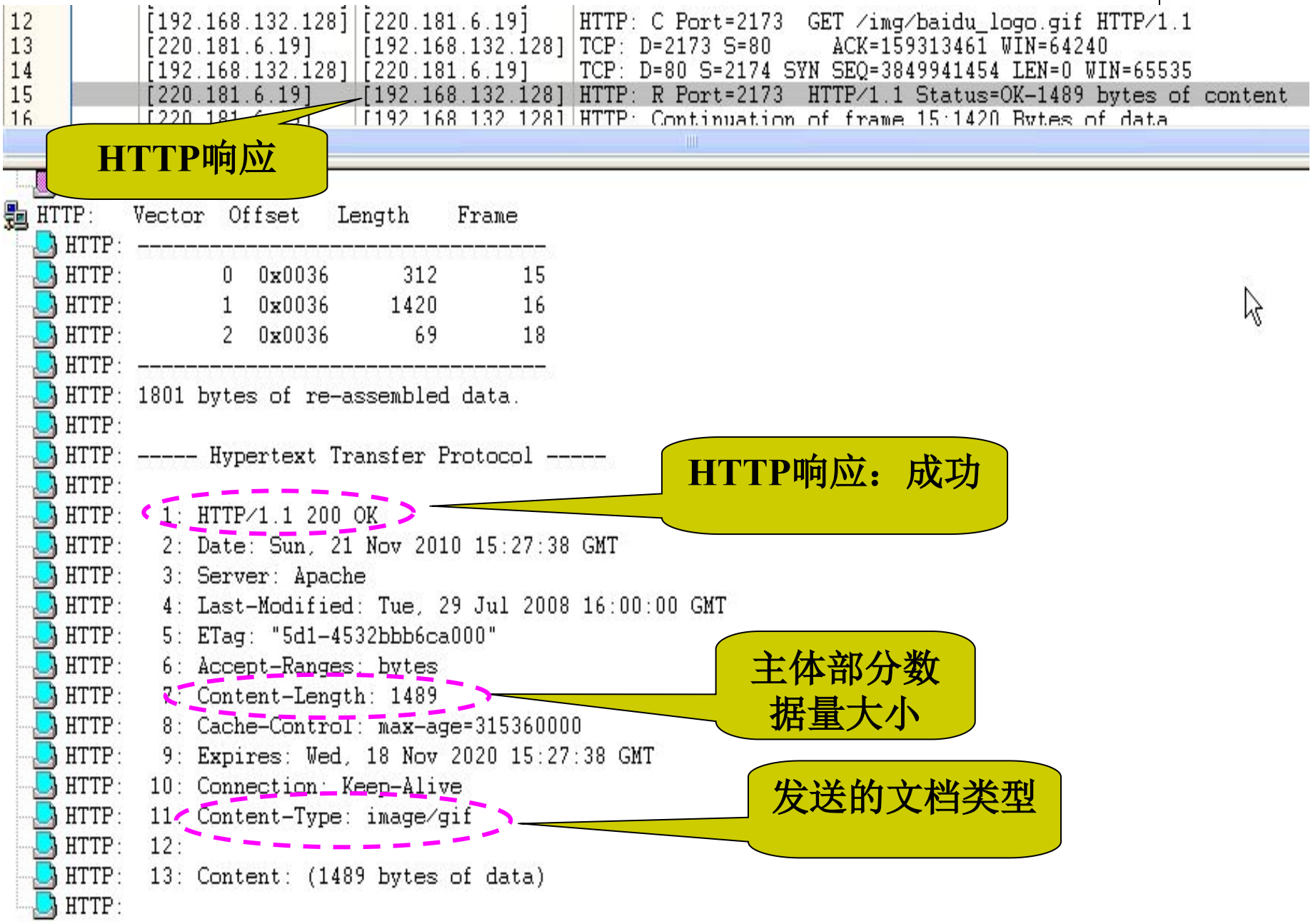


TCP连接成功啦！


序号 状态 源地址 目标地址 摘要
1 M [192.168.132.128] [220.181.6.19] TCP: D=80 S=2173 SYN SEQ=159312869 LEN=0 WIN=65535
2 [220.181.6.19] [192.168.132.128] TCP: D=2173 S=80 SYN ACK=159312870 SEQ=1939499549 LEN=0 WIN=64240
3 [192.168.132.128] [220.181.6.19] TCP: D=80 S=2173 ACK=1939499550 WIN=65535
4 [192.168.132.128] [220.181.6.19] HTTP: C Port=2173 GET / HTTP/1.1
5 [220.181.6.19] [192.168.132.128] TCP: D=2173 S=80 ACK=159313155 WIN=64240
6 [220.181.6] [192.168.132.128] HTTP: R Port=2173 HTTP/1.1 Status=OK-2974 bytes of content
7 [220.18 19] [192.168.132.128] HTTP: Continuation of frame 6;1420 Bytes of data
8 128] [220.181.6.19] TCP: D=80 S=2173 ACK=1939501381 WIN=65535
9] [192.168.132.128] HTTP: Continuation of frame 6;1420 Bytes of data
1] [192.168.132.128] HTTP: Continuation of frame 6;134 Bytes of data
1 128] [220.181.6.19] TCP: D=80 S=2173 ACK=1939502935 WIN=65535
客户端HTTP连接请求
TCP: Flags = 18
TCP: ..0. .... = (No urgent pointer)
TCP: ...1 .... = Acknowledgment
TCP: .... 1... = Push
TCP: .... .0.. = (No reset)
TCP: .... ..0. = (No SYN)
TCP: ....0 = (No FIN)
TCP: Window = 65535
TCP: Checksum = F88D (correct)
TCP: Urgent pointer = 0
TCP: No TCP options
TCP: [285 Bytes of data]
TCP:
HTTP: ----- Hypertext Transfer Protocol -----
HTTP:
HTTP: 1: GET / HTTP/1.1
HTTP: 2: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
HTTP: 3: Accept-Language: zh-cn
HTTP: 4: Accept-Encoding: gzip, deflate
HTTP: 5: User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
HTTP: 6: Host: www.baidu.com
HTTP: 7: Connection: Keep-Alive
HTTP: 8:
HTTP:
HTTP请求命令
持续连接

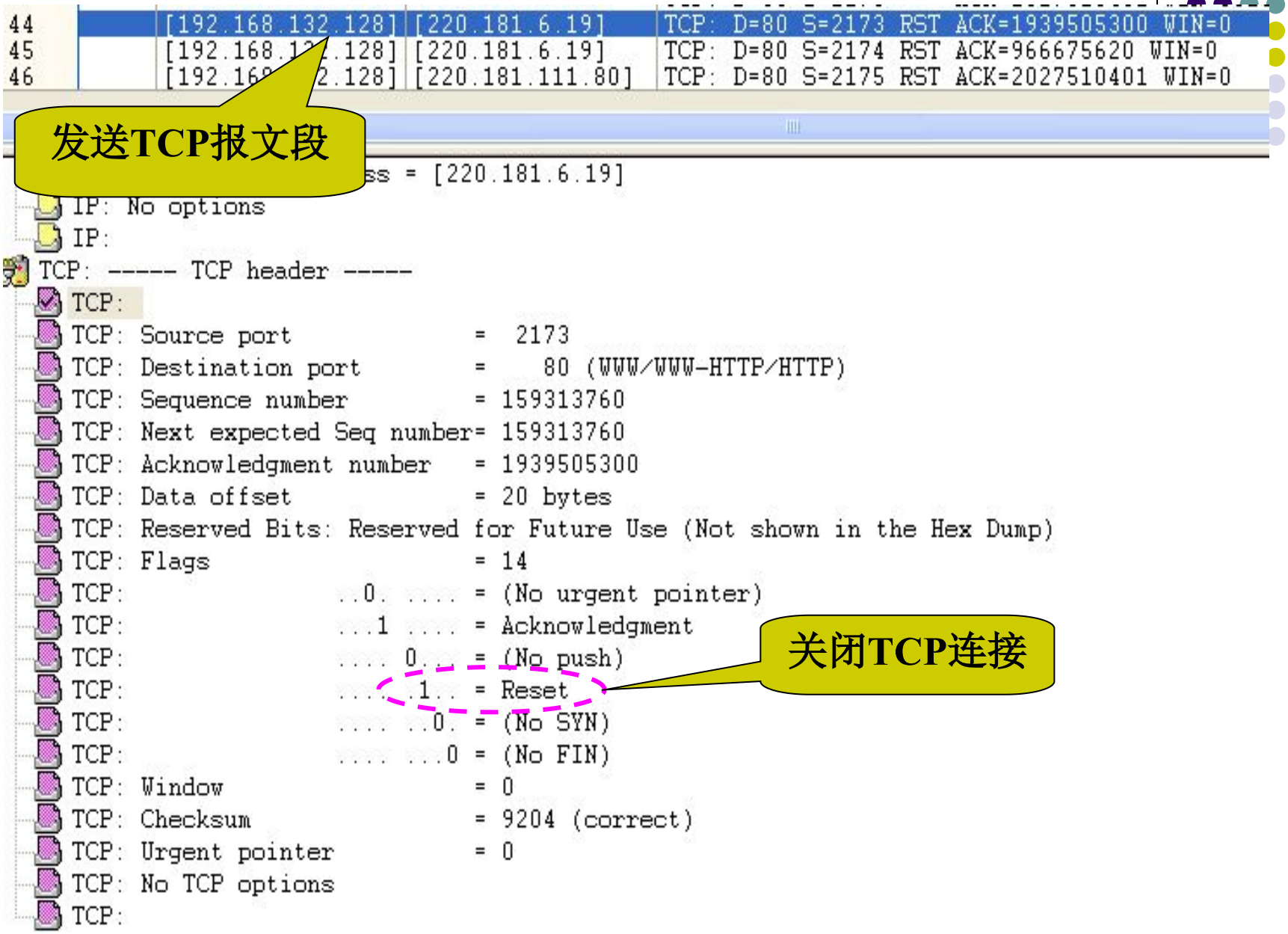

5 [220.181.6.19] [192.168.132.128] TCP: D=2173 S=80 ACK=159313155 WIN=64240
6 [220.181.6.19] [192.168.132.128] HTTP: R Port=2173 HTTP/1.1 Status=OK-2974 bytes of content
7 [220.181.6.19] [192.168.132.128] HTTP: Continuation of frame 6;1420 Bytes of data
8 [220.181.6.19] TCP: D=80 S=2173 ACK=1939501381 WIN=65535
服务器端HTTP响应
HTTP: 1: HTTP/1.1 200 OK
HTTP: 2: Date: Sun, 21 Nov 2010 15:27:38 GMT
HTTP: 3: Server: BWS/1.0
HTTP: 4: Content-Length: 2974
HTTP: 5: Content-Type: text/html;charset=gb2312
HTTP: 6: Cache-Control: private
HTTP: 7: Expires: Sun, 21 Nov 2010 15:27:38 GMT
HTTP: 8: Content-Encoding: gzip
HTTP: 9: Set-Cookie: BAIDUID=08993A0449B7800FFB1929ECECF8D454:FG=1; expires=Sun, 21-Nov-40 15:27:38 G
HTTP: 10: P3P: CP=" OTI DSP COR IVA OUR IND COM "
HTTP: 11: Connection: Keep-Alive
HTTP: 12:
HTTP: 13:
响应码：成功
主体部分数据量大小
文档类型及编码方案
空行，表明后面是主体部分


11 [192.168.132.128] [220.181.6.19] TCP: D=80 S=2173 ACK=1939502935 WIN=65535
12 [192.168.132.128] [220.181.6.19] HTTP: C Port=2173 GET /img/baidu_logo.gif HTTP/1.1
13 [220.181.6.19] [192.168.132.128] TCP: D=2173 S=80 ACK=159313461 WIN=64240
14 TCP: D=80 S=2174 SYN SEQ=3849941454 LEN=0 WIN=65535
获取网页图片
0. .... = (No urgent pointer)
TCP: ...1 .... = Acknowledgment
TCP: .... 1... = Push
TCP: .... .0.. = (No reset)
TCP: .... ..0. = (No SYN)
TCP: ....0 = (No FIN)
TCP: Window = 65535
TCP: Checksum = CA61 (correct)
TCP: Urgent pointer = 0
TCP: No TCP options
TCP: [306 Bytes of data]
TCP:
HTTP: ----- Hypertext Transfer Protocol -----
HTTP:
HTTP: 1: GET /img/baidu_logo.gif HTTP/1.1
HTTP: 2: Accept: */*
HTTP: 3: Referer: http://www.baidu.com/
HTTP: 4: Accept-Language: zh-cn
HTTP: 5: Accept-Encoding: gzip, deflate
HTTP: 6: User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
HTTP: 7: Host: www.baidu.com
HTTP: 8: Connection: Keep-Alive
HTTP: 9: Cookie: BAIDUID=08993A0449B7800FFB1929ECECF8D454:FG=1
HTTP: 10:
请求命令及请求的文件名


12 [192.168.132.128] [220.181.6.19] HTTP: C Port=2173 GET /img/baidu_logo.gif HTTP/1.1
13 [220.181.6.19] [192.168.132.128] TCP: D=2173 S=80 ACK=159313461 WIN=64240
14 [192.168.132.128] [220.181.6.19] TCP: D=80 S=2174 SYN SEQ=3849941454 LEN=0 WIN=65535
15 [220.181.6.19] [192.168.132.128] HTTP: R Port=2173 HTTP/1.1 Status=OK-1489 bytes of content
HTTP响应
HTTP: Vector Offset Length Frame
HTTP: 0 0x0036 312 15
HTTP: 1 0x0036 1420 16
HTTP: 2 0x0036 69 18
HTTP: 1801 bytes of re-assembled data.
HTTP: ----- Hypertext Transfer Protocol -----
HTTP: 1: HTTP/1.1 200 OK
HTTP: 2: Date: Sun, 21 Nov 2010 15:27:38 GMT
HTTP: 3: Server: Apache
HTTP: 4: Last-Modified: Tue, 29 Jul 2008 16:00:00 GMT
HTTP: 5: ETag: "5d1-4532bbb6ca000"
HTTP: 6: Accept-Ranges: bytes
HTTP: 7: Content-Length: 1489
HTTP: 8: Cache-Control: max-age=315360000
HTTP: 9: Expires: Wed, 18 Nov 2020 15:27:38 GMT
HTTP: 10: Connection: Keep-Alive
HTTP: 11: Content-Type: image/gif
HTTP: 12:
HTTP: 13: Content: (1489 bytes of data)
HTTP:
HTTP响应：成功
主体部分数据量大小
发送的文档类型


百度一下，你就知道 - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
收藏夹
地址(D)
停止
转到
链接
搜索设置 | 登录
Bai du 百度
新闻 网页 贴吧 知道 MP3 图片 视频 地图
百度一下
输入法
空间 百科 hao123 | 更多>>
把百度设为主页
加入百度推广 | 搜索风云榜 | 关于百度 | About Baidu
©2010 Baidu 使用百度前必读 京ICP证030173号
Internet

客户端停止连接


44 [192.168.132.128] [220.181.6.19] TCP: D=80 S=2173 RST ACK=1939505300 WIN=0
45 [192.168.132.128] [220.181.6.19] TCP: D=80 S=2174 RST ACK=966675620 WIN=0
46 [192.168.132.128] [220.181.111.80] TCP: D=80 S=2175 RST ACK=2027510401 WIN=0
发送TCP报文段
ss = [220.181.6.19]
IP: No options
IP:
TCP: ----- TCP header -----
TCP:
TCP: Source port = 2173
TCP: Destination port = 80 (WWW/WWW-HTTP/HTTP)
TCP: Sequence number = 159313760
TCP: Next expected Seq number= 159313760
TCP: Acknowledgment number = 1939505300
TCP: Data offset = 20 bytes
TCP: Reserved Bits: Reserved for Future Use (Not shown in the Hex Dump)
TCP: Flags = 14
TCP: ..0. .... = (No urgent pointer)
TCP: ...1 .... = Acknowledgment
TCP: .... 0... = (No push)
TCP: .... .1.. = Reset
TCP: .... ..0. = (No SYN)
TCP: ....0 = (No FIN)
TCP: Window = 0
TCP: Checksum = 9204 (correct)
TCP: Urgent pointer = 0
TCP: No TCP options
TCP:
关闭TCP连接