第7章 网络应用(2)

教材目录

- 域名服务（7.1）
  - 主机名和域名
  - 域名注册和管理
  - 域名解析服务
  - Internet域名和URL
- 传统应用（7.2）
  - **Email**
  - **FTP**
  - **WWW**

主要内容

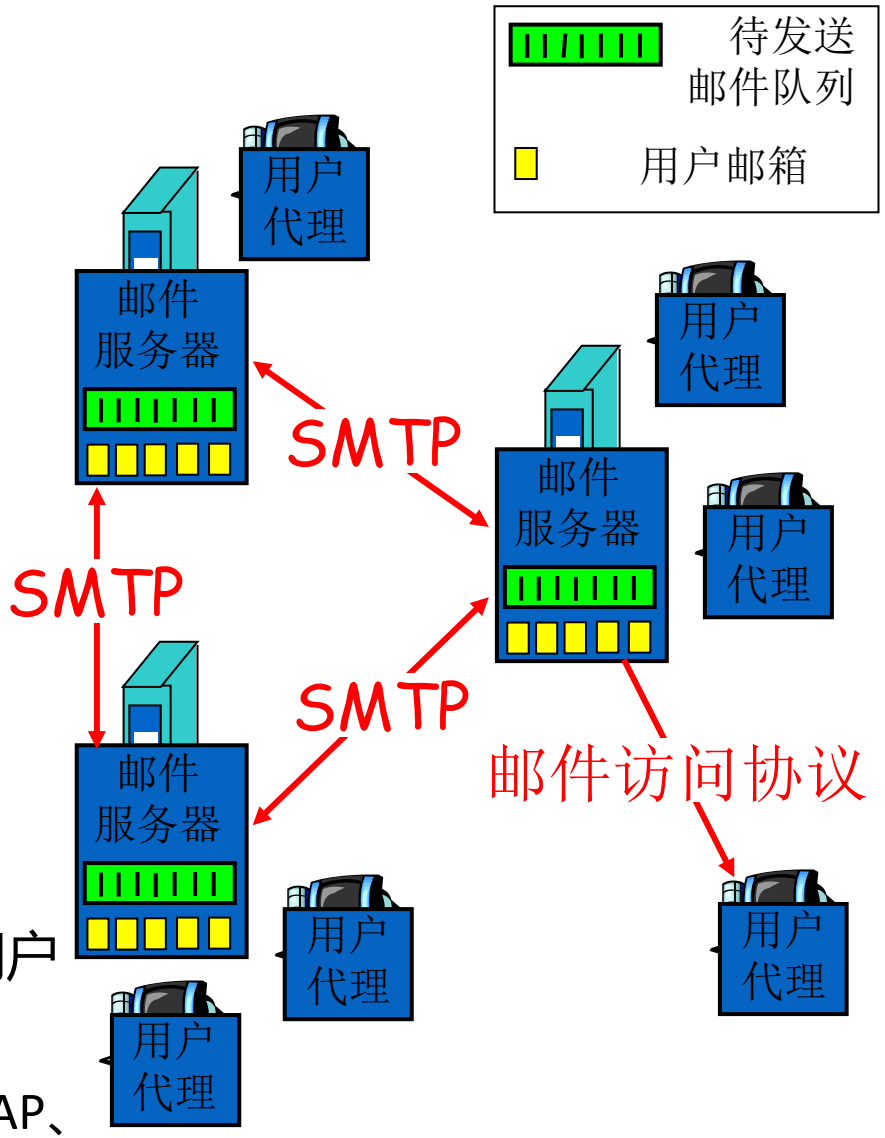
- E-mail
  - 邮件格式：头部和MIME
  - 邮件传输：SMTP
    - SMTP与DNS
    - SMTP命令与响应
    - 传递邮件：Received/Return-Path
    - Anti-Spam
  - 邮件访问：POP3、IMAP和Web-email
- FTP
- Web和HTTP
  - Web概述
  - URI/URL: URI格式和百分号编码
  - HTTP协议：持续连接/HTTP请求和响应/块编码/条件GET/Cookie和认证/Proxy

传统应用：E-mail(Electronic mail)

- 电子邮件是最早的TCP/IP应用之一
- E-mail地址格式：user@domain
  - user为本地解释，可能大小写相关
  - domain用于查询DNS，大小写无关

四个主要部件

- **用户代理(Mail User Agent)**：即“电子邮件应用程序"
  - 撰写、阅读邮件
  - Outlook,Thunderbird, iPhone/Android邮件客户方
- **邮件服务器(Mail Transfer Agent)**：
  - 邮箱：送达给(本地)用户的到来邮件
  - 邮件队列：要发送给其他用户的邮件
- **SMTP协议**：用户代理到邮件服务器(mail submission, 一般要求用户认证)以及邮件服务器之间传递邮件
- **邮件访问协议**：获得邮件服务器上的邮箱中的邮件，POP3、IMAP、Web Email等



邮件格式

每行（不包括CRLF）字符个数建议小于等于78，最长不能超过998

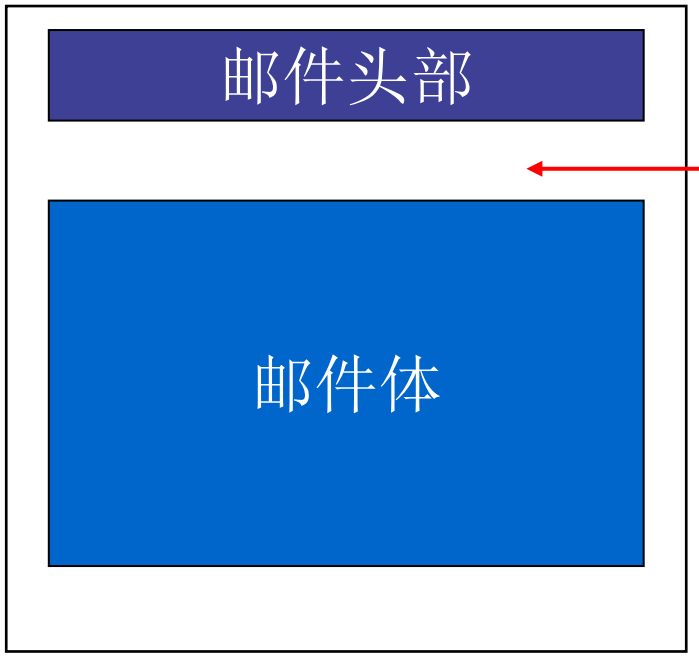
- RFC 822/2822/**5322**定义了邮件格式

头部： 7-bit U.S. ASCII 可打印字符(ASCII码32-126)

- 多行文本，每行包含type和value，中间以冒号:分隔
- 注意**冒号前**不能有空格
- 定义了许多常用的邮件头部类型(From:、To:、CC:、Reply-To:、Subject:)，大小写无关
- 如： “To: dlmao@fudan.edu.cn” and “Subject: Schedule”

邮件体： 7-bit U.S. ASCII 字符（ASCII码1-127）

- 头部和邮件体之间包含一个空行 <CRLF><CRLF> \r\n\r\n
- 邮件内容没有任何其他限制




```
From: John Doe <jdoe@machine.example>
To: Mary Smith <mary@example.net>
Cc: team: <tom@foo.com>, <bob@foo.com>;
Subject: Saying Hello
Date: Fri, 21 Nov 1997 09:55:06 -0600
Message-ID: <1234@local.machine.example>

This is a message just to say hello.
So, "Hello".
```

头部中的邮件地址：

- 单个邮件地址：user@domain或<user@domain>或"Full Name" <user@domain>
- 多个邮件地址：以逗号分割
- 拓展：邮件组地址:
  - 格式为： 组名:0个或多个邮件地址;

  Undisclosed recipients:;

  team: a@foo.com, b@foo.com;

主要头部

M: Mandatory
O: Optional
N: Normally present
S: should be present

用户(邮件服务器)也可以自定义自己的头部，一般以X-开头

组	头部名	必须/可选	出现次数	描述
来源日期	Date	M	1	邮件撰写完毕准备发送的时间
来源	From	M	1	邮件的源(一个或多个发送者)
	Sender	O	1	邮件的实际发送者
	Reply-To	O	1	在回信时应该回送给谁(一或多个地址）。如果没有，则来自于From头部
目的	To	N	1	邮件的接收者，可以多个，之间以逗号隔开
	Cc	O	1	邮件的(抄送)接收者，可以多个
	Bcc	O	1	邮件的(密送)接收者，该头部在其他接收者处被移走
消息ID	Message-ID:	S	1	唯一标识该邮件，在发送时生成 <xxx@xxx>
	In-Reply-To:	O	1	在回信时一般会包括，指出所回信件的ID(一个或多个)
	References	O	1	包含了多封信件的ID，用于关联多个邮件，将其组织为邮件会话
信息	Subject	N	1	信件主题
拓展：跟踪	Received:	MTA	无限制	类似邮戳功能，纪录经过的服务器信息以及相应的时间(分号隔开)
	Return-Path	MTA	1	信件最后递交（离开SMTP环境）时在最前面保存该信件递交过程中通过MAIL FROM传递的反向路径，格式为<addr>。如果信件中已有头部，则移走并添加

邮件服务器递交过程中生成

撰写邮件时有Bcc，时，到达其他接收者时该头部一般被移走

绿颜色为MIME头部

CRLFCRLF表示下面为**消息体**

```
Received: by ajax-webmail-app1 (Coremail) ; Thu, 28 Sep 2017 14:35:35 +0800
 (GMT+08:00)
X-Originating-IP: [10.230.3.18]
Date: Thu, 28 Sep 2017 14:35:35 +0800 (GMT+08:00)
X-CM-HeaderCharset: UTF-8
From: "Mao Dilin" <dlmao@fudan.edu.cn>
To: dlmao@fudan.edu.cn
Cc: dilin.mao@outlook.com
Subject: =?UTF-8?B?5rWL6K+V6YKu5Lu2?=
X-Priority: 3
X-Mailer: Coremail Webmail Server Version XT3.0.8 dev build
 20160401(82936.8581) Copyright (c) 2002-2017 www.mailtech.cn fudan
X-SendMailWithSms: false
Content-Transfer-Encoding: 7bit
X-CM-CTRLDATA: +PeSDGZvb3Rlcl90eHQ9NDE6MTA=
Content-Type: text/plain; charset=UTF-8
MIME-Version: 1.0
Message-ID: <58e9e7b0.25eb4.15ec7329a32.Coremail.dlmao@fudan.edu.cn>
X-Coremail-Locale: zh_CN
X-CM-TRANSID:AQUFCkBJ8kU4mMxZqliNAA--.14714W
X-CM-SenderInfo: 5gopt0w6ixvtvqohv3gofq/1tbiAgMRAFKpx0+8BgABsY
X-Coremail-Antispam: 1Ur529EdanIXcx71UUUUU7IcSsGvfJ3iIAIbVAYjsxI4VWUJw
        CS07vEb4IE77IF4wCS07vE1I0E4x80FVAKz4kxMIAIbVAFxVCaYxvI4VCIwcAKzIAtYxBI
        daVFxhVjvjDU=

Just a test email, please ignore it.
```

<header name>: <head value>

如果一行太长时，可以多行，后面的行以空格或Tab开始

<header name>: <header value part1>

<white-space> <header value part2>

<white-space><header value part3>

- Received头部的基本格式

Received: from <sending host>
by <receiving host>
with <mail protocol>
id <msg-id>;
<datetime received>

MIME(Multipurpose Internet Mail Extension)

- RFC822邮件格式要求7-bit ASCII字符，需要发送附件或非英文文本邮件时，不能胜任
- RFC 2045, 2046, 2047引入了MIME，在设计MIME时，要求
  - 仍然采用原有的RFC 822邮件格式，仍然采用原有的邮件传输(SMTP)和访问协议
  - 在撰写邮件时将**非ASCII文本**进行编码后转变为ASCII文本，在邮件阅读时转换回来
  - 支持**多个附件、消息文本整合**在一起，即多个部分(body)组合在一起成为一个邮件(message)
- 对于原有RFC 822邮件格式扩展，引入了多个MIME头部
  - MIME实体(entity): 由头部以及消息体组成的一个相对完整的部分
  - 整个邮件是一个MIME Entity，通过MIME头部描述整个消息的组织和编码等信息
  - 在邮件为multipart类型时，即多个子消息组成一个大消息时
    - 每个子消息是一个MIME Entity
    - 多个MIME Entity组成一个更大的MIME Entity

MIME 头部

```
Content-Type: text/plain; charset=UTF-8
Content-Type: image/jpeg; name="fudan.jpg"
```

- MIME头部也允许有（多个）参数部分，以分号开始，格式为 ;name=value

头部	描述
MIME-Version	版本号，目前1.0，只在整个消息头部中出现
Content-Type	内容的类型，type/subtype, 缺省text/plain; charset=us-ascii
Content-Transfer-Encoding	编码方式，可选，缺省为ASCII
Content-ID	可选，采用与Message-ID相同格式，可在同一个消息中通过HTML的cid URL引用本内容
Content-Description	可选，内容的简短描述
拓展：Content-Disposition	RFC 2183, 对内容(附件)怎样处理，inline表示内嵌显示，而attachment表示另外保存，如 attachment; filename="http.pptx"

type	subtype	type	subtype
text	plain,html,enriched,css	application	**octet-stream**,zip,pdf,msword, vnd.ms-powerpoint…
image	jpeg, png, gif	multipart	由多个entity组成， mixed, alternative, parallel, digest
audio	basic,mp3	message	封装了一个消息，可以为rfc822, partial， external-body
video	mpeg,mp4		

- text/xxx: 参数charset说明文本属于哪个字符集(编码方式）中的字符

MIME: 类型multipart

子类型

- mixed:各个独立的子部分组合在一起，比如附件
- alternative:同一内容的不同表示，比如文本和HTML版本
- parallel: 各个子部分应该同时使用，比如音频和视频
- digest:多个邮件组成，一般用于邮件列表（mailing list）
- 拓展 related：把相关的子部分组合在一起，类似于mixed，只是某个子部分为根（第一个或者start参数描述），其负责组织其他子部分。比如内嵌多个图片的HTML页面

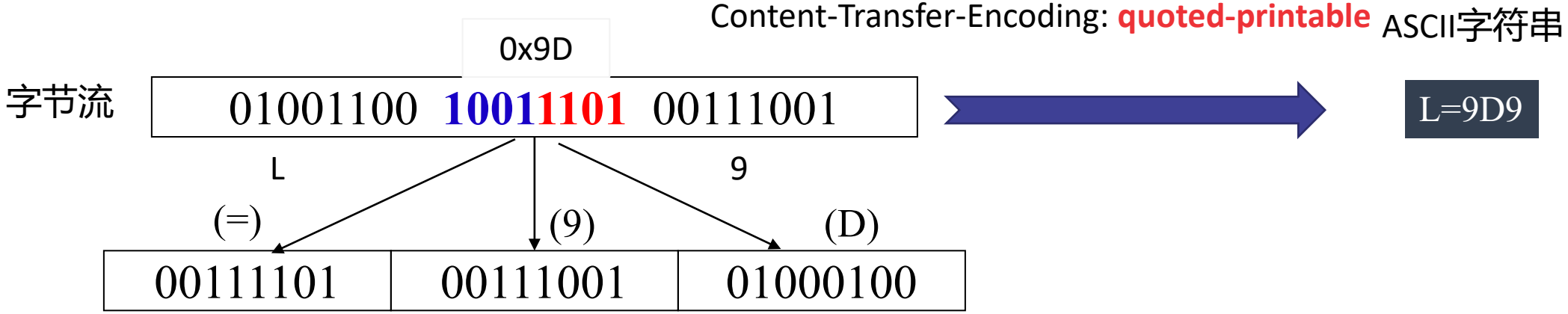
参数boundary给出了各个子部分的边界

- 分界行：**两个连字符开始，然后是boundary参数给出的字符串，最后可包含空格（可选）**
- Boundary参数随机选择，保证MIME Entity中不会出现分界行，或者某行最前面为分界行的内容
- multipart类型的body部分的结束：**两个连字符开始，然后是boundary参数给出的字符串，最后是两个连字符**

```
Mime-Version: 1.0
Content-Type: multipart/mixed;
     boundary="==001_Dragon614750641803_=="

--==001_Dragon614750641803_==
Content-Type: text/plain;charset="gb2312"
Content-Transfer-Encoding: base64


1eLKx9K7t+Ky4srU0…….

--==001_Dragon614750641803_==
Content-Type: application/octet-stream; name="test.doc"
Content-Transfer-Encoding: base64

UEsDBBQABgAIAAAAIQ……

--==001_Dragon614750641803_==--
```

Content-Transfer-Encoding

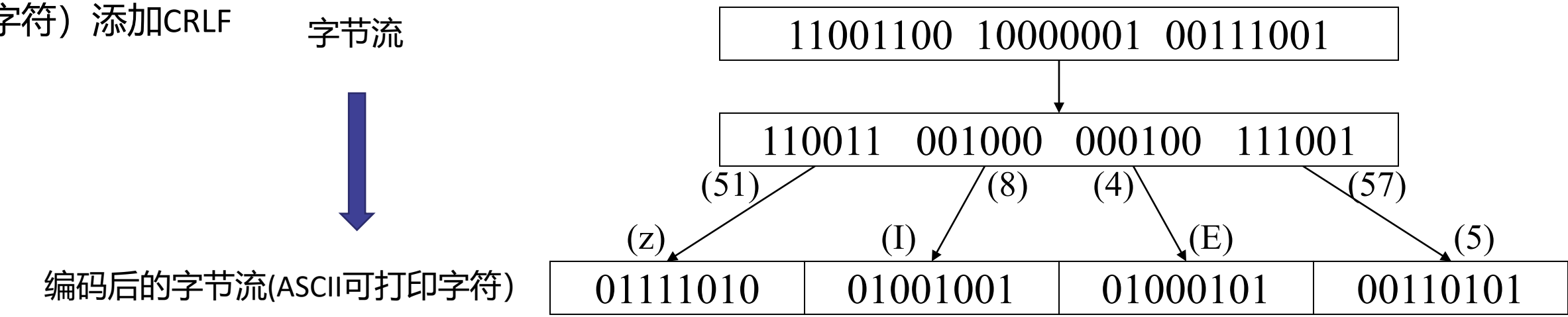
- 7bit： 缺省，即RFC 822格式，每行不超过998个字符，行分隔符CRLF，不允许ASCII码超过127的字符
- 8bit：类似于7bit，每行不超过998个字符，字符不再有7bit的限制，但不能为NULL字符（即值为0）
- binary：二进制方式，表示每个字符没有限制，也没有行的概念
- base64：下一页PPT
- quoted-printable：带引用可打印
  - 可打印字符：ASCII码从32到126，共95个字符，包括空格、数字、大小写英文字母和标点符号
  - 用于只有少量的非ASCII可打印字符的情况
  - 非ASCII可打印字符转换成3个字符： “=”+该字符的ASCII码(采用16进制大写描述)
  - 等于号(=，ASCII码0011 1101)转换为=3D



Content-Transfer-Encoding: Base64

编码效率=$\frac{\text{有效的数据}}{\text{实际传输的数据}}=\frac{3}{4}$

- 基本思想： 要编码的内容以3个字节为单位进行编码，转换为4个ASCII可打印字符
- 3个字节(24个比特)以**每6比特为一组**变成4组。每6比特对应的64个值依次用字符A~Z、a~z、0~9、+和/表示
- 在以24比特组合时最后可能只剩下一个字节或者两个字节
  - 在后面（右边)填上足够的比特0，以便最后凑成6比特，再转换为ASCII可打印字符
  - 最后再**添加多个等号**。剩下一个字节时添加2个等号，两个字节时添加1个等号
- Base64编码后的邮件体只用了65个不同字符(包括用于填充的=)，从而根据需要可以自由地（每76字符）添加CRLF




```
00111001            →  001110  010000          →  OQ  →  OQ==
10000001  00111001  →  100000  010011  100100  →  gTk →  gTk=
```

邮件头部的编码：RFC2047

拓展的内容

邮件头部： <Name>: <Value>

- Name为US-ASCII可打印字符
- 头部Value部分包含非ASCII字符时，与消息体类似，也需要转换为ASCII字符：
  - 需要进行转换的部分按照如下格式描述 =?charset?encoding?text?=
  - 指出采用的字符集(charset)和编码方式(encoding)，text为编码后的字符（ASCII可打印字符，不包括空格）
  - 解码时，发现某个token中有部分内容以=?开始，以?=结束，中间有2个?，分别取得charset和encoding，然后按照encoding进行解码
  - encoding为一个字符，如果为B表示采用Base64，如果为Q表示采用Q-encoding
    - 编码时：原来的属于字符集charset的字符串看成字节流，采用encoding方式编码，最终的结果为text
    - 解码时：text按照encoding方式解码，然后将其看成字符集charset中的字符组成的字符串
    - Q-encoding：类似于quoted-printable
      - ?编码为=3F （=编码为=3D，与quoted-printable一样）
      - 空格用下划线(_)表示，或者采用=20表示
    - 注意：？不会在Q-encoding编码后的字节流出现，也不会在base64编码后的字节流中出现

Subject: =?UTF-8?B?5rWL6K+V6YKu5Lu2?=

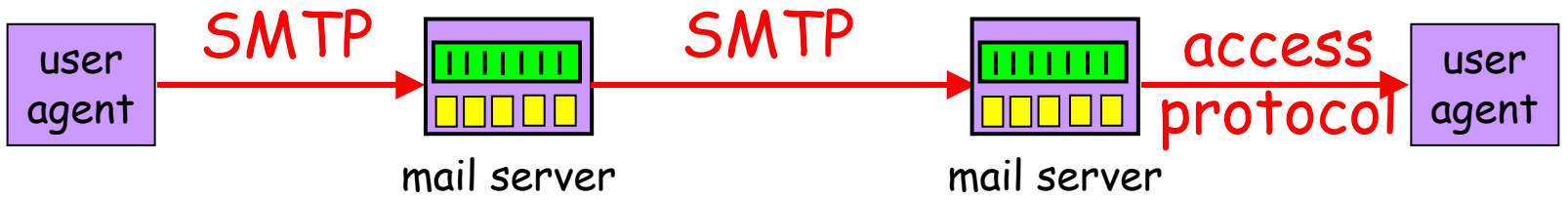
Subject: 测试邮件

SMTP(Simple Mail Transfer Protocol)

- RFC 821/2821/5321给出了SMTP协议
- 基于C/S模型，建立在可靠的数据传输协议TCP之上，端口号25
- SMTP是一种Push协议
  - 邮件从用户逐步推送经过多个邮件服务器(MTA，Mail Transfer Agent)，直到到达最终用户所在的邮件服务器
  - 每一跳采用SMTP协议可靠递交邮件
  - 基于**7-bit ASCII文本**的请求-响应协议：
    - Client发送请求: EHLO、MAIL FROM、RCPT TO、DATA、QUIT等
    - Server发送响应，状态码+文字说明，之间以空格或-隔开
    - 状态码：3个数字，表示执行是否成功，其中第一个数字：
      - 2表示成功
      - 3表示成功，但是需要Client继续发送，比如收到DATA请求时的状态码
      - 4表示临时拒绝
      - 5表示永久拒绝

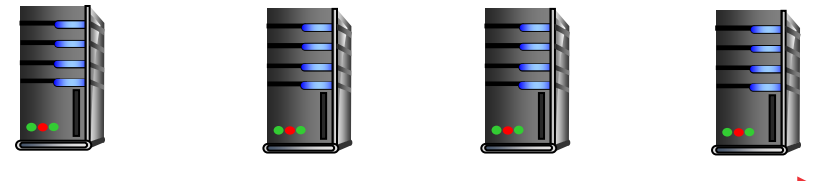
SMTP响应一般为一行，但也可支持多行：
- 每行以状态码-开始，后面是文字
- 最后一行以状态码开始，后面是空格，再后面为文字

```
C: EHLO ubuntu
S: 250-mail
   250-AUTH LOGIN PLAIN
   250 OK
C: MAIL FROM:<xiucao@fudan.edu.cn>
S: 250 xiucao@fudan.edu.cn... Sender ok
   ...
C: DATA
S: 354 Enter mail, end with "." on a line by
C: Blah blah blah.....
```



尽管途中可经过多个邮件服务器。大部分邮件途中一般只会经过1个或**2个邮件服务器**，即发送方所在的邮件服务器以及接收方所在的邮件服务器

SMTP与DNS



发给user@domain的邮件应该经过哪些邮件服务器?

- 查询domain的MX纪录:格式为（优先级 邮件服务器）
- 如果没有MX纪录，查询domain的A或AAAA纪录，相当于MX纪录(0 ipaddr)
- 优先级取值越小表示越靠近最终用户邮箱，优先级更高
- 将邮件**逐步推送到最终的用户邮箱**：
  - 发送邮件时优先尝试优先级取值最小（优先级最高）的邮件服务器，如果当前无法连接时尝试第二小的邮件服务器...
  - 不会将邮件再推送到远离用户邮箱的方向：即通过DNS获得的MX记录列表中移除那些优先级取值大于等于当前邮件服务器的优先级的邮件服务器

```
gmail.com.  MX    10 alt1.gmail-smtp-in.l.google.com.
gmail.com.  MX    5 gmail-smtp-in.l.google.com.
gmail.com.  MX    30 alt3.gmail-smtp-in.l.google.com.
gmail.com.  MX    40 alt4.gmail-smtp-in.l.google.com.
gmail.com.  MX    20 alt2.gmail-smtp-in.l.google.com.
```

- 发送给@gmail.com的邮件优先递交给gmail-smtp，如果其暂时无法连接，则递交给alt1/alt2/alt3/alt4
- 如果邮件在alt3，接下来只会尝试递交给优先级相比它的优先级值更小的邮件服务器，即只会尝试递交给gmail/alt1/alt2
- 在实践中，许多domain一般只会使用一个邮件服务器，Internet上的邮件服务器在无法递交时会缓存起来，一般允许保留最多5天

SMTP命令

命令（大小写无关）	说明
HELO domain	打招呼，现在被EHELO代替
EHLO domain	打招呼，RFC2821定义，通知发送者的domain或者其IP地址。EHLO响应中给出了所支持的**SMTP扩展。**采用扩展的SMTP称为**ESMTP**(Extended SMTP)
MAIL FROM:<addr>	发送者邮件地址。**注意冒号前后没有空格**
RCPT TO:<addr>	接收方邮件地址，可发送多个RCPT TO命令
DATA	接下来是邮件部分，直到只有一个.（dot)的行结束，即**CRLF.CRLF出**现。如果邮件部分的某行的**行首字符为点. 则替代以两个点**..。接收方发现**行首为点且该行后面还有字符，则去掉第一个点**
QUIT	退出

```
C：EHLO ubuntu
S：250-mail
   250-AUTH LOGIN PLAIN
   250 OK
C：MAIL FROM:<xiucao@fudan.edu.cn>
S：250 xiucao@fudan.edu.cn... Sender ok
C：RCPT TO:<wang@pku.edu.cn>
S：250 wang@pku.edu.cn... Recipient ok
C：RCPT TO:<zhang@tsinghua.edu.cn>
S：250 zhang@tsinghua.edu.cn... Recipient ok
C：DATA
S：354 Enter mail，end with "." on a line by itself
C：Blah blah blah……
C：……
C：.
S：250 Ok
C：QUIT
S：221 fudan.edu.cn closing connection
```

SNMP响应一般为一行，但也可支持多行：

- 每行以状态码-开始，后面是文字
- 最后一行以状态码开始，后面是空格，再后面文字

```
250-mail
250-AUTH=LOGIN PLAIN
250-STARTTLS
250-SMTPUTF8
250 8BITMIME
```

- 拓展：服务器收到CRLF.CRLF后会暂存收到的邮件，同时发送250 Ok
- 客户方收到250 OK，表示邮件发送成功
- 客户方没有收到250 OK，而连接此时断开，会假设收到451响应，即会认为邮件发送不成功，以后会再次发送，从而可能出现重复邮件

SMTP：信封和邮件

拓展的内容

- 大部分用户代理(UA）会从用户撰写的邮件的相应头部(From/To/Cc/Bcc)获得信息，通过MAIL FROM/RCPT TO命令来发送邮件
- 邮件服务器通过SMTP收到邮件时：
  - MAIL FROM和RCPT TO命令产生信封部分
    - RCPT TO的地址决定了要递交给的用户邮箱
    - MAIL FROM的地址：在邮件递交出现问题时发送通知(Bounce)邮件到该地址
      - Bounce邮件的MAIL FROM地址为 <>，即不会对该邮件再发送Bounce邮件
      - 邮件组(mailing list)发送的邮件的MAIL FROM地址为 <>, 即递交出错时也不会发送Bounce邮件
      - 邮件离开SMTP环境时，可将MAIL FROM地址保存在Return-Path头部
  - Data命令传递的是邮件部分
    - 邮件服务器会在邮件部分最开始插入**Received**头部, 打上邮戳
    - 途中每个邮件服务器都会添加一个Received头部, 可了解邮件经过的路径以及避免回路
  - SMTP协议并不要求Data部分必须严格按照RFC822格式, 它可以是普通的ASCII文本

Return-Path: <dlmao@fudan.edu.cn>
Received: from barracuda.fudan.edu.cn ([200
by mx.google.com with ESMTP id
ih4si31738720pab.37.2016.10.18.17.51.42
for <dilin.mao@gmail.com>;
Tue, 18 Oct 2016 17:51:42 -0700 (PDT)
Received-SPF: pass (google.com: domain of dlmao@fudan.edu.cn designates 2001:da8:8001:2:225:90ff:fed8:bb36 as permitted sender) client-ip=2001:da8:8001:2:225:90ff:fed8:bb36;
Received: from fudan.edu.cn ([61.129.42.33]) by barracuda.fudan.edu.cn with ESMTP id RS3ftwxAzlsG4KD7 for <dilin.mao@gmail.com>; Wed, 19 Oct 2016 08:51:40 +0800 (CST)
Received: by ajax-webmail-app3 (Coremail) ; Wed, 19 Oct 2016 08:49:30 +0800 (GMT+08:00)
Date: Wed, 19 Oct 2016 08:49:30 +0800 (GMT+08:00)
From: "Mao Dilin" <dlmao@fudan.edu.cn>
To: dilin.mao@gmail.com
Subject: hello from dlmao
Content-Type: multipart/mixed;
boundary="----=_Part_366479_1777208066.1476838170514"
MIME-Version: 1.0
Message-ID: <28a53653.1ac5e.157da6a1f92.Coremail.dlmao@fudan.edu.cn>

信封上的内容：

- Return-Path: 保存MAIL FROM地址
- Received: 类似于邮戳

------=_Part_366479_1777208066.1476838170514
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 7bit

Dear Dilin,

This is a test message, please ignore it.

Mao Dilin

------=_Part_366479_1777208066.1476838170514
Content-Type: text/plain; name="dummy.txt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="dummy.txt"

YWJjZGVmZ2hpamtsbW4=
------=_Part_366479_1777208066.1476838170514
Content-Type: image/png; name="project5.png"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="project5.png"

iVBORw0KGgoAAAANSUhEUgAAAVYAAAA9CAIAAAEOjuZvAAAAAXNSR0IArs4c6QAAAARnQU1BAACx
jwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAABDWSURBVHhe7Z1/TFX1G8fdYqPFgjRwiuScqUFz
------=_Part_366479_1777208066.1476838170514--

发送者认证：SMTP AUTH

- RFC 4954 SMTP Service Extension for Authentication
- Client发送EHLO，服务方的响应中会包括它所支持的扩展
- Client发送AUTH请求开始发送者认证过程
  - PLAIN认证：
    - Client发送AUTH PLAIN IDENTITY
    - IDENTITY为'authorization-id\0authentication-id\0passwd`经过BASE64编码后的内容
  - LOGIN认证：Client发送AUTH LOGIN，服务方发送响应，接下来发送用户名、密码
- STARTTLS：开启TLS协商过程以保护接下来的SMTP会话
  - 之后的AUTH LOGIN或AUTH PLAIN等都会通过TLS协议加密传输

拓展的内容

```
demo@mars:~$ telnet mail.fudan.edu.cn 25
Trying 61.129.42.10...
Connected to mail.fudan.edu.cn.
Escape character is '^]'.
220 fudan.edu.cn Anti-spam GT for Coremail System
ehlo [10.11.253.4]
250-mail
250-AUTH=LOGIN PLAIN
250-coremail
1Uxr2xKj7kG0xkI17xGrUDI0s8FY2U3Uj8Cz28x1UUUUU7I
c2I0Y2Ur7Q3t2UCa0xDrUUUUj
250-STARTTLS
250-SMTPUTF8
250 8BITMIME
AUTH LOGIN
334 dXNlcm5hbWU6
ZGxtYW8=
334 UGFzc3dvcmQ6
xxxxxxx   password here
235 Authentication successful
```

Username: 的base64编码

Password: 的base64编码

上述例子，密码(base64编码后)明文传递。在AUTH LOGIN之前首先执行STARTTLS：

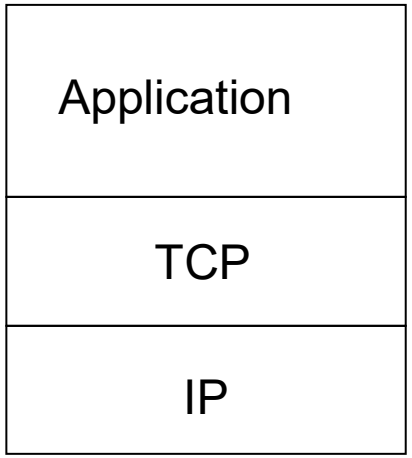
```
STARTTLS
220 Ready to start TLS
```

接下来在当前的TCP连接上进行TLS协商

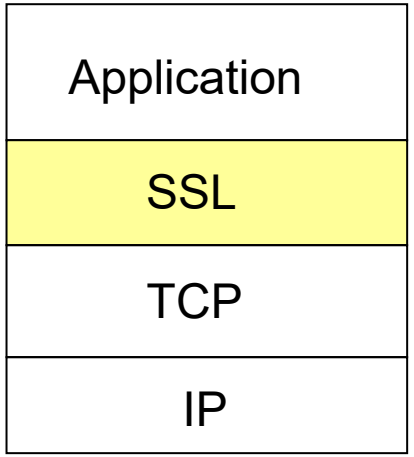
保护TCP连接

拓展的内容

- 传统的应用层协议采用基于7bit ASCII字符集的请求响应协议，消息（包括口令）明文传递
- SSL和TLS：提供用户认证、消息完整性和消息加密支持
  - Secure Socket Layer： Netscape提出，目前的版本SSL 3.0，已不建议使用(RFC 7568)
  - Transport Layer Security: RFC 2246定义TLS1.0，在SSL 3.0的基础上进一步改进。RFC 5246定义了TLS1.2
  - 2018年9月RFC 8446定义了TLS1.3
  - 人们提起SSL，现在一般指TLS
- 隐式的Application over TLS：**仍然采用原有的应用层协议**
  - 首先连接到**Secure Application端口**，采用TLS保护连接
  - 在采用TLS保护的安全连接上采用Application协议交互
  - 整个Application协议都被保护
- 显式的Application over TLS：**对原有的应用层协议扩展**
  - 首先连接到**原有的Application端口**
  - 对原有应用层协议进行扩展，发送请求(比如STARTTLS或AUTH TLS)开启TLS支持
  - 之后的Application协议交互受到保护



应用



应用 over TLS

隐式Application over TLS的端口：

- HTTPS: HTTP over SSL, port 443
- Secure SMTP: 端口465或者587
- IMAP4 over SSL: 端口993
- Secure POP3(SSL-POP): 端口995

防垃圾邮件(anti-spam)

- SMTP服务器增加认证机制，不再允许随意的邮件中继，一般会要求：
  - 只为其所在域的用户提供邮件传输服务
  - 要求验证发送者身份或者验证发送者的IP地址
  - 本地用户要通过该服务器发送邮件时需要验证发送者的确是其用户：SMTP AUTH
  - 在接收其他非本地用户发送过来的邮件时：
    - 验证对方的IP地址，是否的确为发送者所在域(MAIL FROM)的主机的IP地址？ Sender Policy Framework
    - 只允许接收目的地为本地邮箱的邮件，即RCPT TO为本地邮箱
- 基于DNS的黑名单列表: 验证SMTP client的IP地址以及域名(MAIL FROM)是否在相关机构维护的黑名单列表中

Sender Policy Framework (SPF)

拓展的内容

- RFC 4408，验证发送者IP在发送者domain(来自于MAIL FROM命令)的合法IP地址列表中
  - 如何获得发送者所在domain可能使用的IP地址列表?
  - 查询DNS的TXT或 SPF RR(RFC 4408建议使用TXT RR)

```
$ dig +noall +answer fudan.edu.cn  TXT
fudan.edu.cn.        3600    IN      TXT     "v=spf1 ip4:202.120.224.0/24 ip4:61.129.42.0/24
ip6:2001:da8:8001::/48 include:spf.icoremail.net  include:spf.mail.qq.com ~all"
```

上面说明:

- 可以从202.120.224.0/24, 61.129.42.0/24以及 IPv6地址(2001:...)以fudan邮箱的名义发信
- 可以从spf.mail.qq.com以及spf.icoremail.net配置的IP地址列表中以fudan邮箱名义发信

基于DNS的黑名单列表

拓展的内容

- 基于DNS的黑名单列表（DNSBL, DNS-based Blackhole List) RFC5782
- 目前常用的黑名单列表为zen.spamhaus.org等
- IP地址黑名单：假设foo.org维护的黑名单的域为dnsbl.foo.org
  - 将IP地址转换为域名, 比如202.120.224.10 → 10.224.120.202.dnsbl.foo.org
  - 查询其A RR
    - 如果DNS响应为NXDOMAIN（名字不存在），则说明不在黑名单里
    - 如果有回答，其对应的地址一般为127.0.0.2(实际上可通过不同的127.0.0.0/24来说明是哪种类型的原因出现在黑名单)，说明在黑名单里。相应的TXT RR给出了原因
  - 有多个组织维护了黑名单列表，其也是垃圾邮件发送者的诉讼对象

```
$ dig 10.224.120.202.zen.spamhaus.org
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 53548
```

- 域名黑名单(domain black list)：假设foo.org维护的域名黑名单的域为dbl.foo.org
  - 检查domain是否在黑名单，实际上是查domain.dbl.foo.org的A RR以及相应的TXT RR
  - 比如使用spamhaus.org的域名黑名单，可查询domain.dbl.spamhaus.org

```
$ dig +noall +answer 183.202.216.179.zen.spamhaus.org
183.202.216.179.zen.spamhaus.org. 0 IN  A   127.0.0.2
183.202.216.179.zen.spamhaus.org. 0 IN  A   127.0.0.11
```

DKIM(DomainKeys Identified Mail)

拓展的内容

- RFC6376，对于邮件的关键内容进行签名，邮件接收者通过该签名确认该邮件是否合法（发送者合法，邮件没有被篡改）
- 基于公开密钥算法：发送者所在域对邮件采用私钥签名，其他用户可通过对应的公钥验证签名
- 怎么获得邮件签名者的公钥？
  - 一个域可以有多个公钥私钥对，selector用于选择相应的公钥
  - 查询 <selector>._domainkey.domain的TXT记录

```
$ dig +noall +answer dkim._domainkey.fudan.edu.cn TXT
dkim._domainkey.fudan.edu.cn. 0 IN      TXT
"v=DKIM1;k=rsa;p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDlsjX4/ZMUq+pWiZq6P9r
ncYjobRiBlyXR1zTCmJgOaUzj11nD39v64xthJt1nBhbuYNf+7F5dgUFyS11fCkFoW0vZ4XJPi1FdhNST6
HVQkw/6g7KGkrUb3LAaYzuMR2+fzMGjF/7sp6hsrud1GX8Z9Utc5ACo7t409uUAhwaXkwIDAQAB"
```

- 邮件增加DKIM-Signature头部
  - 多个Tag，以分号隔开
  - d和s：决定了哪个私钥签名
  - h：在签名时包含了哪些头部
  - b：邮件的签名，base64编码
  - bh: 邮件body部分的签名

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=fudan.edu.cn; s=dkim;
h=Received:From:To:Subject:Date: Message-ID:MIME-Version:Content-Type:Thread-
Index: Content-Language; bh=LHjbC39b55m94/j2Oki1MMRCr4TxaaSjY7LGJ66/LHc =;
b=4idi0EZ0bEqbAD4j4tcp0zsZaQU9umlWuJ6VJwRpzEZ180Wt62U6ngiDuIb
Ramo3Jd4G114HOeNqmZeNSJ2bSO+APDta/v/DNQaRaZhAz0UMnoyQh6hzKSVU3Jq
Jkop78DzkhQJuMRy/sbabXC9Rrg9//lUBMzfizdDxZZsVsrQ=
```

邮件访问协议：POP3(Post Office Protocol Version 3

- RFC 1939，基于文本方式的请求/响应协议，TCP端口110
- 服务方响应：+OK或 -ERR

认证阶段：
- user: 登录用户名
- pass: 密码

事务阶段：
- STAT: 返回邮件个数和总长度
- list: 列出邮件，编号和长度
- retr: 读该编号对应的邮件
- dele: 给该编号对应邮件打上删除标记
- top: TOP <n> <m>: 列出第n条消息的前m行内容(不包括头部)
- uidl [n]: 列出所有邮件或者某个邮件的唯一ID

更新阶段：
- quit: 删除打过标记的邮件并退出

其他POP3请求
- CAPA： 列出服务方支持的能力
- STLS: 开启TLS协商，以后的会话被TLS保护

- POP3用户从服务器下载邮件到本地，在离线状态也可访问
- POP3用户在另外一台主机访问服务器时会重新下载服务器上的所有邮件
- 拓展：UIDL命令是POP3 Client允许保存邮件在POP3服务器的原因，通过比较本地的UIDL列表与服务器返回的UIDL列表，可以知道哪些邮件需要下载

```
uidl
+OK 7517 5043013623
1 1tbiAQ4FAFKp41U3cQAAsB
2 1tbiAg8FD1Kp2cmYcQAAmF
```

认证阶段

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

事务阶段

```
C: stat
S: +OK 2 1410
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
```

更新阶段

```
C: quit
S: +OK POP3 server signing off
```

邮件访问协议：IMAP和Web Mail

- POP3协议中
  - 邮件下载到本地，在本地组织管理，(也可保留在服务器，通过UIDL知道哪些已经在本地下载)
  - 用户在不同机器上的POP3客户方独自管理已经下载到本地的邮件
  - 一般下载邮件的所有部分，(后来也可采用TOP查看部分内容)
- Internet Mail Access Protocol [RFC 3501]:
  - 基于文本方式的请求/响应协议，TCP端口143， 993 （IMAP over TLS)
  - 邮件在服务方存储，在服务方管理，可以创建文件夹来组织文件
  - 允许不同主机对于**邮件有统一的访问体验**
  - 客户方可选择仅下载邮件的头部，打开时再下载邮件的内容
  - 尽管邮件会在服务方存储，客户方一般也会缓存阅读并下载的邮件
- Web Mail:
  - 无需邮件客户方，一个Web浏览器就可以了
  - 邮件的发送和阅读都采用**HTTP协议**

```
$ telnet mail 143
Trying 202.120.224.10...
Connected to mail.fudan.edu.cn.
Escape character is '^]'.
* OK Coremail System IMap Server Ready(fudan...)
000 capability
* CAPABILITY IMAP4rev1 STARTTLS ENABLE ...
000 OK CAPABILITY completed
001 LOGIN user password
001 OK LOGIN completed
002 NOOP
* 6704 EXISTS
* 2229 RECENT
002 OK NOOP completed
003 LIST "" *
* LIST () "/" "INBOX"
* LIST (\Drafts) "/" "Drafts"
* LIST (\Sent) "/" "Sent Items"
* LIST (\Trash) "/" "Trash"
* LIST (\Junk) "/" "Junk E-mail"
* LIST () "/" "Linux"
* LIST () "/" "network"
003 OK LIST Completed
```

- 第一条命令应该是starttls，保护接下来的imap会话
- 每个命令前面有一个tag
  - tag由client选择，保证递增即可
  - 响应可以有多行，前面的行以*开始，最后的行以tag开始
- 每个邮箱有一些系统定义的和用户添加的flag(如\Seen)
- 邮箱中邮件通过uid或顺序号标识

拓展的内容

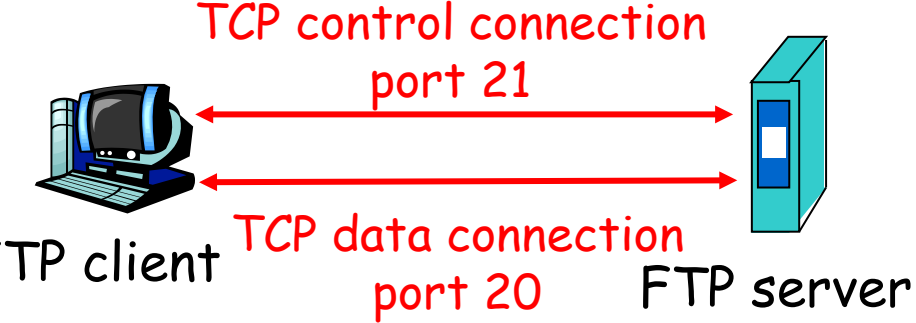
```
004 select inbox
* 3413 EXISTS
* 1 RECENT
* OK [UIDVALIDITY 1] UIDs valid
* FLAGS (\Answered \Seen \Deleted \Draft \Flagged)
* OK [PERMANENTFLAGS (\Answered \Seen \Deleted \Draft \Flagged)] Limited
004 OK [READ-WRITE] SELECT completed
005 search recent
* SEARCH 3411
005 OK SEARCH completed
006 fetch 3411 rfc822
* 3411 FETCH (RFC822 {2853}
.....
006 OK Fetch completed
007 close
007 OK CLOSE completed
008 select network
* 4 EXISTS
* 0 RECENT
....
008 OK [READ-WRITE] SELECT completed
009 logout
* BYE IMAP4rev1 Server logging out
009 OK LOGOUT completed
```

蓝色标记的是client发送的imap命令

主要内容

- E-mail
  - 邮件格式：头部和MIME
  - 邮件传输：SMTP
    - SMTP与DNS
    - SMTP命令与响应
    - 传递邮件：Received/Return-Path
    - Anti-Spam
  - 邮件访问：POP3、IMAP和Web-email
- FTP
- Web和HTTP
  - Web概述
  - URI/URL: URI格式和百分号编码
  - HTTP协议：持续连接/HTTP请求和响应/块编码/条件GET/Cookie和认证/Proxy

File Transfer Protocol (FTP) RFC 959



- Client连接到服务器的TCP 21端口→控制连接
  - 发送FTP命令，包括用户认证，列目录，下载和上传文件等
  - 服务器每收到一个要进行文件传输（包括列目录）的命令时开启一条新的TCP连接（数据连接）
    - 本次传输结束后关闭数据连接
  - 服务器维护状态信息：当前目录，用户身份等

FTP响应: 包括状态码和文本信息

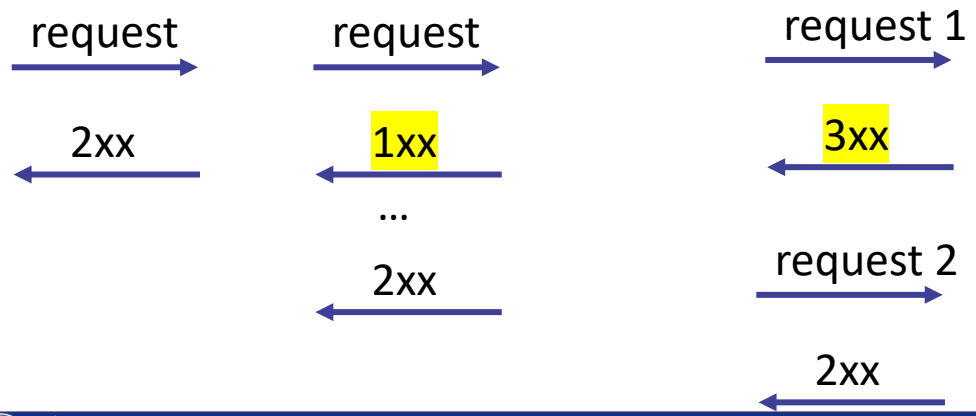
1xx正常操作，是一系列返回中的一个

2xx正常操作

3xx正常操作, 服务器等待进一步信息

4xx暂时错误操作

5xx永久错误操作



常用FTP命令:

- 通过控制连接发送，文本形式传输
- USER username
- PASS password
- CWD dirname
- LIST 列当前目录
- RETR filename **retrieves (gets) 下载file**
- STOR filename **stores (puts) 上传file**
- **REST offset** 指定后面的文件传输时要跳过的字节数，用于文件续传

- 331 Username OK, password required
- 230 Login successful
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

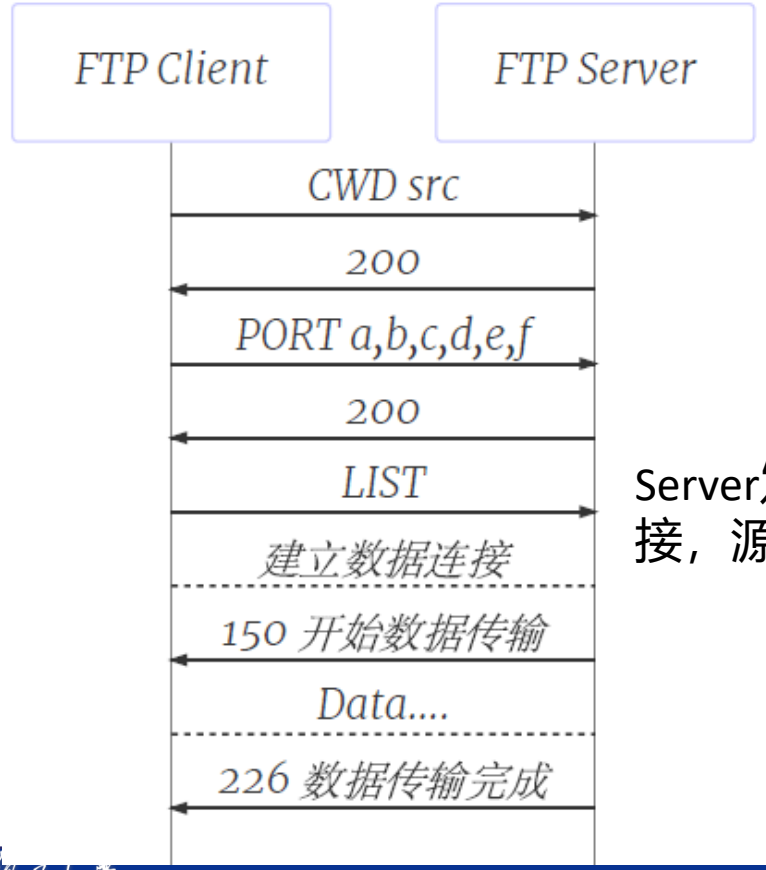
FTP数据传输：由谁来发起TCP连接？

Active方式：PORT h1,h2,h3,h4,p1,p2

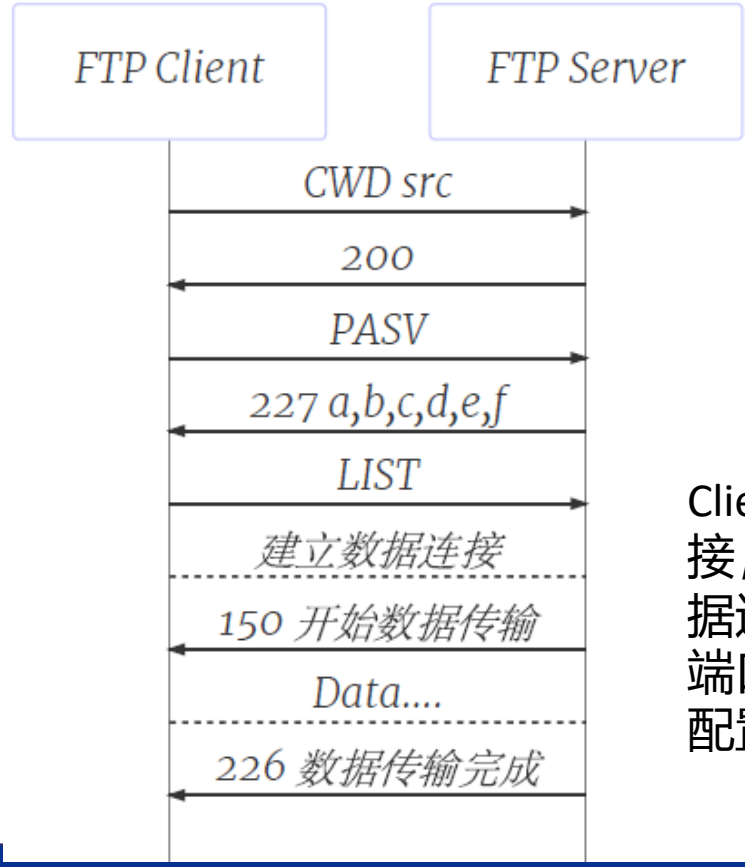
- 客户方在h1.h2.h3.h4上相应的端口监听（p1*256+p2）
- 服务方发起到port所指的目的地的数据连接（源端口20）

PASSIVE方式(PASV)：

- 服务方发送响应： h1,h2,h3,h4,p1,p2
- 主机（一般为服务方）h1.h2.h3.h4在相应端口监听，**客户方发起数据连接**



Server发起数据连接，源端口为20



Client发起数据连接，server方为数据连接所采用的端口范围一般可配置

FTP数据传输示例

```
ftp> ls
---> PORT 10,11,14,51,192,12
200 PORT command successful.
---> LIST
150 Opening ASCII mode data connection for file list.
drwxr-xr-x   3 ftp      ftp     4096 May 22 05:40 incoming
226 Transfer complete.

ftp> passive
Passive mode on.
ftp> ls
---> PASV
227 Entering Passive Mode (202,120,224,5,11,95).
---> LIST
150 Opening ASCII mode data connection for file list
drwxr-xr-x   3 ftp      ftp     4096 May 22 05:40 incoming
226 Transfer complete.
```

主要内容

- E-mail
  - 邮件格式：头部和MIME
  - 邮件传输：SMTP
    - SMTP与DNS
    - SMTP命令与响应
    - 传递邮件：Received/Return-Path
    - Anti-Spam
  - 邮件访问：POP3、IMAP和Web-email
- FTP
- Web和HTTP
  - Web概述
  - URI/URL: URI格式和百分号编码
  - HTTP协议：持续连接/HTTP请求和响应/块编码/条件GET/Cookie和认证/Proxy

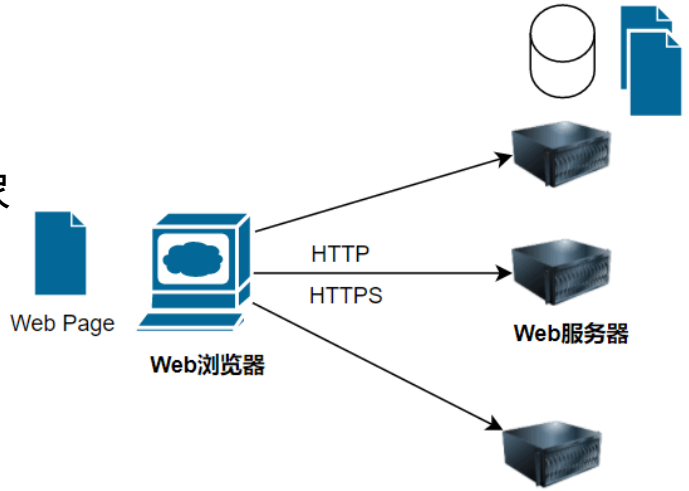
Web(World Wide Web)历史

- 1989年CERN（the European Center for Nuclear Research)的物理学家Time Berners-Lee提出了Web的概念，目的是通过超链接(hyperlink)将粒子物理实验中的产生的各种文档有机地组织在一起
- 1991年第一个Web服务器info.cern.ch
- 1993年University of Illinois的Marc Andreessen开发了第一个图形化的浏览器Mosaic
- 1994年CERN和MIT创立了W3C(World Wide Web Consortium, http://www.w3c.org)，推动Web的发展、协议的标准化
- 1994年Marc Andreessen创办了Netscape公司
- 1994年，启动Opera研究项目
- 1996年微软开始开发浏览器，在第一次浏览器大战中，微软战胜Netscape
- 互联网发展的时代来临。1994年Amazon成立，1995年eBay成立，1998年Google成立，2004年Facebook成立
- 1998年 Netscape成立Mozilla基金会，发起浏览器开源项目
- 2003年 Apple 发布Safari
- 2004年 Mozilla发布**开源浏览器Firefox**，开启第二次浏览器大战
- 2005年，Apple开源浏览器内核Webkit， 2008年，Google创建以Webkit为内核的**开源浏览器Chromium**，同时也发布了浏览器Chrome
- 2013年，Google因为和Apple出现分歧，创建Webkit的分支-Blink
- 2015年，微软放弃IE，发布Microsoft Edge

Web体系结构

Fudan University

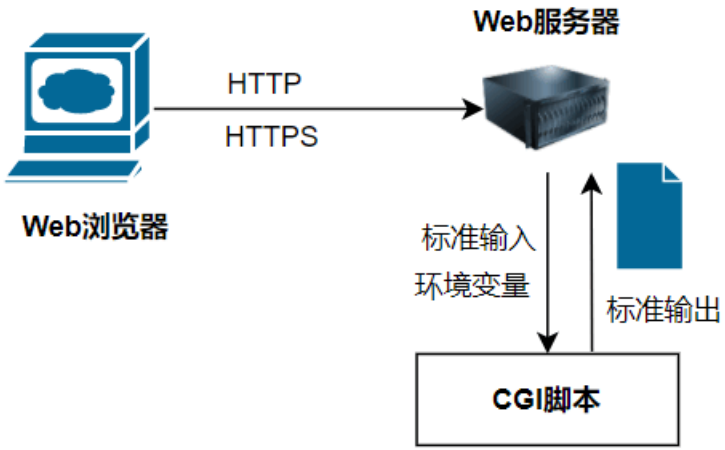
- Web上有各种资源（也称为Web对象）：HTML文档、图像、视频、音频和脚本等
- 可以通过Web Page(Web页面、网页）的形式将多个资源组织在一起
- 各个Web Page通过**超链接(hyperlink)**联系在一起，称为超文本(Hypertext)
  - 超链接也可以将Web页面与其他图像、音视频、文件等关联起来
- Web Page一般采用**HTML 语言**(Hypertext Markup Language，超文本标记语言) 描述
  - 通过Tag(标签）标记要显示的网页中的各个部分，包括文本、列表、表格、图片等信息
  - **层叠样式表CSS**(Cascading Style Sheets)描述HTML文本的样式信息，包括字体、颜色和布局等
- Web Page位于某个**Web服务器**（也称为HTTP服务器、网站website）
- Web Client也称为**浏览器**，浏览器采用HTTP协议（Hyper Text Transfer Protocol，超文本传输协议）与Web服务器交互，请求位于Web服务器的Web Page
- 浏览器在接收到Web Page之后，解析该Web Page，并且将其呈现给用户
- 早期的网页比较小，只有少数几个对象，现在的网页比较大，平均有140个对象



```
<html>
<head>
    <title>计算机网络</title>
</head>
<body>
    <h1>教材</h1>
    <p>高传善、曹袖、毛迪林、王雪平，《计算机网络教程(第2版)》，高等教育出版社，2013年11月，ISBN: 978-7-04-038514-4</p>
    <h1>任课教师</h1>
    <p>毛迪林，<a href="mailto:dlmao@fudan.edu.cn">dlmao@fudan.edu.cn</a></p>
    <h1>助教</h1>
    <ul>
        <li>吉成越 <a href="mailto:20210240107@fudan.edu.cn">20210240107@fudan.edu.cn</a></li>
        <li>徐 豪 <a href="mailto:21210240375@m.fudan.edu.cn">21210240375@m.fudan.edu.cn</a></li>
    </ul>
    <h1>课程进度</h1>
</body>
</html>
```

Web页面

- 静态页面：
  - 该页面包含的对象都是静态对象
  - 每次访问都得到相同的页面，这些页面可以缓存在浏览器或代理中
- 动态页面：该页面包含了一些**在需生**成的动态对象
  - 不同时刻访问时得到的页面不同
  - 可通过服务方、客户方(浏览器）的扩展生成动态页面
- 服务方的动态页面生成：服务器端的扩展
  - 通用网关接口CGI (Common Gateway Interface)：RFC 3875
    - Client要访问cgi-bin目录中的脚本程序时，Web服务器将启动该脚本程序，将请求携带的数据部分作为脚本程序的输入，脚本程序根据输入以及环境变量，输出HTML页面到标准输出，该页面将被发送回Client
  - 脚本语言 (*.php, *.jsp, *.asp等)
    - 静态HTML页面嵌入脚本语言的代码，动态生成其中部分内容
    - JSP(Java Server Pages)、PHP(PHP: Hypertext Preprocessor)
- 客户方(浏览器)的动态页面生成：浏览器端的扩展




```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Web页面

- 客户方(浏览器)的动态页面生成：浏览器端的扩展
  - 浏览器有Javascript或Java applet等解释型语言的执行环境
  - 服务方发送的HTML页面中内嵌了Javascript脚本，客户方在适当的时候执行这些javascript脚本

```
<!DOCTYPE html>
<html>
<head>
    <title>javascript程序</title>
    <script>
        function displayDate() {
            document.getElementById("demo").innerHTML = Date();
        }
    </script>
</head>
<body>
    <h1>我的第一个 JavaScript 程序</h1>
    <p id="demo">这是一个段落</p>
    <button type="button" onclick="displayDate()">显示日期</button>
</body>
</html>
```

URI、URL和URN

urn:oasis:names:specification:docbook:dtd:xml:4.1.2
tel:+1-816-555-1212
https://tools.ietf.org/html/rfc2045
http://www.urp.fudan.edu.cn:88/index.jsp
mailto: dlmao@fudan.edu.cn@subject=test
file:///home/dlmao
ssh://dlmao@mars
about:settings

- Web资源的访问有3个问题：
  - 要访问的是哪个资源?
  - 要访问的资源在哪里?
  - 怎么样访问资源?
- RFC 3986 Uniform Resource Identifier (URI): Generic Syntax
- Uniform Resource Identifier统一资源标识符 (URI): 标识（抽象或物理的）资源
- 拓展：Uniform Resource Name统一资源名称 (URN): 具有名字属性的URI
  - 早期用于描述在scheme(模式)为urn下的URI。描述资源的名称，该名称与目前的资源所在地无关，允许资源到处搬移。URN需要一个支撑架构，由其来负责解析资源的位置。比如urn:isbn:0451450523表示相应ISBN编号的书籍
  - 现在的URN被一般化，只要是具有名字属性的URI都可以称为URN
- Uniform Resource Location统一资源定位符(URL)：是URI的最常见表示形式
  - scheme + ID+ Locator

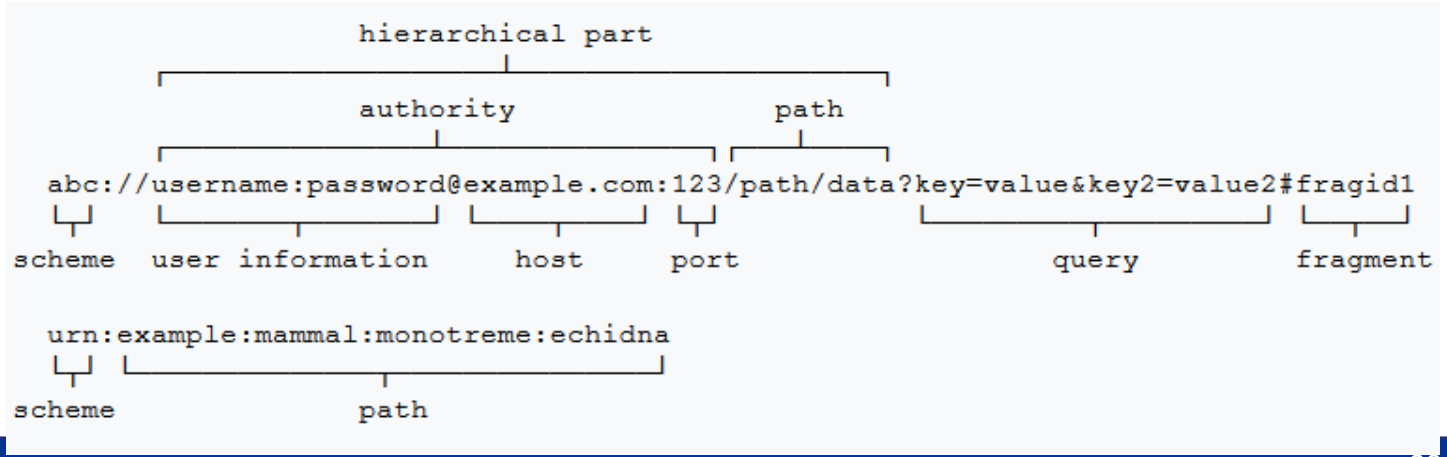
URI(URL)格式

- scheme：给出了标识资源的模式。对于URL而言，给出了如何访问该资源，如协议http(s), ftp, mail, file, rtsp, sip等，缺省为http协议
- authority：给出了管理该资源的实体(如所在的主机)，包括可选的端口号，以及可选的用户名和密码等认证信息。如果有authority，则前面必须有//。主机可以是一个域名，也可以是IPv4 IP地址或者[ipv6 address]
- path: 给出资源在主机中的位置，虽然采用类似于文件系统的目录形式，但并不一定与其对应。如果有authority，则path可为空，否则以/开头
- query: 查询字符串,传递相应的参数来访问该资源。一般为attr=value，之间以&隔开
- fragment: 描述主资源中的子资源，由client来解释

scheme:[//[user[:password]@]host[:port]][/path] [?query][#fragment]

authority

https://tools.ietf.org/html/rfc2045
http://www.urp.fudan.edu.cn:88/index.jsp
https://www.baidu.com/s?wd=uri%20urn



相对URI　scheme:[//[user[:password]@]host[:port]][/path] [?query][#fragment]

- 在使用过程中，除了采用完整的URI来描述资源外，可能使用相对URI，即缺少scheme，还可能缺少authority和某部分
  - scheme-specific URI: 只缺少scheme，根据后面的内容补充，一般为http

    fudan.edu.cn或//fudan.edu.cn → http://fudan.edu.cn
  - 拓展：只包含URI的path(可能包含一部分path）之后的内容，此时需要有一个baseURI作参照
    - scheme与baseURI一致
    - authority与baseURI一致
    - 如果path部分以/开始，则组成scheme://authority/path
    - 如果path部分不以/开始，或者没有path，则以baseURI的path部分为参照，附加path以及之后的部分

- HTML页面通过tag a描述URI

Description

baseURI= http://mars/network/base?name=tom

/network/app	http://mars/network/app
mac/csma	http://mars/network/mac/csma
?y	http://mars/network/base?y
#ref	http://mars/network/base?name=tom#ref

拓展：URI编码

scheme:[//[user[:password]@]host[:port]][/path] [?query][#fragment]

- URI的path/query等部分可能包含非ASCII字符，可能包含空格以及&?=/等元字符
- 百分号编码(Percent-Encoding)可以将URI中的字节转换为%xx的形式
  - xx为该字节的十六进制字符串
  - 回顾Quoted-Printable： =xx
  - 数字和英文字母以及下划线、连字符和点字符不需要转换
  - 非7-bit ASCII字符以及空格&?=/等可进行百分号编码

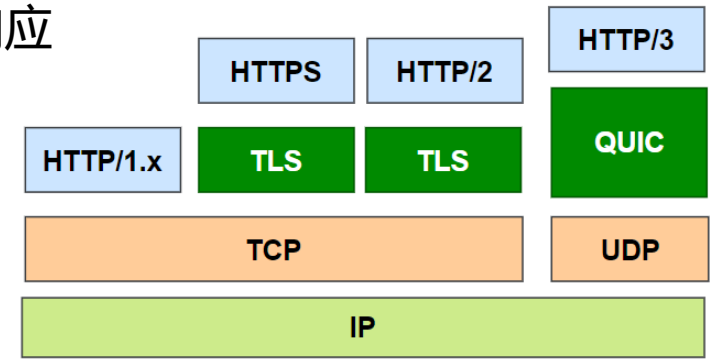
%20表示空格 %2f表示/ %26表示& **%3d表示=** **%3f表示?**

https://www.baidu.com/s?wd=uri urn
===>
https://www.baidu.com/s?wd=uri%20urn

http://search.jd.com/Search?keyword=无线路由器 华为&enc=utf-8&wq=无线路由器 华为
===>
http://search.jd.com/Search?keyword=%E6%97%A0%E7%BA%BF%E8%B7%AF%E7%94%B1%E5%99%A8%20%E5%8D%8E%E4%B8%BA**&enc=utf-8&wq=**%E6%97%A0%E7%BA%BF%E8%B7%AF%E7%94%B1%E5%99%A8%20%E5%8D%8E%E4%B8%BA

HTTP（Hyper Text Transfer Protocol，超文本传输协议）

- 采用TCP，缺省端口为80。如果为https则表示相应的TCP应该采用TLS保护，缺省的https端口为443
- 基于文本的请求响应模式：客户机向服务器发送HTTP请求，服务器发送HTTP响应
- 无状态的协议
  - HTTP服务器并不维护之前收到的**HTTP请求的状态**
  - 简化了服务器的设计，容易支持大量并发的HTTP请求
  - HTTP协议提供**cookie机制来传递状态**信息

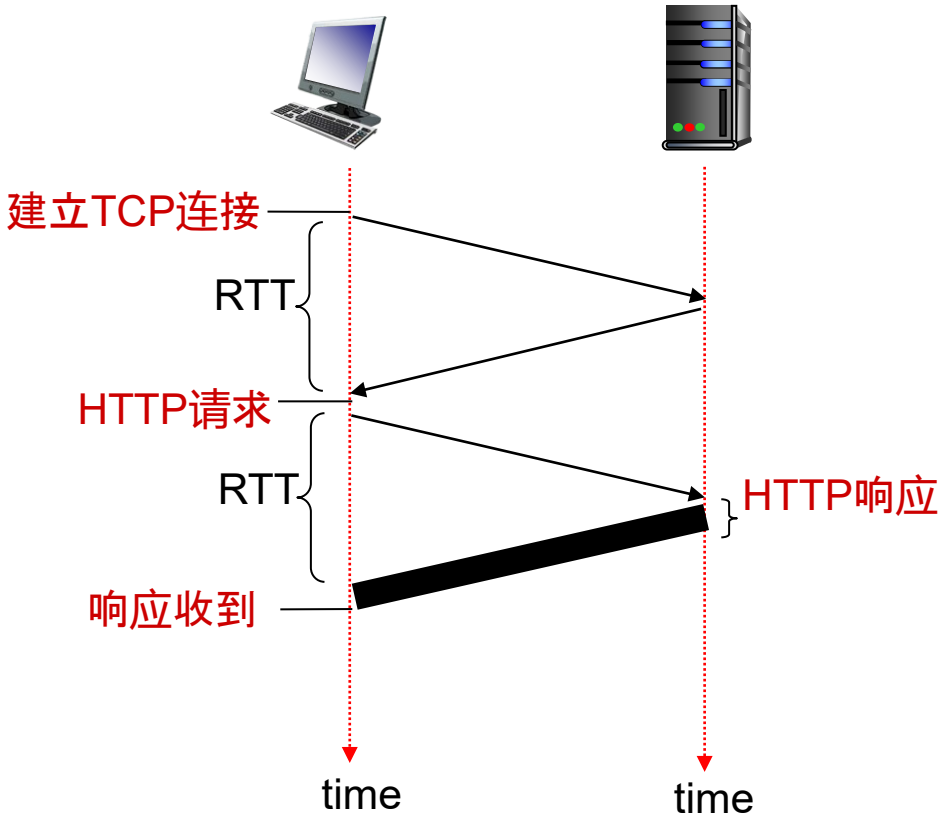


- 1991年HTTP 0.9只有一页纸，Client在TCP连接发送GET path<CRLF>，服务方发送回HTML文件，服务方关闭连接
- 1996年RFC 1945定义了HTTP1.0，请求和响应包括头部字段，支持重定向、条件GET、内容编码等
- 1997年RFC2068定义了HTTP1.1，最新的标准在RFC 7230-7235：HTTP1.1引入了**虚拟主机**(一台主机可以提供多个Web站点）、**持续连接、请求管道化、块编码**、字节范围的请求、内容协商、摘要认证、更好的缓存和代理支持等
- 2015年RFC7540定义了HTTP/2，与HTTP/1.1并不兼容。仍使用原有的HTTP框架，在底层重新定义了**新的封装和传输方法**，还支持服务方推送、头部压缩、请求优先级等
- HTTP/3：HTTP over QUIC (Quick UDP Internet Connections)
  - QUIC：google公司提出，建立在UDP之上，整合了TCP、TLS、HTTP/2等

HTTP/1.0: 非持续(Non-persistent)连接

以前一个页面的对象小于10个，现在的页面平均140个对象

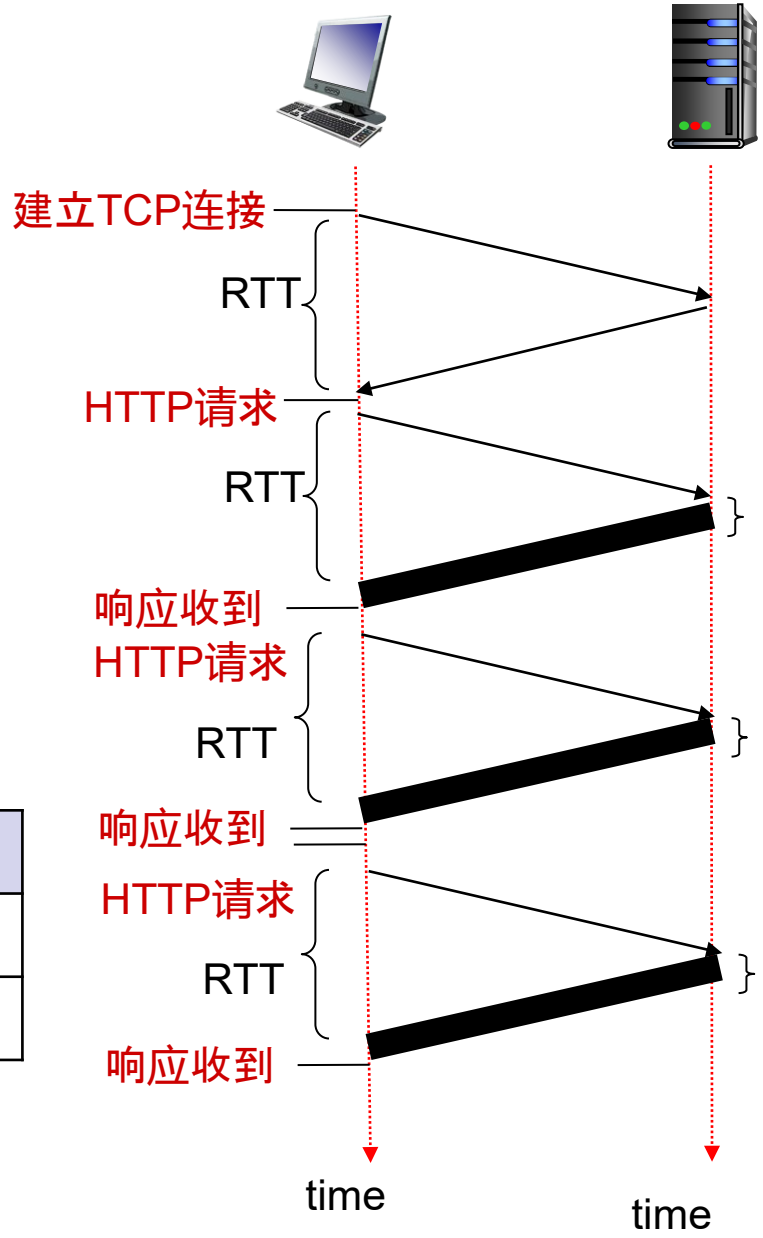
- HTTP1.0采用非持续(Non-persistent)连接方式：
  - TCP连接：
    - 连接建立需要1RTT
    - 新建的连接的拥塞窗口为1，即初始不能发送大量数据
  - 每次HTTP请求都要新建一条TCP连接，然后发送请求，服务方发送HTTP响应并关闭TCP连接
  - 如果一个HTML页面包含了N个同一服务器上的其他Web对象，假设这些对象都可容纳在一个TCP段中（忽略TCP的拥塞控制）
    - 要建立N+1条连接
    - 响应时间为: (N+1)*2RTT + 传输时间
- 有些浏览器支持并行连接方式，同时建立多条（一般最多允许6条）TCP连接，通过这些TCP连接发送HTTP请求



HTTP/1.1 持续(persistent)连接

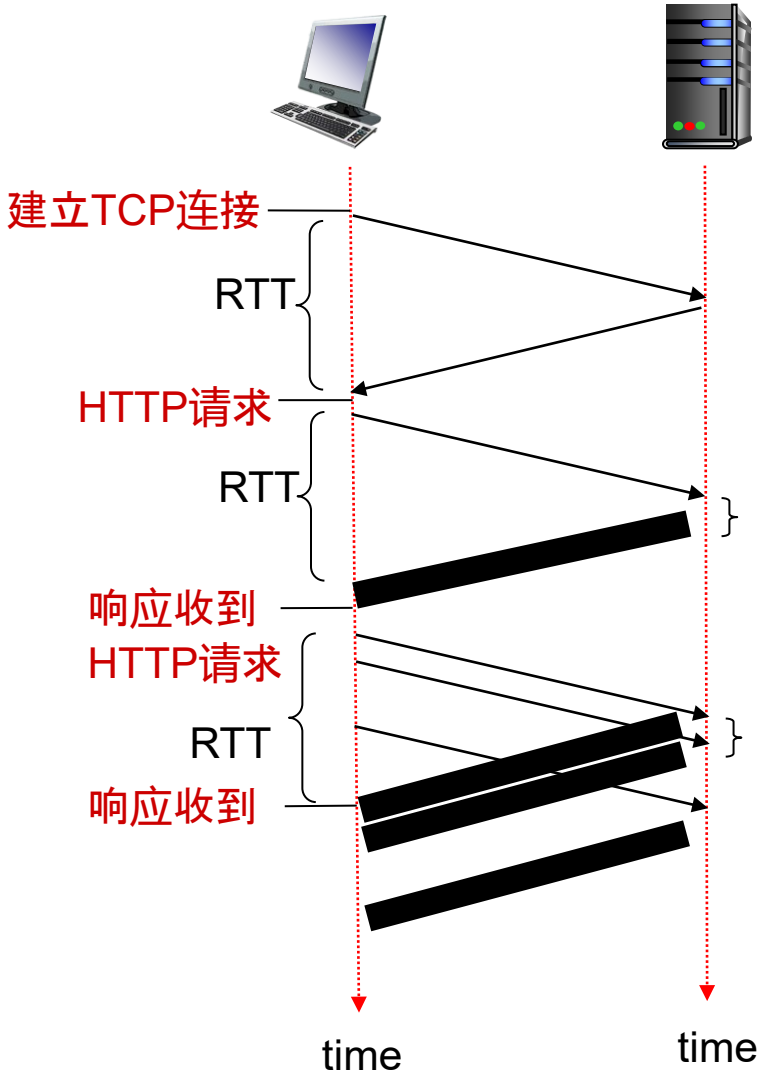
- 在收到HTTP响应时暂时不关闭TCP连接，而是等待一段时间
- 如果持续连接空闲一段时间 (keep-alive.timeout)，则关闭TCP连接
- 在此期间如果有到该服务器的新请求可以重用该连接
- 如果短时间内有多个到同一个服务器的HTTP请求，可避免连接建立的开销，拥塞窗口也不用每次从初始的1开始
- 如果一个HTML页面包含了N个同一服务器上的其他小的Web对象，响应时间为：

非持续连接	持续连接
每个请求需要2RTT	第一个请求需要2RTT，后面N请求只要RTT
(N+1)*2RTT + 传输时间	(N+2)RTT + 传输时间



HTTP/1.1 持续(persistent)连接

- 非流水线方式的持续连接：
  - 发送第一个请求，收到第一个响应
  - 发送第二个请求，收到第二个响应....
- 流水线方式(pipelining，管道化)的持续连接
  - 发送第一个请求，收到第一个响应(HTML页面)
  - 发送第2、3、4...个HTTP请求，收到第2、3、4...个HTTP响应
  - 队头阻塞(Head of line blocking): 按照请求的顺序发送HTTP响应，如果一个对象较大，阻塞后面的对象
  - 要求**HTTP响应自身能够决定边界**（即结束的位置）
  - 绝大部分浏览器缺省关闭流水线方式

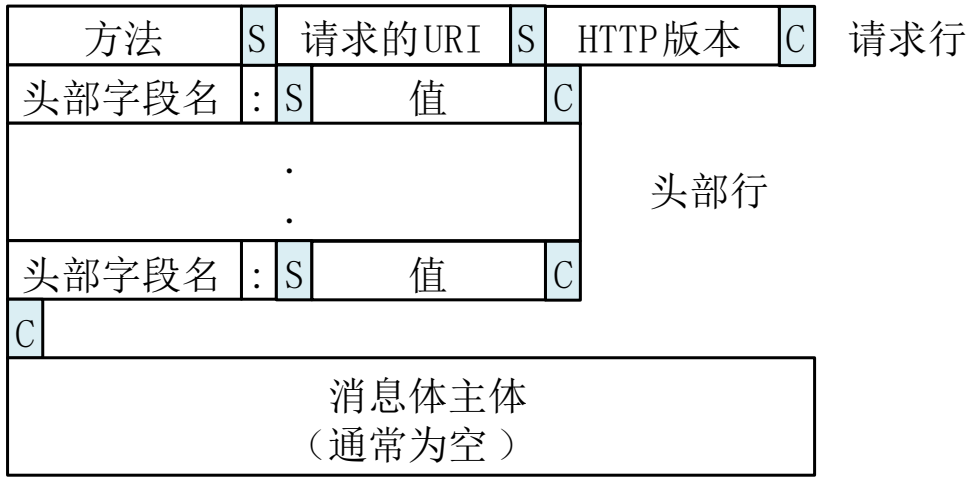


非持续连接	持续连接	流水线方式持续连接
每个请求需要2RTT	第一个请求需要2RTT，后面请求只要RTT	第一个请求2RTT，然后发送所有请求
(N+1)*2RTT + 传输时间	(N+2)RTT + 传输时间	如果所有**请求响应**可以容纳在**一个**TCP段中 3RTT + 传输时间

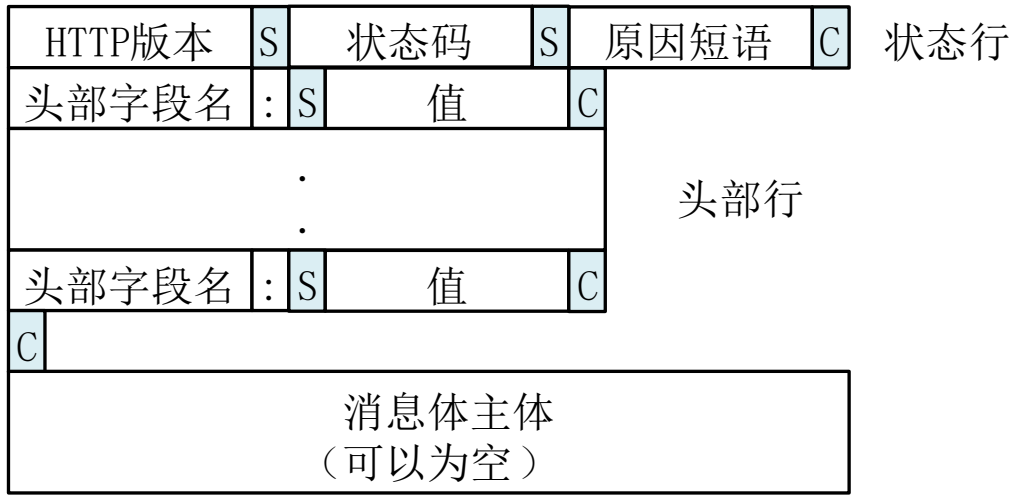
HTTP请求和响应

基于文本的请求响应模式，请求和响应中可包含头部和消息体，支持MIME

- HTTP请求中的方法（包括HTTP版本都是大写，**大小写相关**）一般为GET，而消息体部分一般为空
- 头部和消息体之间通过空行隔开
- 头部用于传递请求或者响应相关的辅助信息，头部字段名**大小写无关**，冒号后面可有0到多个空格（建议使用1个空格），值最后也可有0到多个空格
- HTTP1.1要求必须有**Host头部**，除此之外其他头部都是可选的
  - 引入**虚拟主机**，即一台主机（IP地址）可提供多个Web站点，通过**Host头部**描述是哪个Web站点
- 消息体并不要求一定为US-ASCII字符。消息体的结束位置可通过头部**Content-Length**等描述



注：其中 S 表示空格(Space)，C 表示回车换行(CRLF)



注：其中 S 表示空格(Space)，C 表示回车换行(CRLF)

HTTP请求

- 请求行中的URI一般不包括主机部分
- **Host头部**描述是哪个Web站点

请求行
(GET/POST/HEAD **主要方法**
PUT/DELETE/TRACE
OPTIONS/CONNECT)

头部行

空行表示头部部分的结束
\r\n\r\n

```
GET /2016/index.html HTTP/1.1
Host: www.fudan.edu.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: amlbcookie=01; iPlanetDirectoryPro=xxxxxxxxxxxxxxx
DNT: 1
Connection: keep-alive
```

GET	获取资源。必须实现
HEAD	与GET类似，只是不包括资源的具体内容。必须实现
POST	用于表单(form)提交，请求中包含body部分，对资源进行处理
PUT	请求中包含body部分，替换资源
DELETE	删除资源
CONNECT	建立隧道
OPTIONS	询问服务器支持的HTTP请求方法等选项
TRACE	调试，服务器将收到的HTTP请求作为消息体返回

实际访问的URL为：
http://www.fudan.edu.cn/2016/index.html

来自于Host头部　　来自于请求行

- 请求行中的URI部分不包含fragment部分，**fragment由Client解释**
- 使用代理(**Proxy**)时，请求行中的URI中为**完整URI**，即包含了authority(主机和端口号等）部分

HTTP响应

状态码：

- 1XX – informational，还不是最终的响应
- 2XX – success
- 3XX – redirection，在另一个位置或者缓存中获取
- 4XX – client error
- 5XX – server error

100 Continue
200 OK
204 No Content
301 Moved Permanently　Location头部给出新的位置
304 Not Modified
401 Unauthorized
403 Forbidden
404 Not Found
500 Internal Server Error
505 HTTP Version Not Supported

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 20 Oct 2016 00:41:27 GMT
Content-Type: text/html
Content-Length: 75643
Last-Modified: Tue, 18 Oct 2016 09:58:11 GMT
ETag: "5805f233-1277b"
Accept-Ranges: bytes
X-Cache: MISS from 2ndDomainSrv
X-Cache-Lookup: MISS from 2ndDomainSrv:80
Via: 1.1 2ndDomainSrv (squid)
Connection: keep-alive

...<!DOCTYPE HTML>

HTTP请求头部：为请求提供参数

拓展的内容

- 采用持续还是非持续连接?
  - Connection头部为close，表示非持续连接
  - HTTP/1.0，但Connection取值keep-alive，且服务方支持，持续连接
  - HTTP/1.1，表示持续连接

头部（部分)	含义
Host	服务器可能支持多个虚拟网站，访问哪个网站上的资源
User-Agent	浏览器的相关信息
Referer	（Referrer) 引用页的URI
Connection	收到HTTP响应后是否关闭，可取值 keep-alive, close, upgrade
Accept/Accept-Language/Accept-Encoding	内容协商：客户方可接受的资源类型/语言/编码(压缩)方式(比如gzip)
If-Modified-Since/If-None-Match等	条件GET
Authorization/Proxy-Authorization	发给服务器或者代理的认证信息
Cookie	Cookie信息（来自于之前服务器通过set-cookie发送的信息)
Transfer-Encoding	消息体传输采用的编码(包括压缩）方式，比如gzip, chunked
Content-Length	消息体的长度，以字节为单位
Content-Encoding/Content-Language/ Content-Type	如果有消息体时可包含，描述内容的编码(压缩)方式/语言和类型等

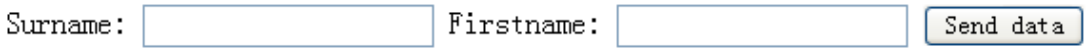
HTTP响应头部：说明响应的其他信息，有些头部可出现在请求和响应头部

拓展的内容

头部	含义
Server	服务器的相关信息
Date	响应发送时的时间
Location	重定向时使用，客户方应该发送请求的新URL
Connection	是否关闭连接
Content-Encoding/Content-Language/Content-Type	响应携带的内容的编码(压缩)方式/语言和类型
Content-Length	HTTP响应的消息体的长度
Transfer-Encoding	消息体传输采用的编码(包括压缩）方式，比如gzip, chunked
Last-modified	资源上次修改的时间
ETag	唯一表示资源内容的Tag
Expires	资源过期时间
Cache-Control	缓存控制，可指定是否缓存或最大age多少
Set-Cookie	设置Cookie，可多次出现
WWW-Authenticate Proxy-Authenticate	支持的认证方式、代理认证方式

HTTP GET和POST方法

```
<form action="http://www.foo.com/myProg.cgi" method="get">
Surname:  <input type="text" name=”surname” >
Firstname:  <input type="text" name=“firstname” >
<button type="submit"> Send data </button>
</form>
```



如何传递表单(form)数据?

- 采用GET方法，通过HTTP URL传递form数据
  - form数据作为URL最后面的查询字符串
  - 可能超过URL的限制，且URL会被纪录到访问历史中

拓展的内容

http://www.foo.com/myProg.cgi?surname=Mao&firstname=Dilin

- 采用POST方法，查询字符串将通过消息体传递
  - POST方法后的URL为处理form的页面
  - Content-Length: 给出POST请求中的消息体长度
  - Content-Type：给出消息体包含内容的类型
    - application/x-www-form-urlencoded, 查询字符串采用%编码

```
GET /myProg.cgi?surname=Mao&firstname=Dilin HTTP/1.1
Host: www.foo.com
```

```
POST /myProg.cgi HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: www.foo.com
Content-Length: 27

surname=Mao&firstname=Dilin
```

HTTP：块编码(chunked encoding)

拓展的内容

客户方发送请求后如何知道返回的HTTP对象(消息体)的结束?

- 服务方发送响应后关闭TCP连接
- Content-Length头部给出了消息体的长度
- Transfer-Encoding头部给出了消息体是怎么编码的，如果为chunked，表示采用块编码
  - 对象被分隔成多个块，分多次传输，一次一块
  - 每个块格式: chunk-size [chunk-ext]CRLF chunk-data CRLF
  - 第一行最前面是chunk-size，给出了chunk-data部分的长度，以16进制方式描述
  - 如果chunk-size为0，表示为最后一块，接下来为CRLF，即以一个空行结束

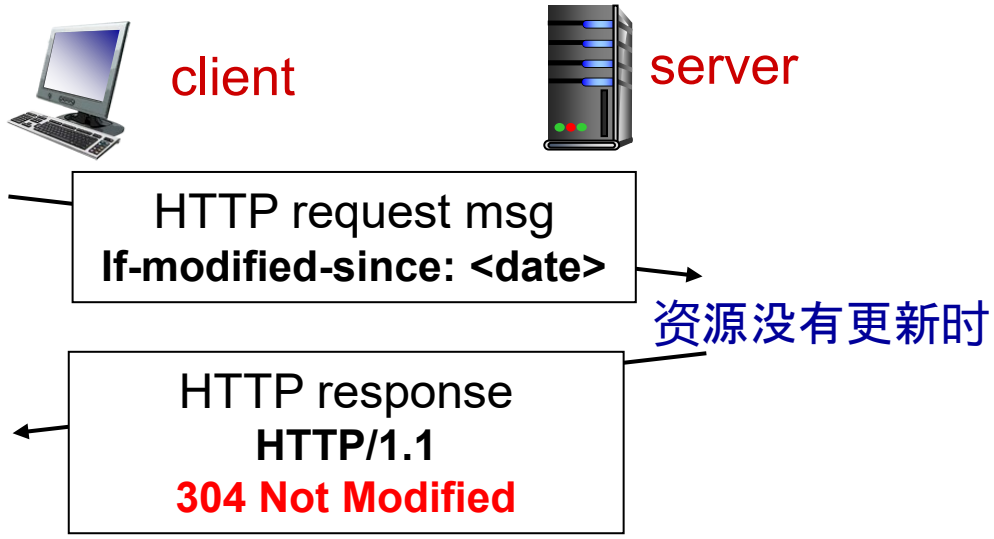
```
HTTP/1.1 302 Moved Temporarily
Server: nginx
Date: Thu, 20 Oct 2016 00:41:27 GMT
Content-Type: text/html; charset=UTF-8
Location: 2016/index.html
Transfer-Encoding: chunked
Connection: keep-alive

A1



<!--[if lt ie 9]>
 <script language="JavaScript" type="text/javascript">
        window.location.href="index.html";
 </script>
<![endif]-->




0
```

缓存控制

拓展的内容

- 浏览器一般会缓存最近GET的Web对象，根据HTTP响应的头部：
  - 控制如何缓存?
    - Cache-Control: 缓存控制
      - Cache-Control: no-cache 不允许缓存
      - Cache-Control: max-age=15 可以缓存的时间(15秒)
    - Expires: 什么时候过期。Cache-Control的max-age优先
  - 如果缓存, 还会记录：
    - Last-Modified: 上次修改的时间
    - Etag: 资源唯一的散列值
- 启发式策略：根据Expires头部判断缓存中的资源尚未过期。但服务器也很难判断资源的更新时间，较少使用Expires头部
- 询问式策略：如果缓存中有且没有过期，可以发送条件GET检查缓存中的资源对应的原始资源是否有更新
  - If-None-Match：散列值改变时返回新资源
  - If-Modified-Since: 在此之后修改过时返回新的资源。If-None-Match头部优先

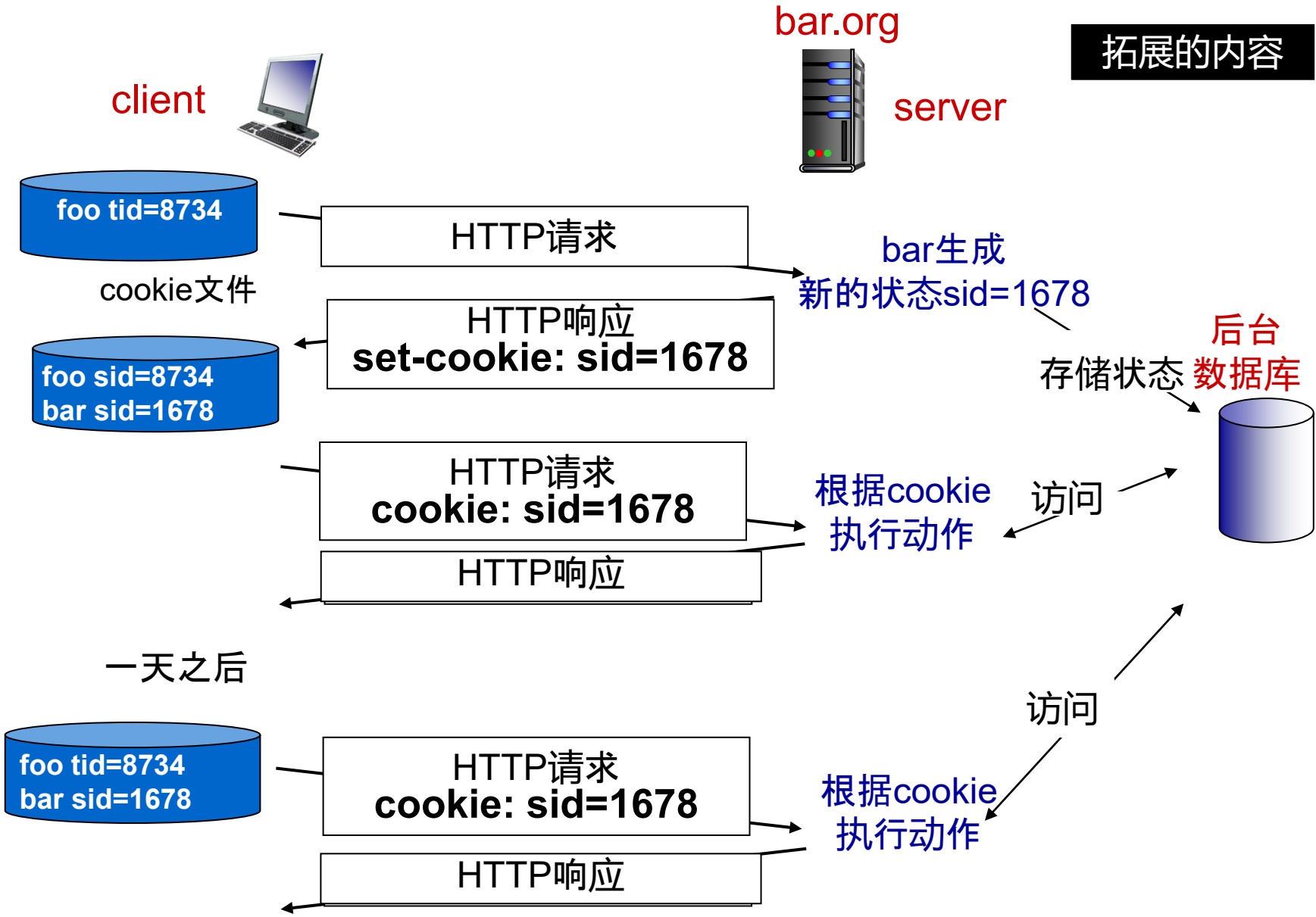
Last-modified	资源上次修改的时间
ETag	唯一表示资源内容的Tag
Expires	资源过期时间
Cache-Control	可指定是否缓存或最大age多少



HTTP: Cookie

拓展的内容

- RFC 6265，可采用Cookie维护客户的状态
  - HTTP提供了传递Cookie的机制：HTTP请求的Cookie头部和HTTP响应的 Set-Cookie头部（可多个）
  - Web站点的后台服务器生成和处理Cookie
  - 浏览器保存cookie
  - 浏览器根据所访问的站点的主机名以及路径部分，从保存的cookie文件中挑选出匹配的cookie，然后通过Cookie头部传递



Set-Cookie: SID=31d4dad42; Path=/; Domain=example.com
Cookie: SID= 31d4dad42

Cookie: NAME1=VALUE1; NAME2=VALUE2

HTTP: Cookie

拓展的内容

- 响应可以包括多个Set-Cookie头部，传递多个状态(如会话ID和用户偏好等)
- 每个Set-Cookie头部描述某个状态的值是什么，以及该状态的相关属性
  - 格式(分号和冒号后面有一个空格)　Set-Cookie: name=value; attr1=val1; attr2=val2

  比如Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=example.com
  - Domain属性表示该cookie可适用的域名，缺省为当前服务器
  - Path属性表示该cookie可以用于的路径部分，缺省为/，表示服务器的任意位置的资源
  - cookie-name+path+domain唯一标识一个cookie
  - Expires属性给出了cookie的过期时间，如果其为过去的日期，表示删除该cookie
  - Max-Age属性给出了cookie可以保存的最长时间，比Expires优先级高

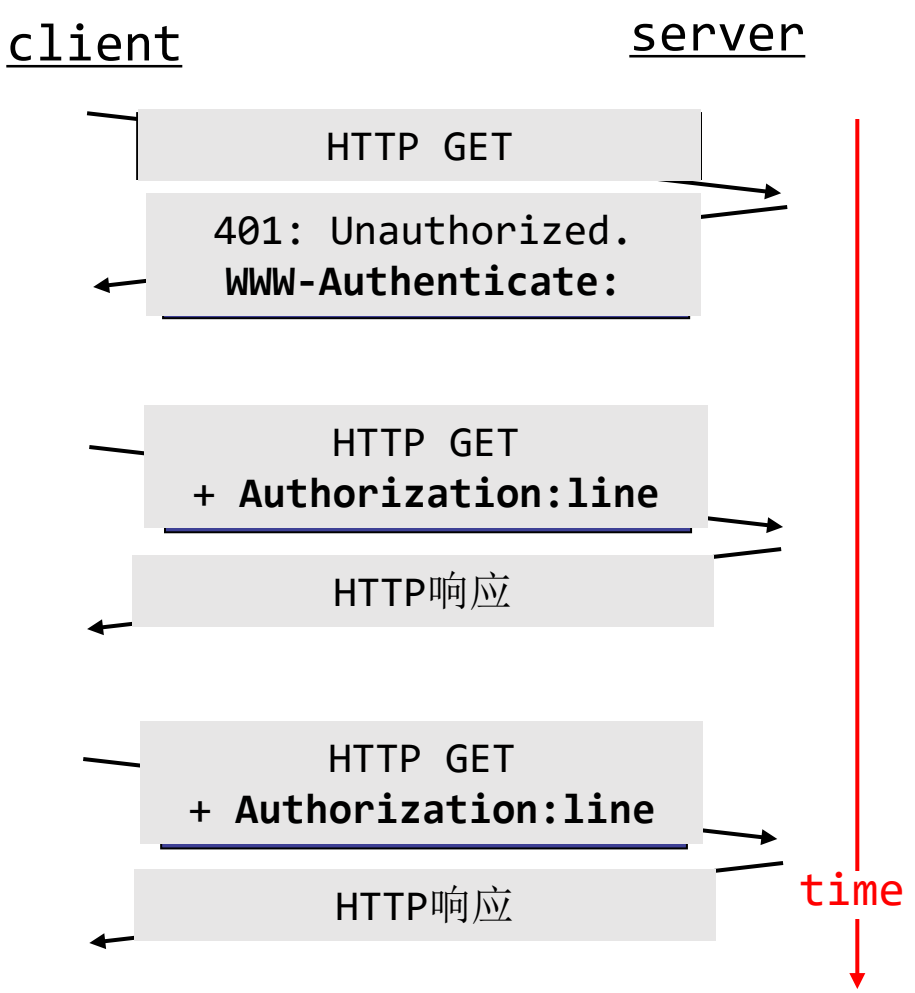
➢ Cookie技术是把双刃剑

  - 能维护用户状态信息，分析用户喜好，向用户进行个性化推荐
  - 跟踪用户网络浏览痕迹，泄露用户隐私
  - 用户可以限制Cookie的使用

HTTP：用户认证

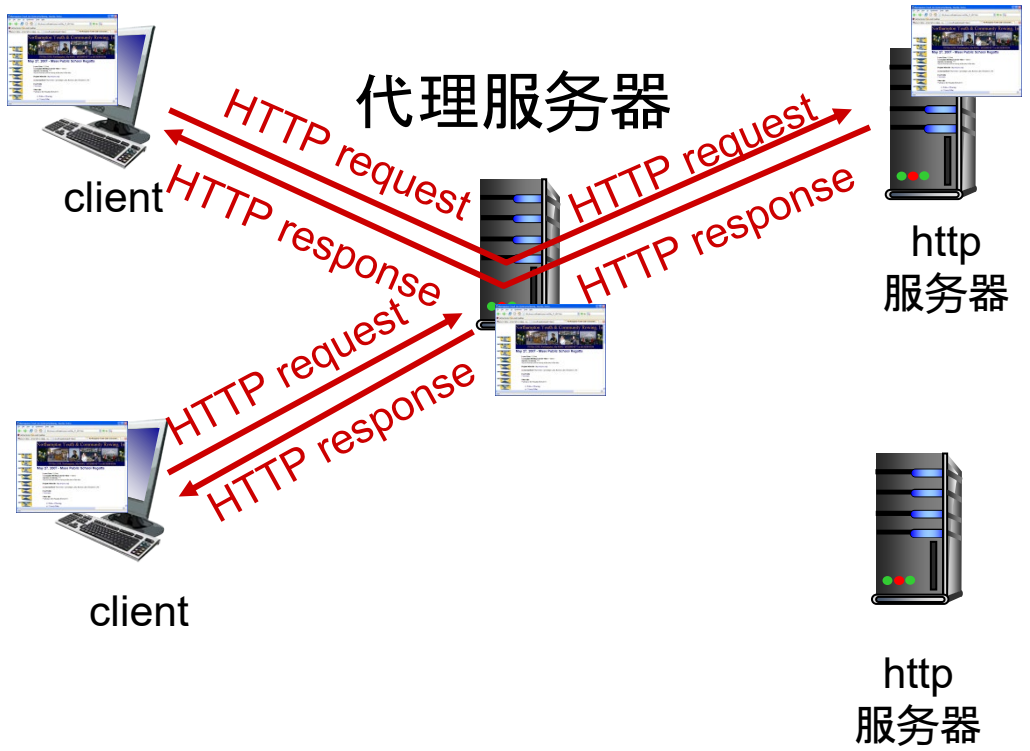
拓展的内容

- 某个资源要求验证用户身份时，发送HTTP响应401(Unauthorized)
  - WWW-Authenticate头部给出了用户应该采取的认证方法和区域(realm) **WWW-Authenticate: Basic realm="foo"**
  - 区域的内容在浏览器中通过弹出窗口形式显示，询问用户名和密码等
- HTTP请求中可通过Authorization头部传递认证信息
  - RFC 7617定义了Basic认证方式。user:password采用base64编码，HTTP请求包括头部Authorization: Basic user:password
  - RFC 7616定义了Digest认证方式，基于共享密钥，不明文传递密码，而是传递hash值
- 代理服务器的认证类似，HTTP响应使用Proxy-Authenticate头部，HTTP请求使用Proxy-Authorization头部



Web缓存或代理服务器

- 浏览器设置代理服务器的地址
- 用户浏览器访问HTTP服务器时HTTP请求发送给代理服务器
  - 如果代理服务器的缓存中已有该资源，则返回给用户
  - 否则代理服务器作为HTTP协议的客户方发送请求给服务器，在收到资源后再发送HTTP响应给用户
- 代理服务器减少了响应时间
- 代理服务器减少了外出链路的负载



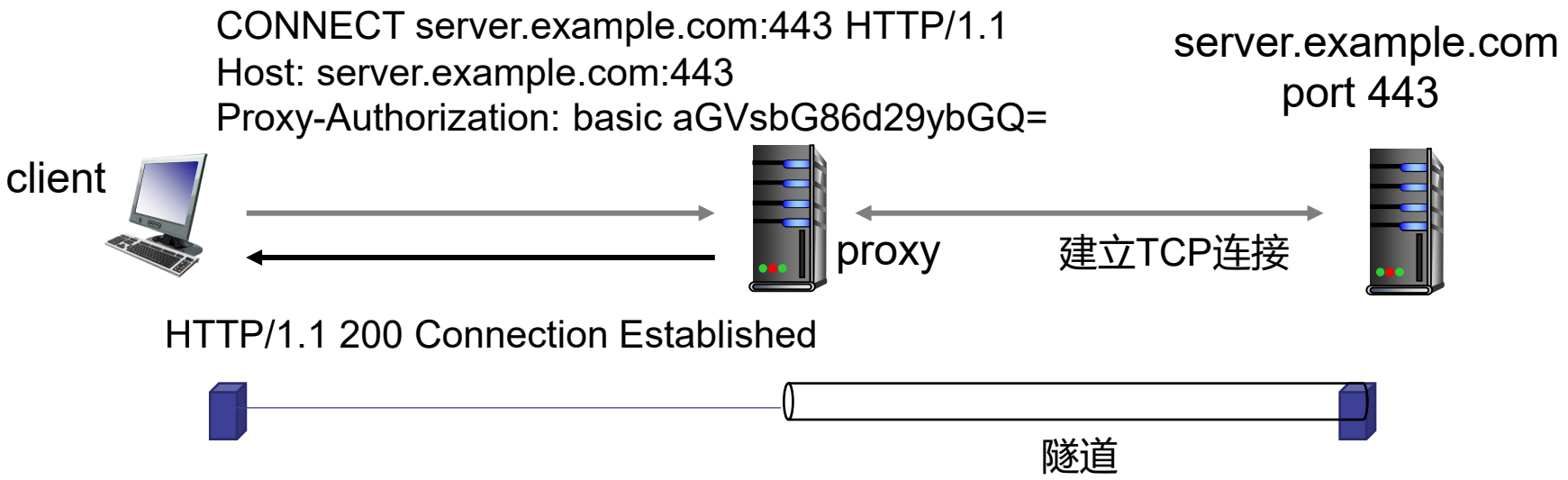

```
GET http://mail.qq.com/ HTTP/1.1
Host: mail.qq.com
Proxy-Authorization: Basic aGVsbG86d29ybGQ=
User-Agent: curl/7.58.0
Accept: */*
Proxy-Connection: Keep-Alive
```

- 代理服务器对于client来说充当了client方的HTTP服务器的角色
- 对于HTTP服务器而言，代理服务器充当了客户方的角色

HTTP隧道

如果要通过代理使用HTTPS协议（首先通过TLS协议建立安全连接，然后HTTP over TLS）访问某个HTTP服务器呢？ 引入HTTPS代理

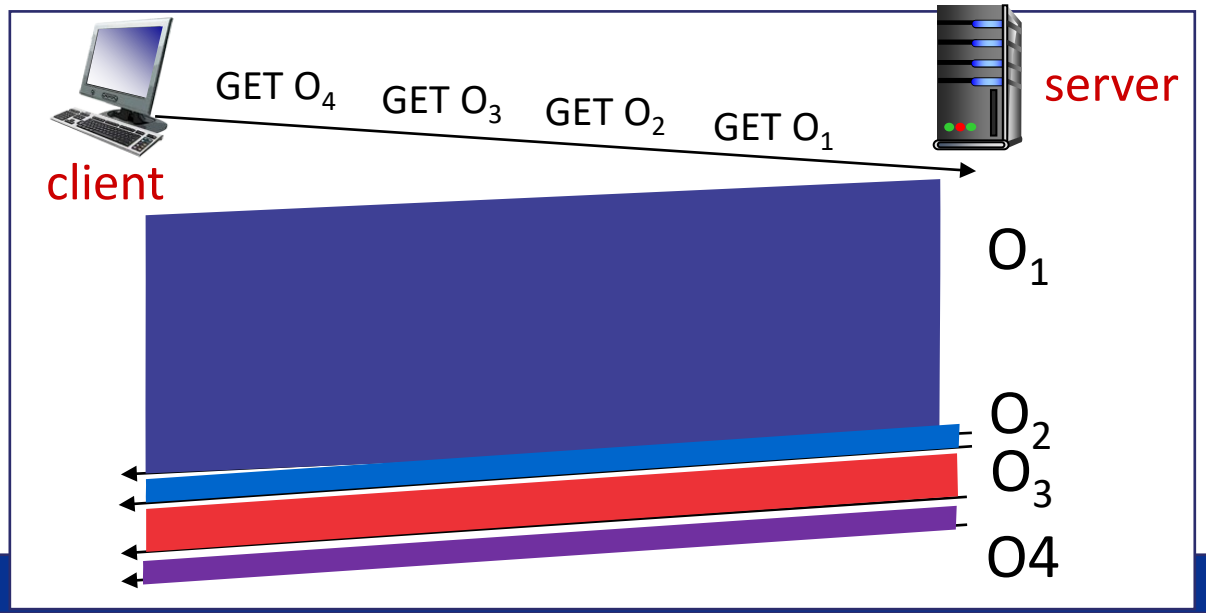
- 通过HTTP Connect方法建立从proxy到（https)服务器间的隧道
- 一旦隧道建立好之后，Client和Server之间可以使用任何协议
  - client发送的任何数据(不一定是HTTP请求）通过该TCP连接到达proxy后通过隧道转发到Server
  - Server来的分组通过隧道到达proxy然后转发给client
- 不是所有的代理服务器支持HTTP隧道，即便支持一般也限制其连接的端口号或者地址，比如仅仅限制连接到443号，即支持HTTPS负载的转发

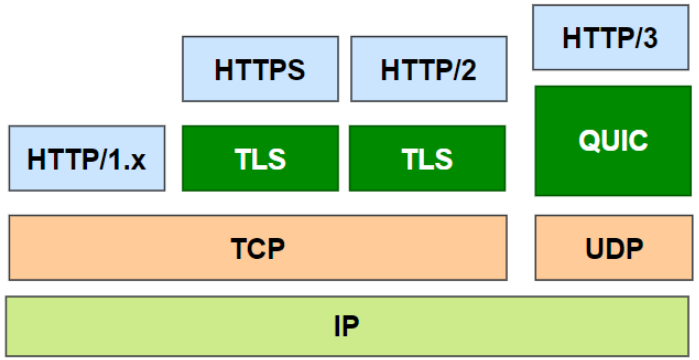


HTTP/2

拓展的内容

- 2015年RFC7540定义了HTTP/2。最新版本为2022年6月发布的RFC9113
- 仍使用原有的HTTP框架，包括请求方法、状态码、请求和响应头部等， 与HTTP/1.1并不兼容
- 在底层重新定义了**新的封装和传输方法**，还支持**服务方推送**、头部压缩、请求优先级等
- 关键目标：提高带宽利用率，减少延迟
- 考虑访问一个包含1个视频对象和3个小对象的页面：
  - HTTP/1.1: 管道化的持续连接方式
    - 发送多个HTTP请求，响应按照请求到来的先后顺序回应
    - 小的对象要等待前面的大对象传输完成后才可以发送，即出现队头阻塞(Head-of-Line Blocking)
    - 如果TCP段丢失，需要进行重传，使得后续的对象的传输延迟：HTTP/2(over TLS over TCP)并不解决这个问题





HTTP/2

- 把object分成多个Frame，这些Frame交错(interleaved)发送
- 平均延迟减少：
  - 小的对象相比HTTP/1.1更早到达
  - 大的对象稍微延迟到达
- 页面中的对象的重要性不同，HTTP请求引入优先级机制，优先级高的请求所请求的对象优先传输
- 推送(push)机制：服务方将客户方尚未请求但是有很大可能即将请求的对象主动推送给客户方
- 不再采用基于文本方式的请求和响应协议，而是基于二进制方式的封装，实现起来更加高效
- 引入头部压缩机制，减少头部传输的开销