HTTP

1. **http请求种类（h0001）**
2. 简单请求（simple request）
3. 非简单请求（not-so-simple request）

**判断简单请求的种类的两个条件：**1）.请求方法必须是以下的一种

get 查询

head

post

2）.HTTP头只能包括以下信息

Accept

Accept-Language

Content-Language

Last-Event-ID

Content-Type只限于：[application/x-www-form-urlencoded,

multipart/form-data,text/plain]

DPR

Downlink

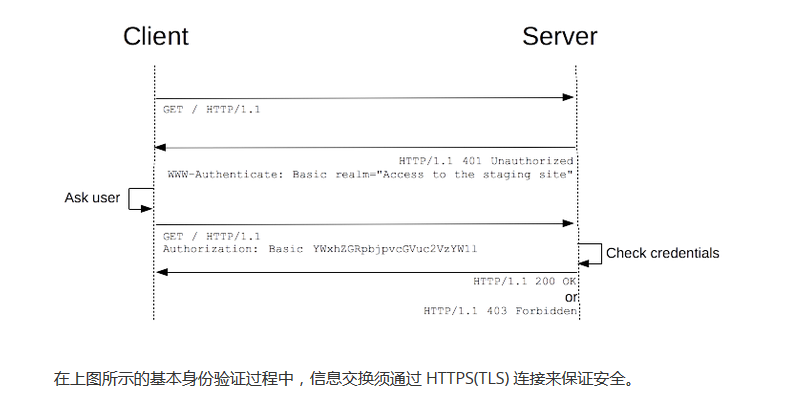
Save-Data

View-Width

Width

**不满足以上两个条件的视为非简单请求**

1. **authorization**



**区别于独立的头信息和响应状态码，资源认证和代理认证可以并存。**

# url与资源

## url

是uri的一个子集（http规范将uri作为其资源标识符，实际上http处理的只是url）

url也可以用http之外的其他协议来访问

如 e-mail ftp

mailto:president@whitehouse.gov

## 构成部分

url语法建立在9部分构成的通用格式

<scheme>://<user> :<password>@<host>:<port>/<path>;<params>?<query>#<frag>

方案 用户 密码 主机 端口 路径 参数 查询 片段

方案

如何访问资源（也就是协议）

大小写无关

用户名与密码

某些方案需要输入用户名和密码，如ftp。如果一些代理没有输入用户名和密码，如ie（会插入anonymous作为用户名，密码为IEUser）

主机与端口号

路径

说明资源位于服务器什么地方

参数

用 ；将其余路径分割，每一个路径段都可以有参数

[www.e.com/laalalal;sale=false/index.html;good=false](http://www.e.com/laalalal;sale=false/index.html;good=false)

查询

？a=1&b=2

片段

片段支持指向文档中一个特定的小节（文档中位置）

## 相对url

相对url是基于 基础url的。

基础url

在资源中显示提供

<base>

封装资源的基础url

当前资源文件的url

## url在浏览器地址栏中输入的自动扩展

主机名扩展

历史扩展

## url字符

由于url是可移植的，其需要对互联网上的所有资源命名，因而其在所有互联网协议上都可以安全传输（不能损失任何信息）很重要。

而一些协议（如简单邮件协议的传输方法会剥去一些特定字符），为避免这一问题，url只能使用一些较小的，通用的安全字符表中的字符。

同时也为了可阅读性，不可见，不能打印的字符

同时也需要通用的安全字符表之外的二进制数据或字符。（需要一种转义机制）

# HTTP报文

## 报文流

上游，下游：描述报文流向

请求 ，响应 上下游相反

在流动过程中，会经过代理

## 组成部分

简单的格式化数据块

起始行（对报文描述 ： start line）

首部块（包含属性的 ： header）

主体（数据 ：body）

起始行和首部就是由行分隔的ASCII文本。分隔符号为：一个回车符 + 一个换行符（CRLF） 或者 一个换行符

entity-body：实体

reason-phrase：原因短语--只对人有意义

请求格式

<method> <request-url><version>

<headers>

<entity-body>

响应格式

<version> < status> <reason-phrase>

<headers>

<entity-body>

## 方法

安全方法： GET ，HEAD

GET :

请求资源

HEAD:

作用：

1. 在不获取资源的情况下了解资源的情况
2. 通过查看响应中的状态码，看看某个对象是否存在
3. 查看首部，测试资源是否被修改了

服务端实现

遵循HTTP/1.1规范，必须确保服务端返回首部于GET返回的首部完全相同。

PUT:

向服务器写入文档，它的语义实际上让服务端用请求的主体部分来创建一个由所请求的url命名的新文档，如果url已经存在，就用这个主体替换它。

POST:

向服务端输入数据。

TRACE:

请求经过防火墙，代理，网关或其他一些应用，每个中间节点都可能修改原始的HTTP请求。

TRACE请求会在目的服务端发起一个“环会”诊断，当请求到最后一站的服务器，会弹回一条TRACE响应，并在响应主体中携带它收到的原始请求报文。这样就可以判断请求原始报文是否被修改，毁坏过。

OPTIONS：

请求服务器告知其支持的各种功能

可以询问支持的特定的方法

DELETE:

常用扩展方法：

LOCK:锁定资源 MKCOL：创建资源 COPY：赋值资源 MOVE：移动资源

## 状态码

100~199 信息状态码

100 continue

客户端发送一个 100 continue expect ，期待服务端返回一个是否会接收请求实体的 100 continue 响应。

返回 100 continue后再发送实体（但也要考虑服务端超过一段时间后不发送 100 continue，这时仍然要发送实体）

发送 100 continue前，服务端已经接收到实体了，应该不再发送100 continue状态。

代理如果知道下一跳服务器是否与 HTTP/1.1兼容，如果知道不兼容，则返回 417 expectation failed。除此之外，将expect 转发即可。

## 首部

### 通用首部

服务端，客户端通用首部

Connection : 连接有关选项

keep-alive

Date: 创建时间

MIME-Version: MIME类型

Transfer-Encoding: 对报文采取的编码方式

Update：对发送端发送想要‘升级’的新版本或则协议

Trailer：分块传输，列出报文拖挂部分的首部集合

Via: 报文经过的中间节点（代理，网关）

通用缓存首部

Cache-Control：报文缓存指示

Pragma：报文缓存指示（优先级低于Cache-Control）

### 请求首部

Client-IP : 客户端的机器的IP地址

UA-Color UA-CPU UA-Disp UA-OS UA-Pixels User-Agent

Client-IP 与 UA- 不在规范中，但是许多客户端都已经实现了（尽量不要用）

Form： 客户端用户的E-mail

Host: 服务器主机名

Referer： 全部的URI（包括路径和query）

Accept首部

Accept : 接收媒体类型

Accept-Charset ： 接受的字符集

Accept-Encoding ： 接受的编码方式

Accept-Language ： 接受的语言

TE ： 可以使用的扩展传输编码

条件请求

Expect 要求的服务端行为

If-Match 实体标记匹配，获取文件

If-None-Match 实体标记不一致，获取文件

If-Modified-Since 资源在指定日期后被修改，获取文件

If-Unmodified-Since 资源没有在指定日期后被修改，获取文件

If-Range 对文档某个范围进行请求

Range 如果服务器支持范围请求，请求资源的指定范围

安全请求首部

Authorization

Cookie

Coolie2 说明请求端支持的coolie版本

代理请求首部

Max-Forward 最大通过代理或者网关的次数（与trace方法一起使用）

Proxy-Authorization 与 Authorization首部相同，代理时认证采用

Proxy-Connection 与 Connection 首部相同，代理时连接使用

### 响应首部

Age 响应持续时间(从最初创建开始，暗示响应是通过中间节点，很可能是从代理缓存中传送过来)

Public

Retry-After 如果资源不可用，再次日期重试。

Server 服务端应用程序 和版本

Title

Warning 比原因短语中更详细一些的警告报文、

协商首部

Accept-Ranges 服务器可接受的范围类型（对此资源）

Vary 服务器查看其它首部的列表，根据这些首部内容挑选出最适合的资源版本发给客户端。

安全响应首部

Proxy-Authenticate 代理对客户端打的质询

Set-Cookie 设置cookie

Set-Coolie2 设置coolie版本

WWW-Authenticate 服务端对客户端的质询

### 实体首部

对应实体主体部分的首部，用于描述HTTP报文的负荷

Allow 列出了可对此实体执行的方法

Location 告知客户端实体实际位于何处，将接收端定向到资源(可能是新的)

内容首部

Content-Base 主体中相对url使用的基础url（规范中未定义）

Content-Encoding 对主体执行的编码方式

Content-Language

Content-Length 主体的长度或尺寸

Content-Location 资源实际所处的位置

Content-MD5 主体的MD5校验和

Content-Range 在整个资源中实体表示的字节范围

Content-Type 主体的mime类型

实体缓存首部

ETag 与此实体相关的实体标记(对应If-None-Match If-Match)

Expires 在这个时间后实体失效

Last-Modified 这个实体最后一次被修改后的日期和时间

### 扩展首部（自定义首部）

# 连接管理

几乎所有的HTTP通信都是由TCP/IP承载，TCP/IP是一种分组交换网络分层协议集。

## TCP连接

HTTP连接实际上就是TCP连接及其使用规则。

TCP为HTTP提供了一条可靠的比特传输通道。

TCP的数据流是分段的，是通过 IP分组（IP数据报）的小数据块来发送的。

HTTP over TCP over IP ,而HTTPS是在HTTP与TCP之间插入了一个 （TLS 或 SSL）密码加密层。

### 传输过程：

HTTP要传送一条报文，会以流的形式将报文数据的内容通过一条打开的TCP连接按序传输。

TCP收到数据流后，会将数据流砍成被称为段的小数据块，并将其放在IP分组中，通过互联网进行传输

而这些都是通过TCP/IP软件处理的。

**TCP连接的底层细节**

**85页**

### IP分组构成

一个IP分组首部

一个TCP 段首部

一个TCP数据块

### TCP性能

TCP性能很大程度决定了HTTP事务的性能

86页

HTTP的时延的影响因素

TCP连接建立握手

在建立一条新的连接之前（甚至是发送任意数据之前），tcp软件之间会交换一系列的ip分组，对连接的相关系数进行沟通。这对于小的http事务，消耗相对很大。

建立新的tcp连接时，会

客户端发送一个小的TCP分组（40~60字节），内部设置了一个特殊的SYN标记，说明这是一个连接请求

服务器接收连接，会对连接参数进行计算，并向客户端发送一个tcp分组，这个分组内的 SYN和 ACK标记都被置位，说明连接已被接受。

最后客户端发送一个确认信息（现代的TCP栈运行客户端在这里发送数据）

TCP慢启动拥塞控制

数据聚集的 Nagle算法

用于捎带确认的TCP延迟确认算法

TIME\_WAIT时延和端口耗尽

### 并行连接

相对于串行连接，同步等待上一个请求完成，再发起下一个请求。并行请求通常要快一些。

但是实际上由于宽带限制，并行加载的多个对象会竞争有限的宽带，而且会产生一些额外的开销。有可能使得加载速度变慢。（浏览器会限制连接总数，一般是4个）

### 持久连接

# web服务器

## web服务连接

建立一个半相关

建立连接

客户端请求一条到web服务器的TCP连接时，web服务器会获取另一端ip（通过套接字），判断是否允许连接，建立连接。

-----------可以通过‘反向DNS’对大部分服务器进行配置，以便将客户端ip转换为客户端主机名。

接受请求

请求可能在任意时刻到达，所以web服务器会不停观察有无新web请求。

对于：

1. 单线程web服务器

一次只能处理一个请求，所有其他请求会被忽略。只适合低负荷的服务器。

1. 多进程及多线程web服务器

可以创建多个线程/进程，同时处理多个请求，自选消耗高

③复用I/o的服务器

同时监控所有连接上的活动。对连接状态改变（数据可用，出现错误），就对那条连接进行少量的处理。处理结束之后，将连接返回到开发连接列表中。

在空闲连接上等待的时候并不会绑定线程和进程。

④复用的多线程web服务器

理由计算机上多个cup。多个线程中每一个都在观察打开的链接。

## 处理请求

。。。

## 访问资源

对资源的映射及访问：

将请求报文中的uri映射为web服务器上适应的内容或内容生成器，以识别出内容的源头。

不同类型的资源映射：

1. docroot

document root + uri

根目录 + uri

虚拟docroot

可以通过uri 或 host首部 中的ip 分别识别 根目录。

用户主目录docroot

在web服务器上为人们提供私有的web站点

②目录列表

可以将路径解析为目录，然后查找目录中的index.html文件。

③动态内容资源映射

映射到 按需动态生成的内容的程序

## 构建响应

响应实体（如果有的话）

content-type （mime类型）

服务器用以将MIME类型和资源关联的方法

①扩展名MIME类型相关

服务器根据一个包含所有扩展名的MIME类型的文件，确定文件MIME

②magic typing

服务器根据一个已知的模式表，对资源的内容进行扫描。从而匹配获取MIME类型。

③显示分类

对服务器进行配置，使其不考虑文件的扩展名和内容，强制特定文件或者目录内容拥有某个MIME类型。

④类型协商

服务器进行配置，可以以多种文档格式类存储资源。可以配置服务器，使其可以通过与用户协商决定使用哪种格式。

content-length （实体长度）

主体内容

## 发送响应

对于非持久连接的处理，发送完整条报文后，关闭自己这一端的链接。

对于持久连接，连接可能仍然保持打开的状态，这种情况下，服务器需要特别注意：

需要正确的计算content-type 首部，否则客户端无法知道响应什么时候结束。

## 记录事务处理过程

# 代理

## 公共与共享代理

公共代理：

集中式代理成本效率更高，易管理。

对于高速缓存代理服务器，会利用用户共同的请求。

私有代理：

## 代理与网关

严格来说，代理连接的是两个或多个使用相同协议的应用程序

而网关连接的则是两个或者多个不同协议的端点（扮演协议转换器的角色）

实际上：代理与网关的定义很模糊，由于：

服务器与浏览器使用的http版本不同

ssl安全协议

socks防火墙

ftp访问

基于web的应用程序

商业化代理服务器也会实现网关的功能来支持这些。

## 代理的作用

web缓存

反向代理

内容路由器

转码器

匿名者

# 缓存

## 缓存命中与新鲜度验证

缓存命中

已有缓存文件命中了发起的请求。直接使用缓存。

新鲜度验证（http再验证）

对于未命中的文件，到服务端发起新鲜度验证，仍然是新鲜的，使用缓存（返回304）。注意：如果服务端目标文件被删除，则会返回404，客户端缓存也会被清除。

缓存在验证头部工具：

最常用的：

请求头：If-Modified-Since

返回头：Last-modified

文件最后修改时间：但是服务端渲染的文件会被误认为被修改过了

If-None-Match 与 Etag

实体标签（版本标识符）：

Etag优先级更高，但是如果last-Modified与 etag同时存在。就必须同时满足才能返回304

## 缓存拓扑结构

私有缓存

如浏览器中缓存

共有代理缓存

多层级的代理缓存

第二级 --》 第三级

网状结构

不是简单的缓存层次结构，代理缓存间进行复杂的对话。做出动态缓存通信决策。

---内容路由

缓存中间这些更为复杂的关系允许不同的组织互为对等实体。

## 控制缓存

cache-control在请求头和返回头都可以使用：

pragma: no-store; // 兼容HTTP/1.0+

Cache-Control: no-store;

含义：

都是表面返回的内容是否应该被缓存，如何被缓存

no-store

禁止对响应进行缓存复制

no-cache

实际上响应可以被缓存，只是在与原始服务器进行新鲜度进行验证之前，缓存无法提供给客户端使用。

max-age

服务器将文档传入之时起，缓存处于新鲜的状态的时间。

s-maxage

作用域公共缓存（与max-age类似）

must-revalidate

如果本地缓存过期，在与原始服务器验证之前，不能使用过期缓存。

expires

过期时间--不要使用

## 缓存的处理步骤

接收--缓存从网络中读取抵达的请求报文。

# 客户端识别与cookie

## 客户端识别的几种方式

HTTP首部（承载用户信息）

Form

用户email

User-Agent

浏览器信息

Referer

从哪个页面调到现在的页面

Authorization

auth登录

服务器向浏览器发送一个401的 Login Required

登陆后每次这个首部都发送用户信息。

用户名和密码

Client-IP

客户端IP地址

X-Forwarded-For

客户端IP地址

Cookie

客户端IP地址跟踪

用户登录

胖URL

路径中记录用户的身份的识别码

无法分享url，破坏缓存，无法直接敲击url进入

cookie

大多数缓存和浏览器都不允许对任何cookie进行缓存。

# cookie

cookie类型

会话cookie

设置了discard 或者没有设置expires 或 max-age 参数 ，即是会话cookie

持久cookie

cookie的域属性

服务端响应通过Set-cookie属性写入cookie，通过domain属性限定域，通过path限定请求路径。

版本为0的cookie返回首部属性

name=value

expires

domain

path

secure

必须通过SSL安全连接才能发送cookie

# 接口设计原则：

对所接收的内容宽松一点（预留多种可能），对所发送的内容严格一点。