**How does HTTP protocol work?**

**1. Foundation**

Diagram

Description automatically generated

HTTP (Application Layer): 超文本传输协议, 在A和B之间传输HTML的一种规则.

**2. TCP/IP (协议族的总称) 和 HTTP**

**TCP/IP:**Text

Description automatically generated

应用层: 向用户提供的应用服务. 实例：FTP, DNS, HTTP.

传输层: 提供处于网络连接中的两台计算机数据传输功能. TCP, UDP

网络层: 处理流动的数据包, 数据包是网络传输的最小单位.

链路层: 硬件连接部分. 实例：设备驱动, 光纤, 网卡NIC(Network interface Card)

在发送端, 每层协议都会在原来的基础上加上自己的头部信息; 在接收端, 数据会从下往上把下层头部信息删除.

Diagram, engineering drawing

Description automatically generated

Graphical user interface, diagram

Description automatically generated

TCP: 三次握手 (1. SYN, 2. SYN+ACK, 3. ACK) 目的在于确认双方收发功能都是正常的.

DNS: 优先解析系统本地的hosts文件, 可在本地hosts中建立映射.

CDN: 加速访问. 拥有多个节点, 将静态文件cache out to servers across the world, 根据访问者location访问最近的CDN节点. 会有文件资源pull 和 push 的操作.

HTTP是基于TCP/IP协议的一个子集. HTTP请求之前需要先建立TCP连接 (三次握手); 握手成功后, 浏览器向服务器发送http请求，请求数据包; 服务器收到处理的请求 , 将数据返回至浏览器; 浏览器收到HTTP响应后进行解析, 如果响应可以缓存, 则存入缓存; 浏览器发送请求获取嵌入在HTML中的资源(CSS, JS, Video, Pic, Text etc), 对于未知类型, 会弹出对话框; 直至页面渲染结束; 访问结束后释放TCP连接.

**3. HTTP协议**

支持client/server模式; 多种请求方法get, post, put, delete; 通过Content-Type支持多种数据类型对象; 无连接, 无状态 (很健忘, 通过Cookies, Session解决这问题)

**请求头：**



Accept: text/html 浏览器可以接受的媒体类型, 如果服务器无法返回text/html类型的数据, 服务器返回406 错误 (Non Acceptable). Accept: \*/\* 接受所有类型 可以通过参数p=1 设置返回优先级 越大权重越高

Accept-Encoding: 浏览器申明自己的编码方法, 通常指定压缩方法, 支持什么压缩方法 (gzip, deflate).

Accept-Language: 浏览器申明自己接受的语言 Accept-Language: zh-cn,zh; q=0.7, en-us,en; q=0.3

Connection: keepalive 网页打开后, client和server用于HTTP传输的TCP链接不断开, 如果client再次访问, 会继续使用这条已经建立的连接. Close 代表request完成后, client和server之间用于HTTP数据传输的TCP连接会关闭. 再次发送request需要重新建立TCP连接.

Host: 包含被请求资源的域名主机和端口号 Host: [www.flij.com:8080](http://www.flij.com:8080)

Referrer: 从哪个网址连接转跳过来

User-Agent: 用户的操作系统, 浏览器名称和版本

Content-Type: 说明了请求体内对象的媒体类型 application/json, application/xml, application/pdf, application/msword.

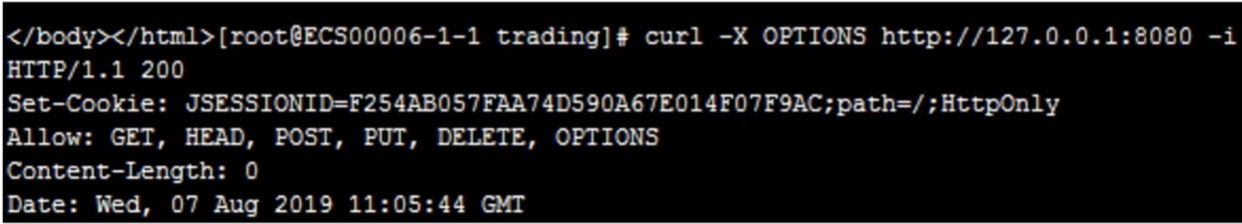
**响应头：**



**请求方法：**

HEAD: 类似于GET, 不过返回的响应中没有响应体, 用于获取头部信息 可用于测试超链接的有效性 超链接探测工具大多都基于HEAD. (拨测平台)

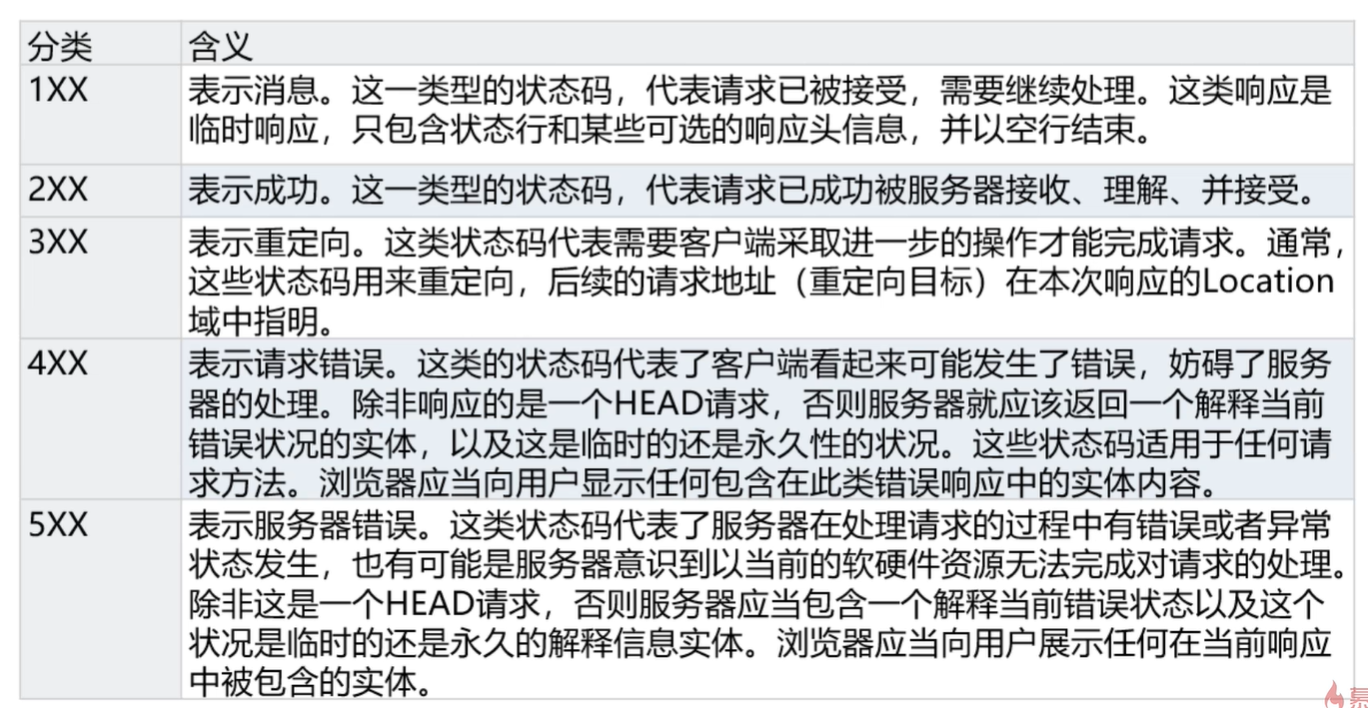
OPTIONS: 查询指定的资源支持的方法.

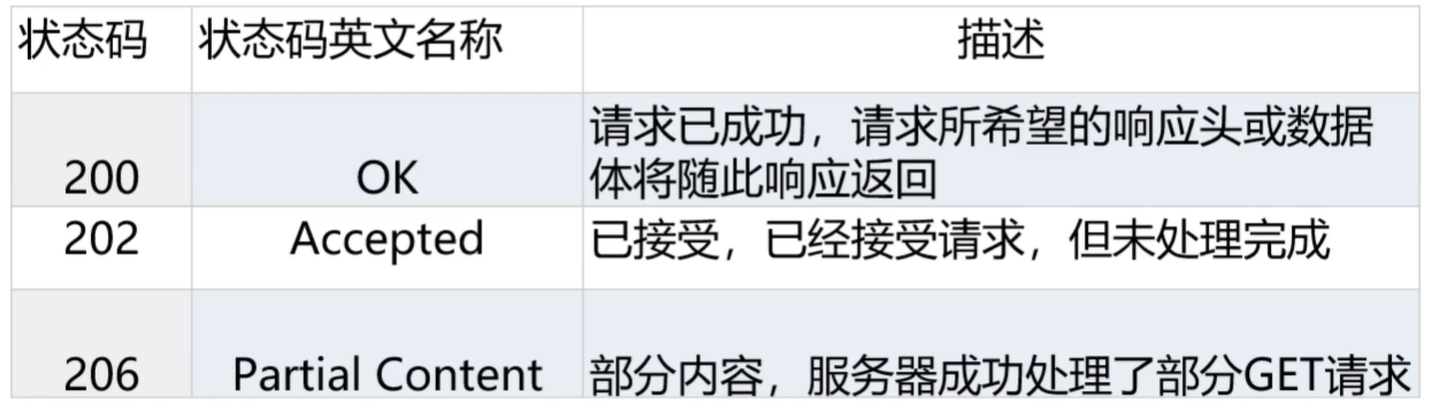


TRACE: 回显服务器收到的请求, 用于测试或诊断

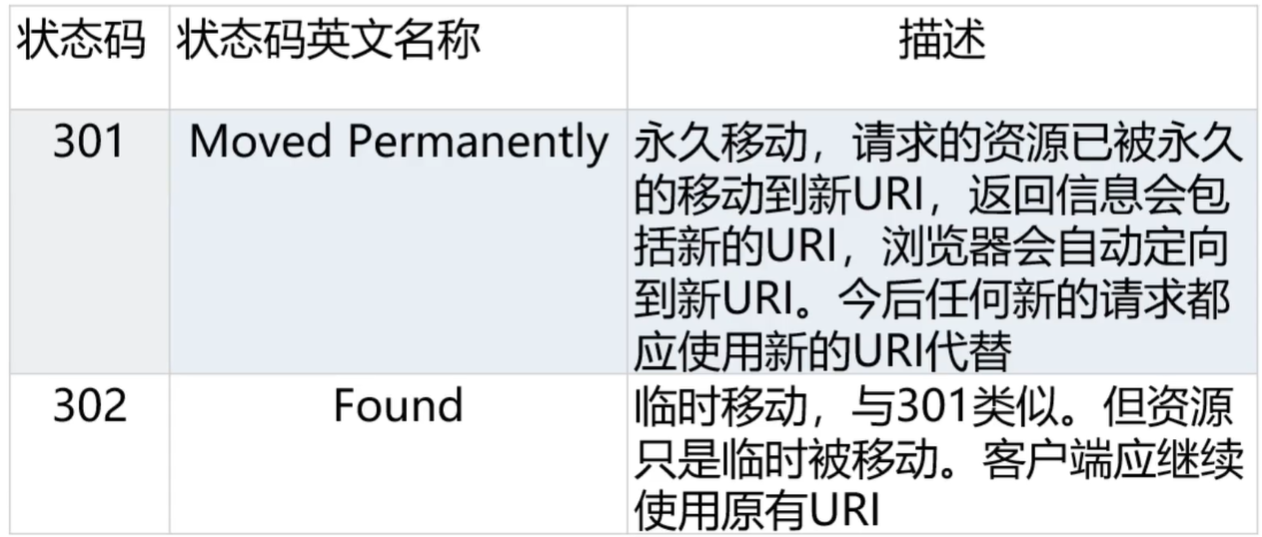
CONNECT: 开启一个client和server的双向沟通通道, 可用来创建隧道. 比如通过HTTP代理方式访问Facebook

**状态码：**





206 HTTP断点续传



304 Not modified. 页面已缓存, 响应结果和缓存相同

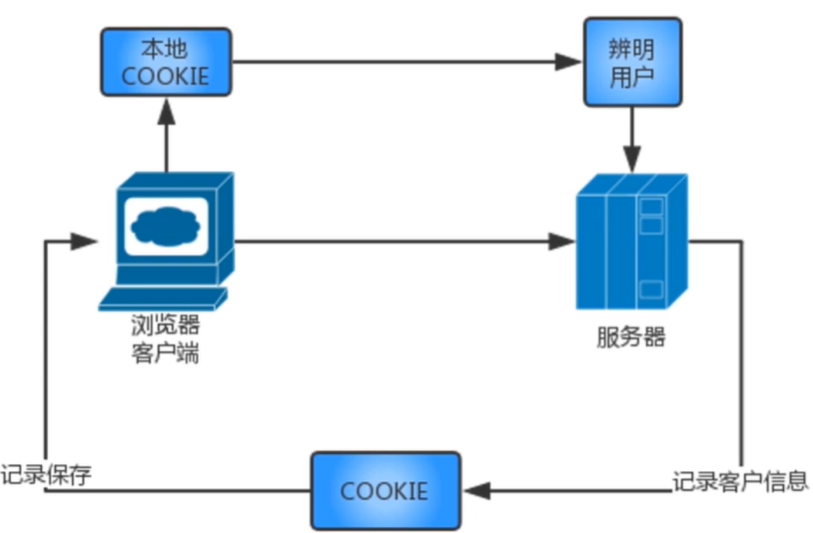




**状态管理Session和Cookies：(会话跟踪技术)**

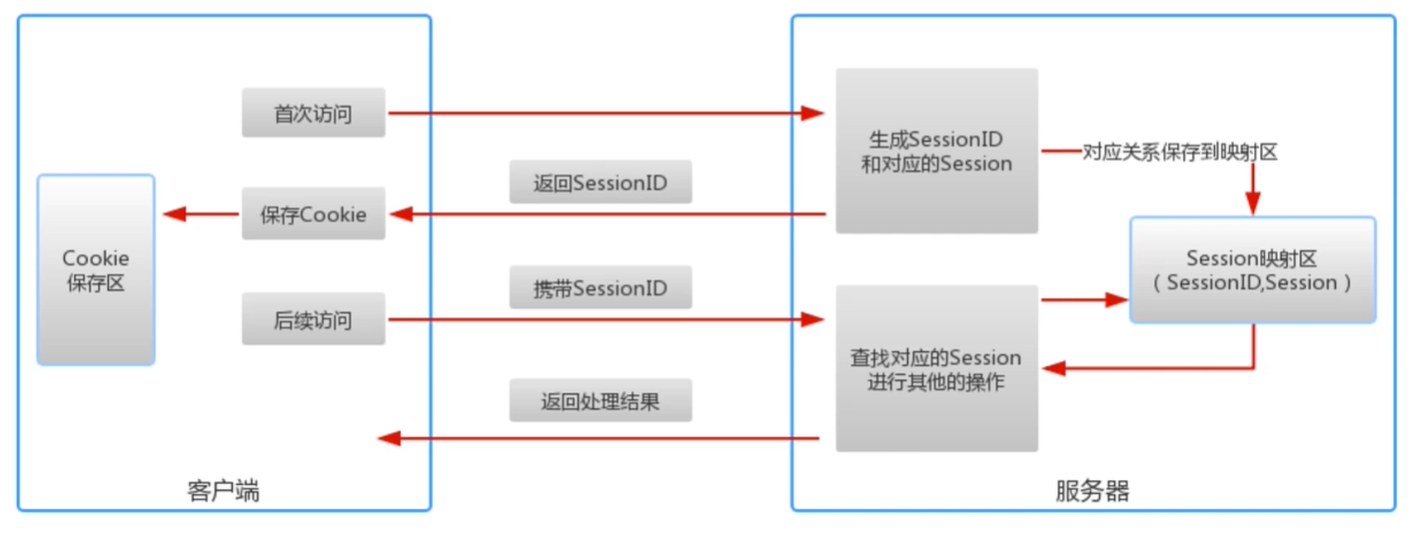
**Cookie:** 客户端请求服务器, 如果服务器需要记录client状态, 就向client颁发一个cookie. Client会把cookie保存起来, 当client再次请求时, client会把网址和cookie一同提交给server. Server通过检查该cookie, 以此来辨认用户状态.

浏览器域名输入Javascript:alert(document.cookie) 可查看cookie信息.



Server检索client是否含有对应cookie\_id并在db里检索, 没有则颁发一个cookie\_id并写入库, client将cookie信息保存到本地; 重复访问时client会把获得的cookie和url发送到server, server检索cookie\_id并将相关的业务操作记录到db, 有必要的话也会刷新cookie信息.

**Session:** 另外一种记录客户状态机制, 服务器把client的信息以某种形式记录在服务器上面.



Session\_id 在服务端是唯一不重复的, 利用cookie来保存session\_id. 假如cookie被禁用, 则使用URL重写, 把id附加在URL末端或者以参数形式来传递. 又或者使用隐藏表单, 在表单中添加session\_id.

**Session时效性：**

1. 为了防止内存溢出, session会有时效性, 把长时间不活跃的session给删除掉.
2. 程序也可以主动调用HttpSession.invalidate() 来删除session.
3. 服务器进程被停止

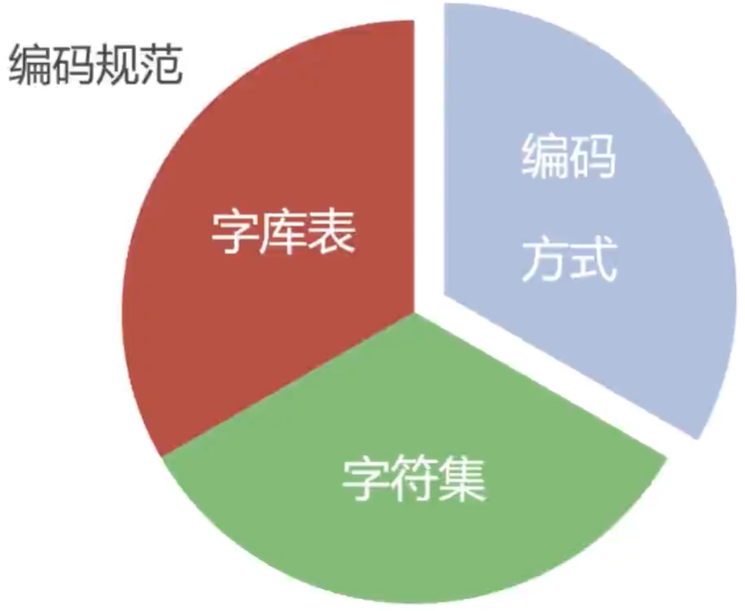
Session, Cookie的差异  
(1). 存放位置不同 Cookie在客户端，Session在服务端

(2). 安全性不同 cookie信息最好加密

(3). 有效期不同 服务器会定期删除session

(4). 对服务器压力不同 并发量大的时候, 服务器内存session占用大

**HTTP编码和解码：**



字库表 (比如在gbk规范中含有所有的中文文字)与字符集 (二进制数)一一对应, 比如ASCII 的 A (字库表) 为65, 对应字符集为65 的[binary](https://www.dict.cc/?s=binary) [system](https://www.dict.cc/?s=system) 100000001. 总之就是一个较短的二进制数通过编码方式转化成一个字符集中的地址, 再通过字库表找到对应的字符.

常见编码规范：ASCII, GBK, ISO-8859-1, Unicode(32字符集). Utf-8是一种编码方式, 英文字母为1byte, 中文为3bytes.

**乱码：**

编码和解码使用的方式不一致或者编码的时候它的字库表里不包含相应字符. URL采用ASCII字符集进行编码, 如果含有非ASCII字符集中的字符, 则需要对其编码 (比如 ‘&’).

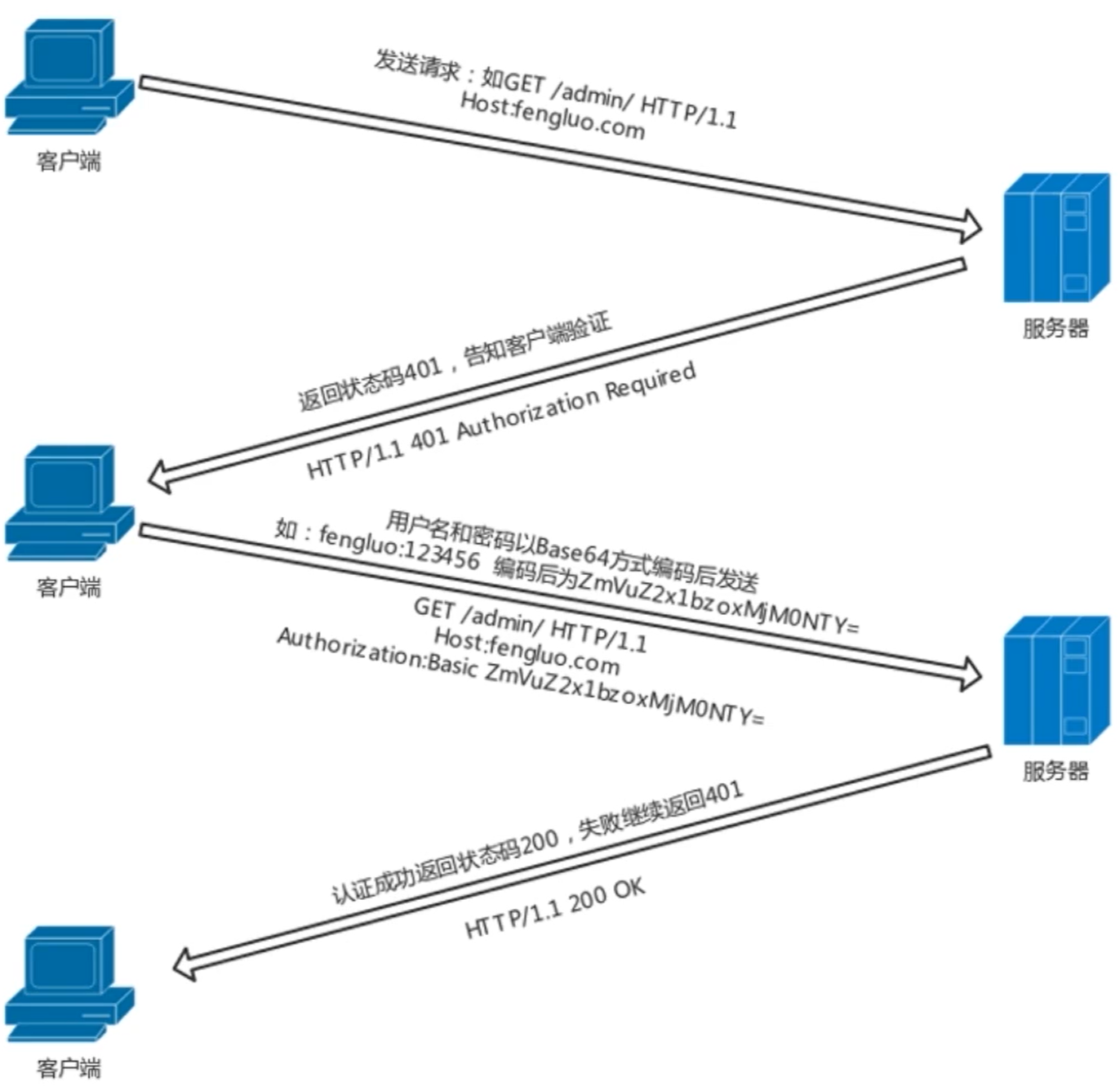
浏览器只会对URL中的非ASCII字符集进行编码 (比如 参数username包含中文, 则中文会被转码成 %E9%A3……) (更多infos可用Fiddler抓包工具进行测试, 启动后访问相应链接 查看请求头和体, 响应头和体信息)

**HTTP协议基本认证：**

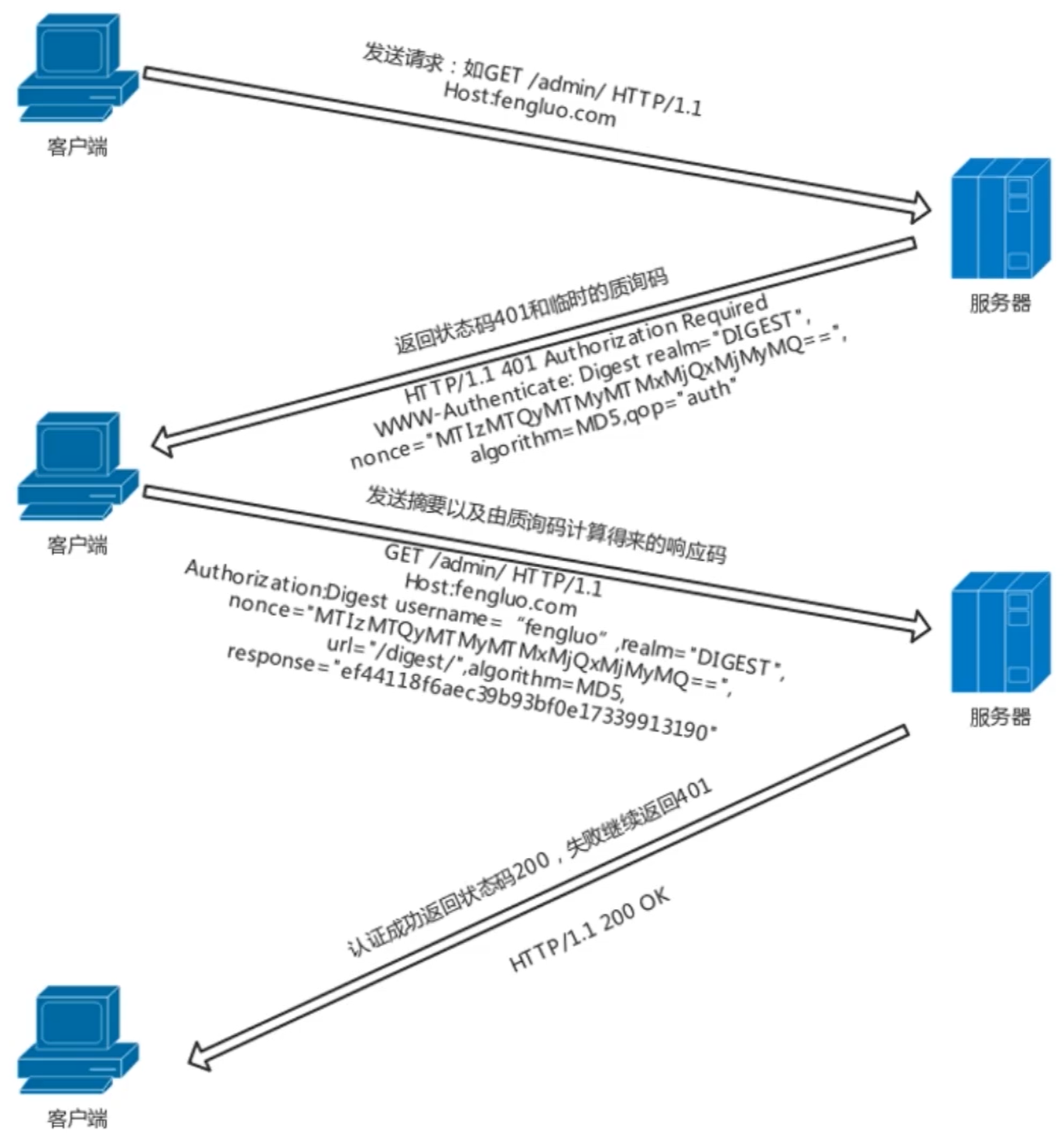
身份认证：密码, 动态令牌, 数字证书, 生物认证, IC card etc.

**常用认证方式：**

1. BASIC认证：采用Base64的编码方式.



1. DIGEST认证方式: 同样使用质询/响应 (质询码) 的方式, 但不会像BASIC认证那样直接发送明文密码.



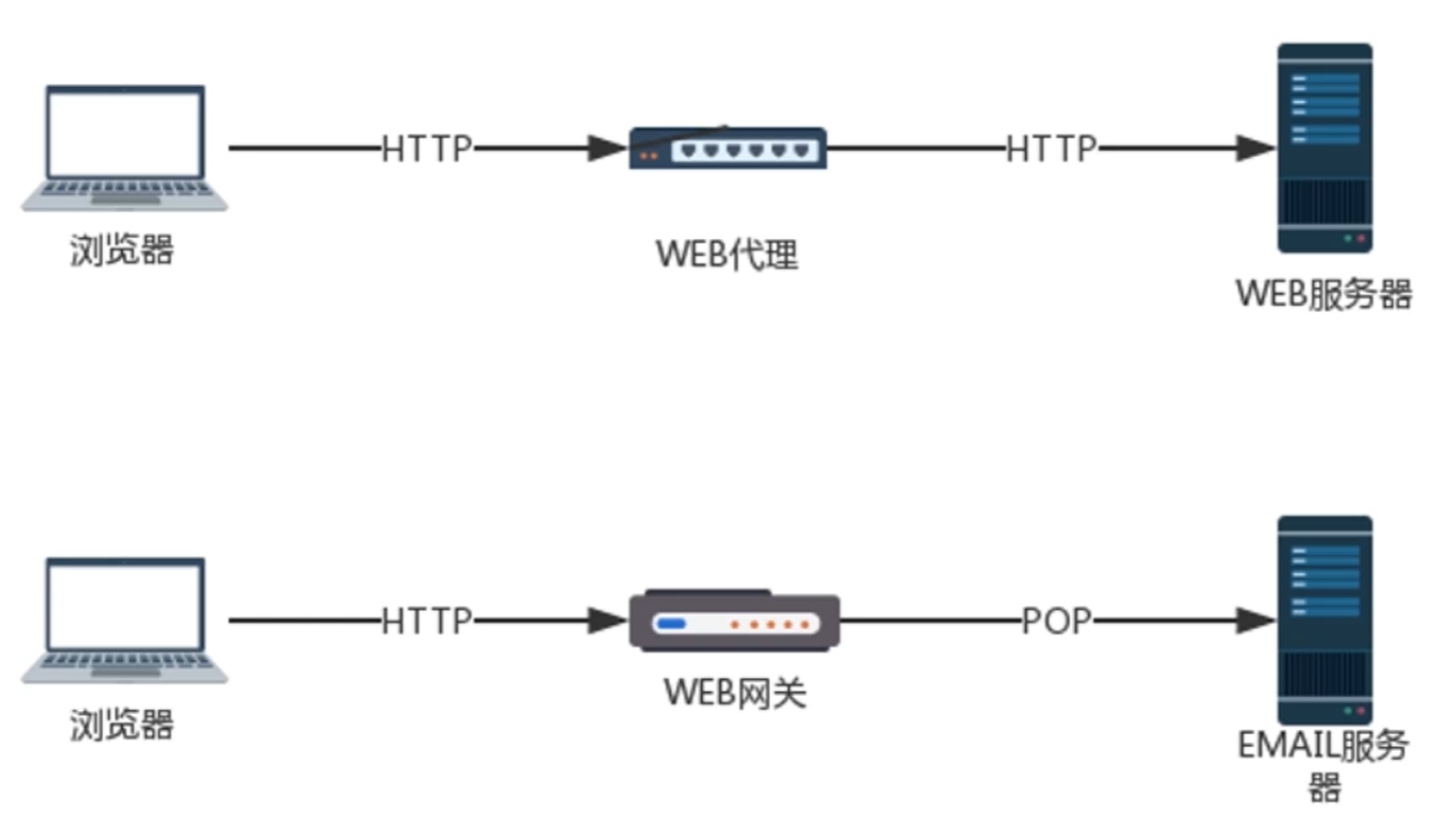
1. SSL客户端认证: 借由HTTPS的客户端证书完成认证的方式. (ELB中有证书 过ELB变成HTTP 华为内网连接https://域名... 自动不加密变成http同样可以访问 但是外网是禁止不加密的)
2. 基于表单的认证: 使用web应用程序各自实现基于表单的认证方式. (Cookie & Session)

**HTTP长连接和短连接：**直观上就是地铁线路和打的士的区别, 长连接能节约很多资源和多次复用. 短连接表示一锤子买卖. (本质上是TCP的长连接和短连接)

HTTP/1.1, 默认使用长连接, 长连接更普遍, 节省建立连接的时间和避免同一个客户端多次访问建立多个连接占用带宽资源. 但在服务器端则需要对连接进行优化, 定期删除不常用的连接.

长连接又叫做持久连接, HTTP/1.0 使用非持久连接, 而HTTP/1.1 默认使用持久连接. HTTP协议的初始版本中, 每进行一次HTTP通信就要断开一次TCP连接. 以当年的通信情况来说, 因为都是些容量很小的文本传输, 所以即使这样也没有多大问题. 随着 HTTP 的 普及，文档中包含大量图片等其他资源。比如，使用浏览器浏览一个包含多张图片的 HTML 页面时，在发送请求访问 HTML 页面资源的同时, 也会请求该 HTML 页面里包含的其他资源. 因此, 每次的请求都会造成无谓的 TCP 连接建立和断开, 增加通信量的开销. 为解决上述 TCP 连接的问题, HTTP/1.1 和一部分的 HTTP/1.0 想出了持久连接（HTTP Persistent Connections, 也称为 HTTP keep-alive 或 HTTP connection reuse）的方法. 持久连接的特点是, 只要任意一端没有明确提出断开连接, 则保持TCP连接状态. 持久连接的好处在于减少TCP 连接的重复建立和断开所造成的额外开销, 减轻了服务器端的负载.减少开销的那部分时间, HTTP 请求和响应能够更早地结束, Web 页面的显示速度也会相应提高.

**HTTP中介 (Fidder(Proxy))：**



**代理(proxy):** 作用(抓包, FQ, 匿名访问(正向代理)), 过滤器)

**web网关(gateway):** 协议转换器, 连接多个使用不同协议的端点. 抽象理解为翻译, 将两个说不同语言的人能交流起来.

(HTTP/\*: 服务器端Web网关)

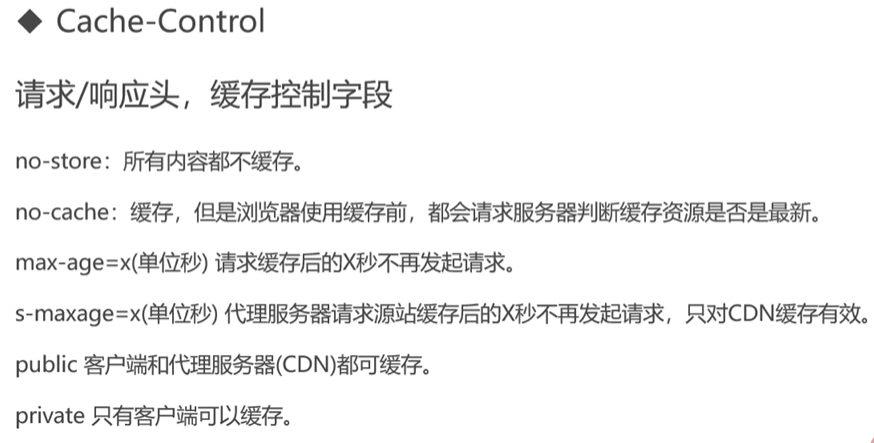
(HTTP/HTTPS: 服务器端安全网关)

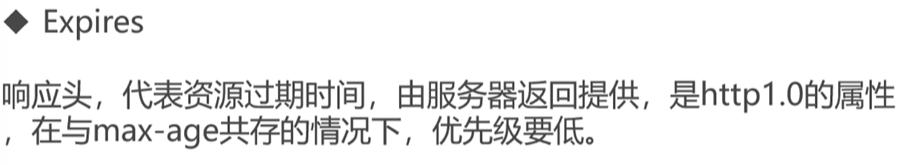
(HTTPS/HTTP: 客户端安全加速器网关 又称为安全网关) 类似ELB, 将HTTPS转换成HTTP, 即HTTPS入, HTTP内网调用

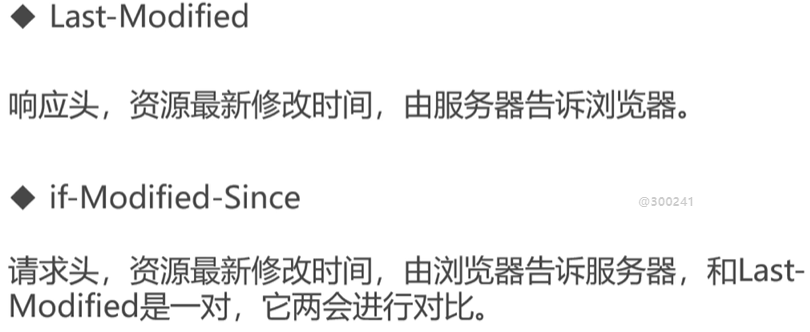
(资源网关: 如网络摄像头, 电子识别系统)

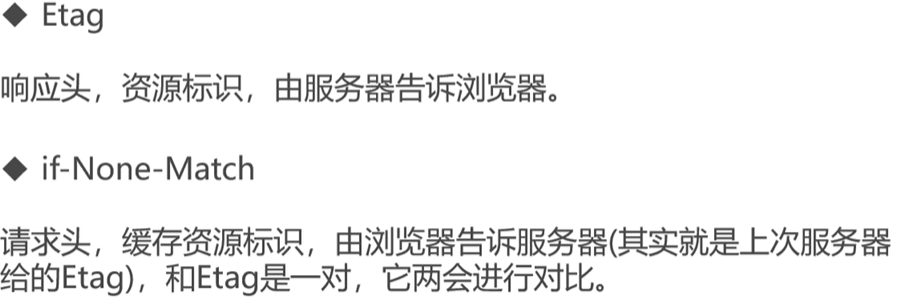
**HTTP缓存：**

主要缓存静态资源文件, 针对更新频率不高的文件, 加快页面二次或多次访问速度.









HTTP缓存工作方式：



通过对比上图里面两个字段判断文件标识是否相同. 304代表文件相同无修改, 继续使用本地缓存页面. (针对文件路径相同的情况)

MD5/hash缓存 (改进)：

不缓存html, 为静态文件添加MD5或者hash标识, 使文件变成两个完全不相同的文件, 解决浏览器无法跳过缓存过期时间 (max-age) 主动感知文件变化的问题. 用工具可为文件添加MD5或者hash标识

CDN缓存: (可以理解为建立多个春运火车票售票点, 避免主火车站出现买票挤兑拥堵现象)

服务器返回给CDN, CDN自己进行静态文件缓存, 同时CDN返回给浏览器, 解决跨地区访问问题. CDN缓存不过期, 浏览器始终被拦截, 无法拿到最新文件. 类似平台, 可以登录手动更新CDN缓存, 变向解决浏览器缓存无法手动控制问题.

**HTTP内容协商机制：**

客户端驱动: 客户端发起请求, 服务器发送可选列表, 客户端发送二次请求.

服务器驱动: 服务器检查客户端的请求头决定提供哪个版本.

Accept:mp3 -> Content-Type

Accept-Language: zh-cn:q=0.5, fr;q=0.0(不接受), en;q=1.0 (q代表优先级) -> Content-Language //如果en, cn都没有 一般服务器会设定一个默认值返回.

Accept-Charset: unicode (字符集) -> Content-Type

Accept-Encoding: utf-8 (编码方式) -> Content-Encoding

透明协商: 某个中间设备, 通常是缓存代理.

**HTTP断点续传和多线程下载：**

HTTP在Header里两个参数实现, 请求Range和响应对应的Content-Range.

Range: (unit=first byte pos) - [last byte pos]

Content-Range: bytes (unit first byte pos) – [last byte pos]/[entity length]

HTTP/1.1 200 ok (不使用断点续传方式)

HTTP/1.1 206 Partial Content (使用断点续传方式)

Beispielsweise:

1. 下载一个1024K的文件, 已经下载512k
2. 网络中断, client请求续传 Range: bytes=512000-
3. 服务器收到请求, Content-Range: bytes=512000-/1024000

**多线程下载**：主动分片下载, 同样使用range, 比如分成100片, 第一个片0-1024000，第二个1024001-2048000…… 每一片只有1/100.

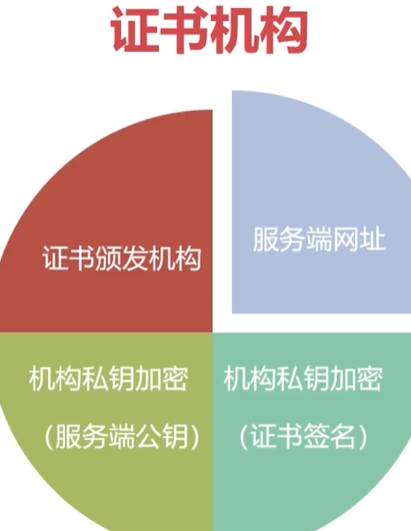
**HTTP简易请求服务器云端部署：**

**HTTP调用测试和抓包工具**

调用测试：Postman

抓包测试：Fiddler

**HTTP和HTTPS:**



**对称加密:** 双方使用同一个秘钥 比如凯撒加密 约定双方秘钥为偏移量5

**非对称加密:** 两把不同的钥匙, 用一把钥匙上锁, 然后只能用另外一把钥匙开锁, 这两把钥匙分别称为公钥和私钥. 公钥能解密私钥加密的数据, 私钥也能解密公钥加密的数据 (RSA加密算法双向). 且公钥私钥是配对生成的, **私钥拥有者=公钥拥有者 (这点通过数字证书来确保)**.

比如A->B发送消息, B有自己的公钥和私钥, 公钥是公开的, 任何人都能获取. A用B的公钥加密数据, B再用私钥解开. 但是存在问题, 入侵者可以胡乱修改或删除加密报文的一部分.

**数字签名：**让接收方确定收到的消息来自所期待的发送方. 原理是让发送方独一无二的东西附加到报文上(类似签名). 在密码学中, 独一无二的签名就是秘钥, 用私钥对报文进行加密, 这种时候叫做文件的‘签名’, 然后将签名和plaintext（表示为m）一同发送给接收方 (m, ); 接收方收到后用发送方的公钥对签名进行解密, 并与收到plaintext进行对比. 如果相同则来自发送方 () =m, 保证了数据的完整性, 如果m被篡改为n, 则有() !=n

HTTPS: HTTPs=HTTP+TLS.

内容加密, 身份认证 (**数字证书：** (1). 包含公钥拥有者的信息 (2). 包含私钥加密过的公钥信息, 即包含拥有者信息和他的公钥. **数字证书由权威CA机构颁发, 能确切证明该公钥属于其拥有者.** 同时, 为了杜绝CA机构公钥被伪造的可能, CA机构会用秘钥对拥有者信息及其公钥数据这一文件内容进行数字签名, 现在大部分浏览器和操作系统都整合了各大权威CA机构的根证书, 根证书记录可信赖的CA机构信息及其公钥, 通过CA机构公钥可以验证数字证书里的数字签名, 确保数字证书(其他文件和数字签名的组合体)的不可伪造. **正式的证书需要绑定域名.** 通过哈希算法对plaintext生成一段特定长度的唯一哈希值, 这个过程不可逆 [要找到两个不同m和n, 使得H(m)=H(n)在计算上是不可行的]; 常用哈希算法有 MD5, SHA-1, SHA-256 etc.

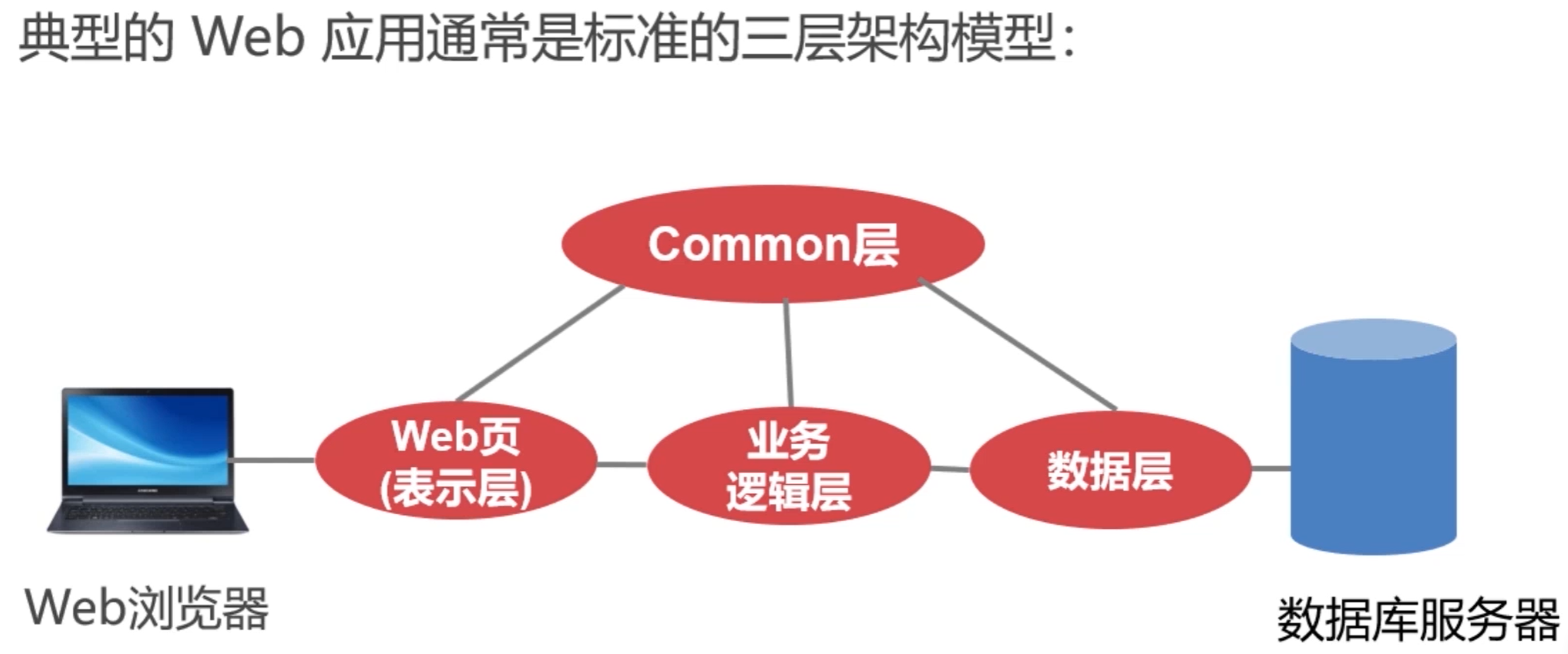
更多可以了解：https://www.bilibili.com/video/BV18N411X7ty

HTTPS对性能的影响：

增加交互的网络RTT (Round Trip Time), 加解密计算耗时, 浏览器计算耗时(包含浏览器解析证书签名, 各种秘钥交换, 应用层数据加解密, 一致性的交换), 服务端计算耗时. HTTPS优化问题(进阶), 比如TCP fastopen, HSTS, etc, 包含TCP, 会话层的优化.

HTTPS的证书也能自己生成, 生成后需要在Browser中导入安装, 但Browser会有不安全提示.

**Web安全威胁解析：**



WASC组织奖Web应用安全威胁分为6大类:

1. Authentication
2. Authorization
3. Client-Side Attacks
4. Command Execution
5. Information Disclosure
6. Logical Attacks

**验证机制：**

(1). Cookie会话验证: failedlogin=0 尝试登陆失败次数. Cookie在客户端, 包含在头部当中, 未传递到服务端.

(2). 双因子认证: 密码+手机双重认证.

(3). 多阶登陆机制

**会话管理：**

在网络安全中需要关注Session生存, Cookie写入, Cookie读取.

会话终止攻击, 应用程序退出并没有与服务器通信交互, 服务器不采取行动.

1. 令牌只能通过HTTPS传送
2. 增加软硬会话过期 软会话, 一定时间没有交互自动过期退出; 硬会话, 到达特定时间强制退出.
3. 完善logout功能

**SQL注入原理:**

Beispielsweise: 登陆账号: ’or’ 1=1 -> Select count(\*) from user where username is null or 1=1. 恒等, 直接返回数据库第一条信息.

参数化查询: 指定查询结构, 用户输入预留占位符, 指定占位符的内容.

**XSS (Cross Site Scripting) 跨站脚本攻击原理:**

允许恶意web用户将代码植入到提供给其他用户使用的页面中, 其他用户在观看网页时, 恶意脚本就会执行; 通常通过注入HTML或JS等脚本发动攻击. 请求端能够把恶意脚本提交到response里面.

反射式XSS：出现在服务器直接使用客户端提交的数据. 如url的数据, html表单中提交数据.

存储式XSS：将攻击脚本上传到web服务器 (Myspace samy事例), 使得所有访问用户都面临信息泄露的可能, 包括admin.

基于DOM的XSS：通过客户端javascript动态生成信息, 不会跟服务器交互.

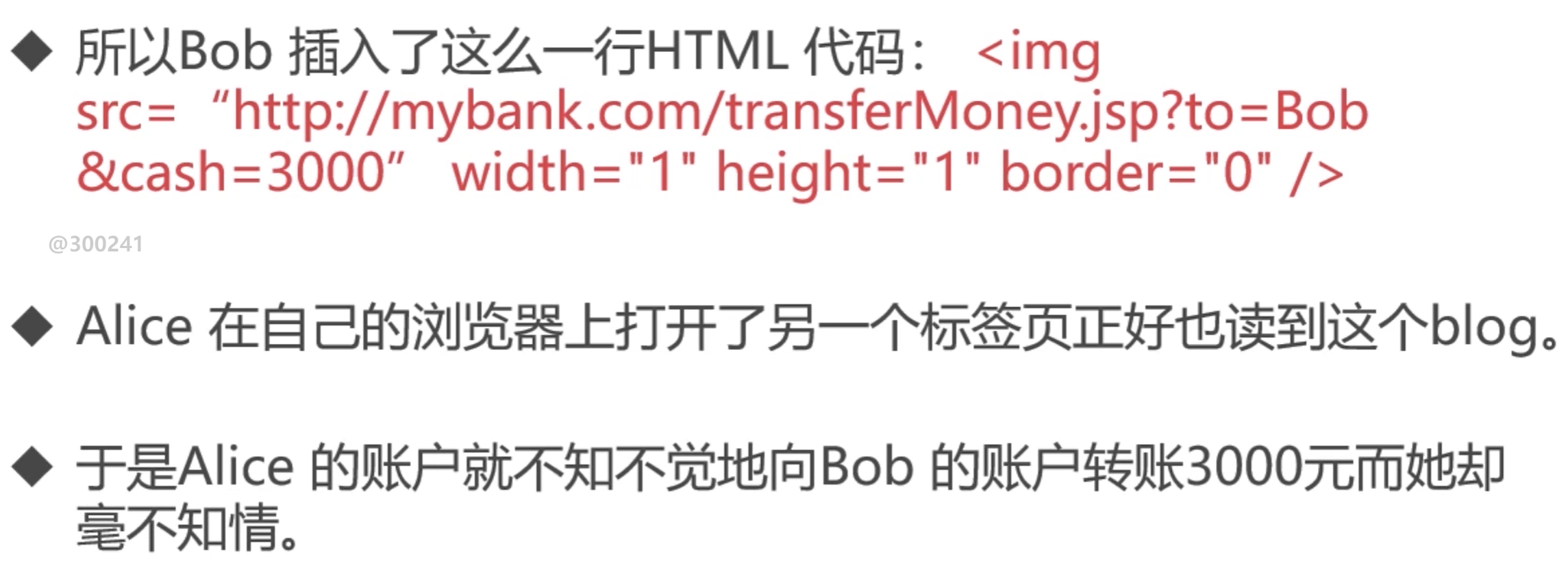
XSS攻击载荷：

1. 会话令牌: 截取令牌, 劫持会话, 任意操作并占有该用户的账号.
2. 虚拟置换: 在页面注入恶意数据, 提示服务器在维护, click here转跳到陌生网页, 传递误导信息.
3. 注入木马:

**CSRF (Cross-site Request Forgery) 跨站请求伪造：**

XSS利用站点的信任用户, CSRF伪装成受信任用户的请求来利用网站. 事例如下:





浏览器可以访问会话管理信息, 如果web应用完全依赖这类信息来识别用户, 容易为CSRF创造条件, web应用不会判断是否是合法用户发送 (session 已在服务器注册).

1. 确认操作
2. 重新认证
3. 使用Token: 在需要保护的表单中, 增加一个隐藏字段来存放Token, 用于校验服务器端Token和用户session中的Token是否一致. 此Token和session在用户第一次访问页面时初始化 simultaneously