在浏览器中输入url地址->显示主页的过程

打开一个网页，整个过程会使用到那些协议

过程	使用的协议
1. 浏览器查找域名的IP地址 (DNS查找过程：浏览器缓存、路由器缓存、DNS 缓存)	DNS：获取域名对应IP
2. 浏览器向web服务器发送一个HTTP请求 (cookies会随着请求发送给服务器)	• TCP：与服务器建立TCP连接 • IP：建立TCP协议时，需要发送数据，发送数据在网络层使用IP协议 • OPSF：IP数据包在路由器之间，路由选择使用OPSF协议 • ARP：路由器在与服务器通信时，需要将ip地址转换为MAC地址，需要使用ARP协议 • HTTP：在TCP建立完成后，使用HTTP协议访问网页
3. 服务器处理请求 (请求 处理请求 & 它的参数、cookies、生成一个HTML响应)	
4. 服务器发回一个HTML响应	
5. 浏览器开始显示HTML	

具体的过程就是：

1. 浏览器查找域名的IP地址；DNS查找过程：浏览器缓存，路由器缓存，DNS缓存；
2. 浏览器向web服务器发送一个HTTP请求；cookies会随着请求发送给服务器；
3. 服务器处理请求；请求，处理请求，他的参数，cookies，生成一个HTML响应；
4. 服务器发回一个HTML响应；
5. 浏览器开始显示HTML；

上述过程中涉及的协议是：

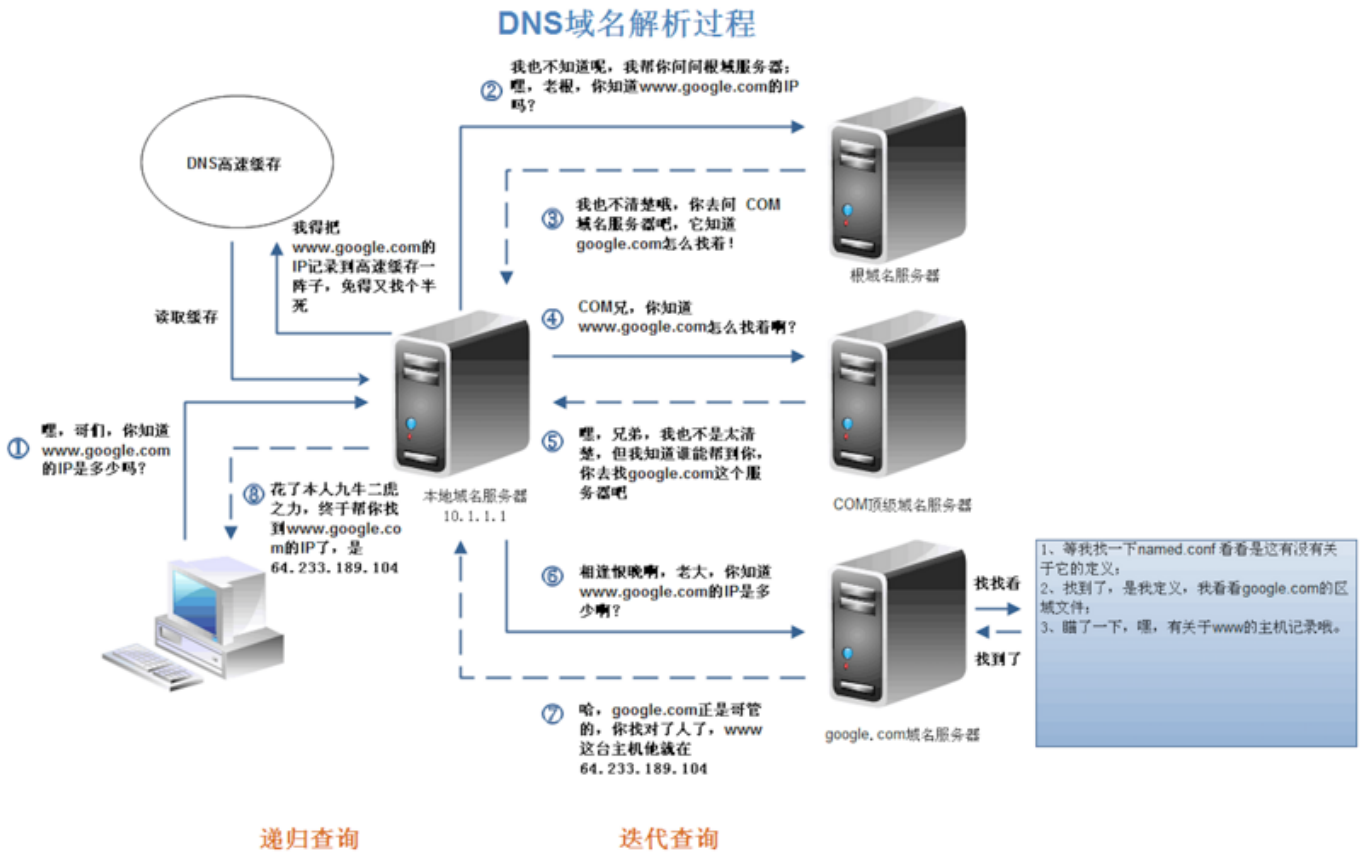
1. DNS：获取域名对应的IP；
2. TCP：与服务器建立TCP连接；
3. IP：建立TCP连接时，需要发送数据，发送数据在网络层使用的IP协议；
4. OPSF：IP数据包在路由器之间，路由选择需要使用OPSF协议；
5. ARP：路由器与服务器通信时，需要将IP地址转换成MAC地址，需要使用ARP协议；
6. HTTP：在TCP建立之后，使用HTTP协议访问网页；

总的来说，分为以下几步：

1. DNS解析
2. TCP连接
3. 发送HTTP请求
4. 服务器处理请求并返回HTTP报文
5. 浏览器解析渲染页面
6. 连接结束

DNS具体的解析过程

形象的解析过程如下图：



查找www.google.com的IP地址过程；

1. 本地域名服务器上查询IP地址，有的话直接返回，没有的话向根域名服务器发送请求；
2. 根域名服务器上查询IP地址，有直接返回，没有的话，本地域名服务器向com顶级域名服务器上发送请求；
3. 以此类推，知道最后本地域名服务器得到google的IP，并将其缓存在本地；

域名解析过程就是：.->com.->google.com.->www.google.com.; 上述过程中有一个点,其实不是多打了一个.，这个.对应的就是根域名服务器，默认情况下所有的网址的最后一位都是.，既然是默认情况下，为了方便用户，通常都会省略，浏览器在请求DNS的时候会自动加上。

DNS优化

DNS域名解析使用的是UDP还是TCP?

DNS在区域传输的时候使用TCP协议； 区域传输就是：DNS的规范规定了2种类型的DNS服务器，一个叫主DNS服务器，一个叫辅助DNS服务器。在一个区中主DNS服务器从自己本机的数据文件中读取该区的DNS数据信息，而辅助DNS服务器则从区的主DNS服务器中读取该区的DNS数据信息。当一个辅助DNS服务器启动时，它需要与主DNS服务器通信，并加载数据信息，这就叫做区域传输；

区域传输为什么使用TCP呢?

1. 辅助域名服务器会定时（一般3个小时）想主域名服务器进行查询一遍了解数据是否变动，如有变动则会执行一次区域传输，数据同步，区域传输使用TCP，因为数据同步传输数据量比一次请求和应答的数据量大很多；
2. TCP可靠，可以保证这部分数据准确；

普通域名解析时使用UDP协议； 客户端向DNS服务器查询时，一般返回的内容都不超过512字节，使用UDP传输即可，不用经过TCP三次握手，DNS服务器负载更低，相应更快；

DNS缓存

多级缓存：浏览器缓存，系统缓存，路由器缓存，IPS缓存，根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存；

DNS负载均衡

DNS可以返回一个合适的机器的IP给用户，例如可以根据每台机器的负载量，该机器离用户地理位置的距离等等，这种过程就是DNS负载均衡，又叫做DNS重定向。大家耳熟能详的CDN(Content Delivery Network)就是利用DNS的重定向技术，DNS服务器会返回一个跟用户最接近的点的IP地址给用户，CDN节点的服务器负责响应用户的请求，提供所需的内容。

状态码

	类别	原因短语
1XX	Informational信息性状态码	接收的请求正在处理
2XX	Success成功状态码	请求正常处理完毕
3XX	Redirection重定向状态码	需要进行附加操作完成请求
4XX	Client Error客户端错误状态码	服务器无法处理请求
5XX	Server Error服务器错误状态码	服务器处理请求出错

HTTP协议

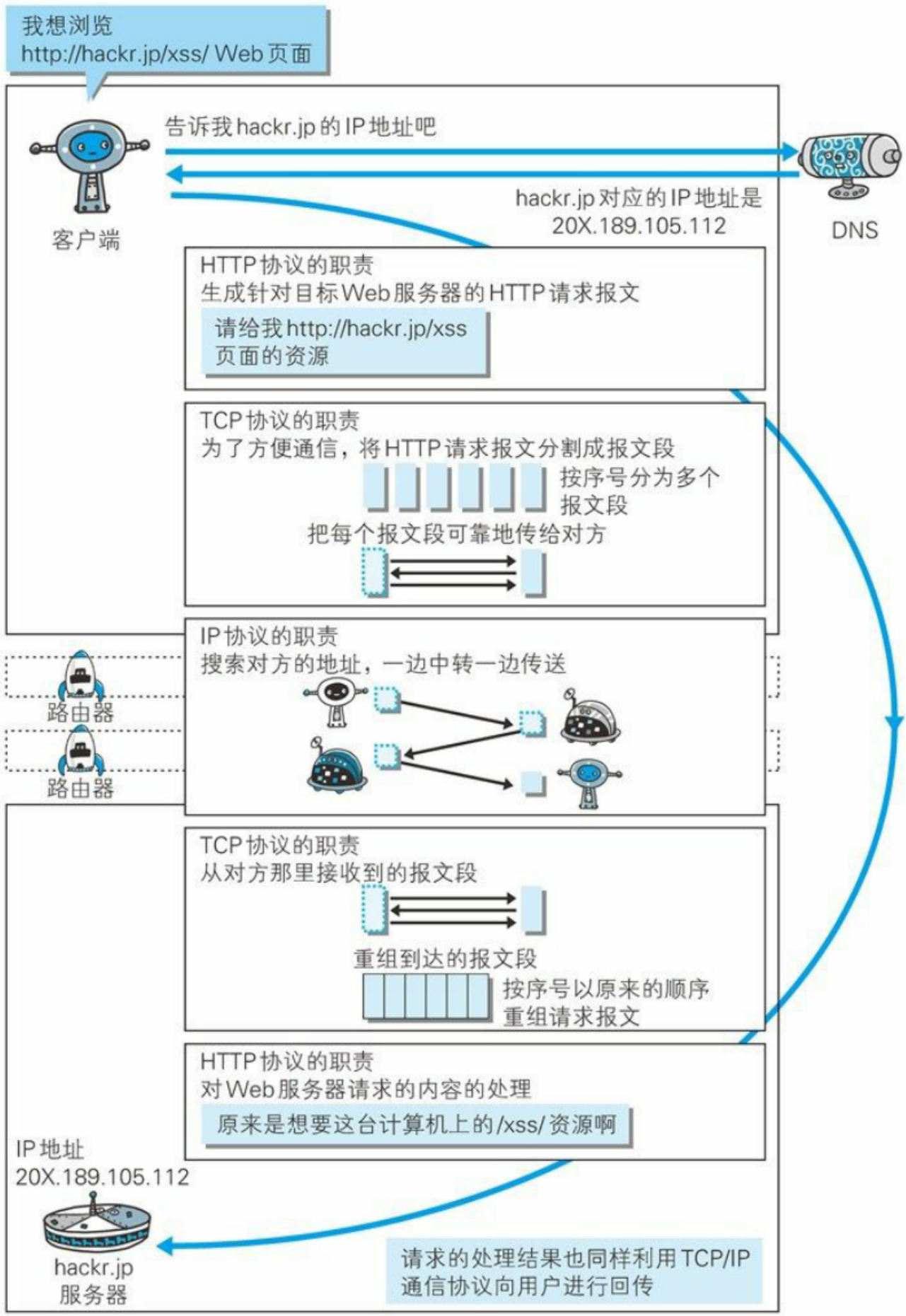
HTTP方法

客户端发送请求报文的第一行为请求行，包含了方法字段；

1. **GET** 获取资源； 当前网络请求中，绝大部分使用GET方法；
2. **HEAD** 获取报文首部； 和GET方法类似，但是不返回报文实体主体部分。 主要用于确认URL的有效性以及资源更新的日期时间等；
3. **POST** 传输实体主体； POST主要用来传输数据，而GET主要用来获取资源；

4. **PUT** 上传文件； 由于自身不带验证机制，任何人都可以上传文件，由于存在安全性问题，一般不用；
5. **PATCH** 对资源进行部分修改； PUT可以修改资源但是只能完全代替原始资源，PATCH允许部分修改；
6. **DELETE** 删除文件； 与PUT功能相反，并且同样不带验证机制；
7. **OPTIONS** 查询指定的 URL 能够支持的方法。
8. **CONNECT** 要求与代理服务器通信时建立隧道； 使用SSL和TLS将通信内容加密后经网络隧道传输；
9. **TRACE** 追踪路径； 服务器会将通信路径返回给客户端。 通常不用，容易受到XST攻击；

各种协议和HTTP协议之间的关系


我想浏览
http://hackr.jp/xss/ Web页面
告诉我hackr.jp的IP地址吧
客户端
DNS
hackr.jp对应的IP地址是
20X.189.105.112
HTTP协议的职责
生成针对目标Web服务器的HTTP请求报文
请给我http://hackr.jp/xss
页面的资源
TCP协议的职责
为了方便通信，将HTTP请求报文分割成报文段
按序号分为多个
报文段
把每个报文段可靠地传给对方
IP协议的职责
搜索对方的地址，一边中转一边传送
路由器
路由器
TCP协议的职责
从对方那里接收到的报文段
重组到达的报文段
按序号以原来的顺序
重组请求报文
HTTP协议的职责
对Web服务器请求的内容的处理
原来是想要这台计算机上的/xss/资源啊
IP地址
20X.189.105.112
hackr.jp
服务器
请求的处理结果也同样利用TCP/IP
通信协议向用户进行回传

简述上述过程：

1. 请求DNS服务器，DNS协议将域名转换为具体的IP地址；
2. HTTP协议：生成目标Web服务器的HTTP请求报文；（应用层）
3. TCP协议：将HTTP协议包裹在TCP报文中，分割成报文段，将每个报文可靠的发给服务端（传输层）
4. IP协议：在路由器上面搜索对方的地址，一边中转一边传送（网络层）（下面的链路层以及物理层对网络层透明）
5. TCP协议：从对方那里接受到的报文段，按照序号重组成原来的请求报文（传输层）
6. HTTP协议：对Web服务器请求的内容进行处理 最后请求的处理结果按照上述的过程同样的返回给用户；

HTTP长连接和短连接

在HTTP1.0中，默认使用短连接，也就是说，客户端和服务器端每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。客户端访问的某个HTML或者其他类型的Web页面中包含有其他的Web资源，每次遇到一个这个Web资源就要重新建立一个HTTP会话；

从HTTP1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头加入这行代码"

```
connection:keep-alive
```

在使用长连接时，当一个网页打开完成之后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时，会继续使用这个已经建立的连接，Keep-alive不会保持永久连接，有一个时间参数，可以设置；

HTTP的长连接和短连接实际上是TCP的长连接和短连接；

TCP短连接： lient向server发起连接请求，server接到请求，然后双方建立连接。client向server发送消息，server回应client，然后一次请求就完成了。这时候双方任意都可以发起close操作，不过一般都是client先发起close操作。上述可知，短连接一般只会在 client/server间传递一次请求操作。

TCP长连接： client向server发起连接，server接受client连接，双方建立连接，client与server完成一次请求后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接；

TCP的保活功能主要为服务器应用提供，如果客户端已经消失而连接并未断开，则会使得服务器上保留一个半开放的连接，而服务器有在等待来自客户端的数据，此时服务器将永远等待客户端的数据。保活功能就是试图在服务器端检测到这种半开放的连接；

如果一个给定的连接在两个小时内没有任何动作，服务器就向可获段发送一个探测报文段，根据客户机的响应，探测到4个客户端状态；

1. 客户端依然正常，服务器可达，此时TCP响应正常，服务器将保活定时器复位；
2. 客户机已经崩溃，并且关闭或者正在重新启动，上述情况下客户端都不能响应TCP，服务端将无法收到客户端对探测的响应，服务器总共会发送10个这样的探测相隔75秒，若一个响应都没收到，就认为客户端已经关闭并终止连接；
3. 客户端崩溃并已经重启，服务端将收到一个保活的响应，这个响应是一个复位，服务端终止这个连接；
4. 客户机正常，但服务器不可达，情况类似第二种情况；

长短连接，优缺点很明显，短连接对于频繁访问的客户不友好，每次连接占用很大带宽，长连接对于频繁连接的客户友好，但是容易出现恶意连接，拖垮后台的网络问题；

HTTP是不保存状态的协议，如何保存用户状态呢?

HTTP是一种不保存状态，即无状态的协议，也就是说HTTP协议自身不对请求和相应之间的通信状态进行保存； Session机制，就是为了解决这个问题，Session的主要作用就是通过服务端记录用户的状态，典型的场景就是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为HTTP协议是无状态的，服务端给特定用户建立Session就可以标识这个用户并且跟踪这个用户了（但是Session是有时间限制的，过了时间限制就会销毁这个Session）； 在客户端保存Session的方法有很多，一般就是内存和数据库，比如使用redis保存，Session保存在服务器端，那么我们如何实现Session的跟踪呢，大部分情况是在Cookies中添加一个Session ID来跟踪；

如果Cookies被禁用，最常见的方法是利用URL重写把SeesionID直接附加到URL后面；

Cookies和Session的作用与区别；

Cookies一般用来保存用户信息，比如：

1. Cookies中保存已经登录过的用户信息，下次访问直接可以帮你登录；
2. 一般网站都会保持登录，也就是下次不用再重新登录，这是因为用户登录的时候可以存储一个Token在Cookies中，下次登录的时候只需要根据Token值来查找用户，重新登录一般Token重写；
3. 登录一次网站，其他页面不用重新登录；

Session是通过服务器端记录用户的状态： 因为HTTP是无状态的，服务器给特定用户创建特定的Session之后就可以表示这个用户并跟踪这个用户了；

Cookies保存在客户端，Session保存在服务器端； Session相对安全，因为Session在服务端，如果Cookies中要存储一些敏感信息，最好不要直接写入Cookies中，加密之后，在服务端解密会好一些；

缓存

1. 优点 缓解服务器压力； 降低客户端获取资源的延迟；缓存通常位于内存中，读取缓存的速度更快。并且缓存服务器在地理位置上也有可能比源服务器来的近。
2. 实现方法 让代理服务器进行缓存； 让客户端浏览器进行缓存；
3. Cache-Control http1.1通过Cache-Control首部字段控制缓存； (1)禁止进行缓存：no-store指令规定不能对请求或响应的任何一部分进行缓存。 (2)强制确认缓存：no-cache指令只要先向源服务器验证缓存资源的有效性，有效之后才能使用该缓存对客户端请求进行缓存； (3)私有缓存和公共缓存： private 指令规定了将资源作为私有缓存，只能被单独用户使用，一般存储在用户浏览器中。 public 指令规定了将资源作为公共缓存，可以被多个用户使用，一般存储在代理服务器中。 (4)缓存过期机制： max-age 指令出现在请求报文，并且缓存资源的缓存时间小于该指令指定的时间，那么就能接受该缓存。
4. 缓存验证： 需要先了解 ETag 首部字段的含义，它是资源的唯一标识。URL 不能唯一表示资源，例如 http://www.google.com/ 有中文和英文两个资源，只有 ETag 才能对这两个资源进行唯一标识。

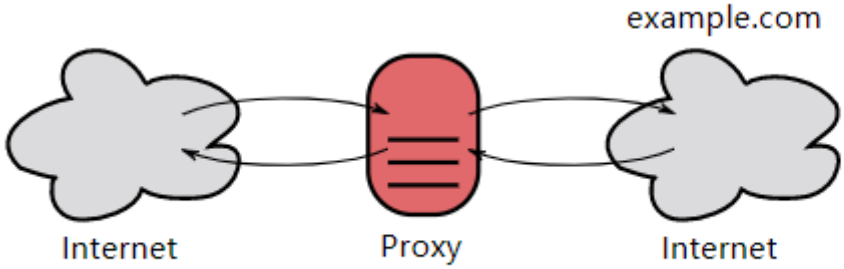
虚拟主机

HTTP/1.1 使用虚拟主机技术，使得一台服务器拥有多个域名，并且在逻辑上可以看成多个服务器。

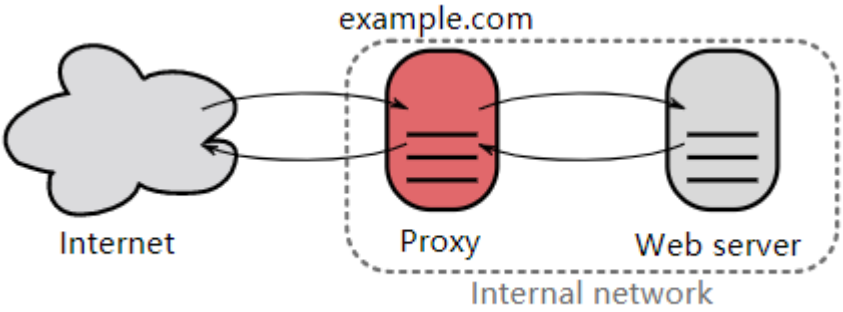
通信数据转发

1. 代理 代理服务器接受客户端的请求，并且转发给其他服务器； 使用代理的目的是： 缓存； 负载均衡； 网络访问控制； 访问日志记录；

代理服务器分为正向代理和反向代理两种： 用户察觉得到正向代理的存在；



反向代理存在于内部网络中，用户察觉不到。



2. 网关 与代理服务器不同，网关服务器会将HTTP转化为其他协议进行通信，从而请求其他非HTTP服务器的服务；

3. 隧道 使用SSL等加密手段，在客户端和服务器之间建立一条安全的通信线路；

HTTP1.0和HTTP1.1的区别

1. 长连接 : 在HTTP/1.0中，默认使用的是短连接，也就是说每次请求都要重新建立一次连接。如果每次请求都要这样的话，开销会比较大。HTTP 1.1起，默认使用长连接 ,默认开启Connection： keep-alive。HTTP/1.1的持续连接有非流水线方式和流水线方式 。**流水线方式**是客户在收到HTTP的响应报文之前就能接着发送新的请求报文。与之相对应的**非流水线**方式是客户在收到前一个响应后才能发送下一个请求。
2. 错误相应码：新增了24个错误相应码；
3. 缓存处理：引入更多的缓存控制策略；
4. 带宽优化及网络连接的使用，1.0存在浪费资源带宽的现象，客户端只需要某个对象的一部分，服务器却将整个对象送过来了，不支持端点续传；1.1则在请求头引入range头域，他允许请求资源的某个部分；

URI和URL的区别

URI是统一资源标志符，可以唯一标识这个资源； URL是统一资源定位符，可以提供该资源的路径，他是一种具体的URI，即URL可以标识一个资源也指明了如何locate这个资源；

HTTP和HTTPS的区别

1. **端口**，HTTP的URL由http:// 起始默认使用80端口，而HTTPS的URL由https:// 起始默认使用端口443.
2. **安全性和资源消耗**，HTTP协议运行在TCP之上，所有传输的内容是明文，客户端和服务器端都无法验证对方的身份，HTTPS运行在SSL/TLS协议之上的HTTP协议，SSL/TLS运行在TCP之上，所有运输的内容都经过加密，加密采用对称加密，但对称加密的秘钥用服务器整数进行了非对称加密，所以说HTTP安全性没有HTTPS高，但是HTTPS比HTTP消耗更多资源； 对称加密：密钥只有一个，加密解密为同一个密码，且加解密速度快，典型的对称加密算法有DES，AES； 非对称加密：秘钥成对出现，且根据公钥无法推出

私钥，根据私钥没法推出公钥，加解密使用不同的密钥，相对对称加密速度较慢，典型的就是RSA，DSA等；