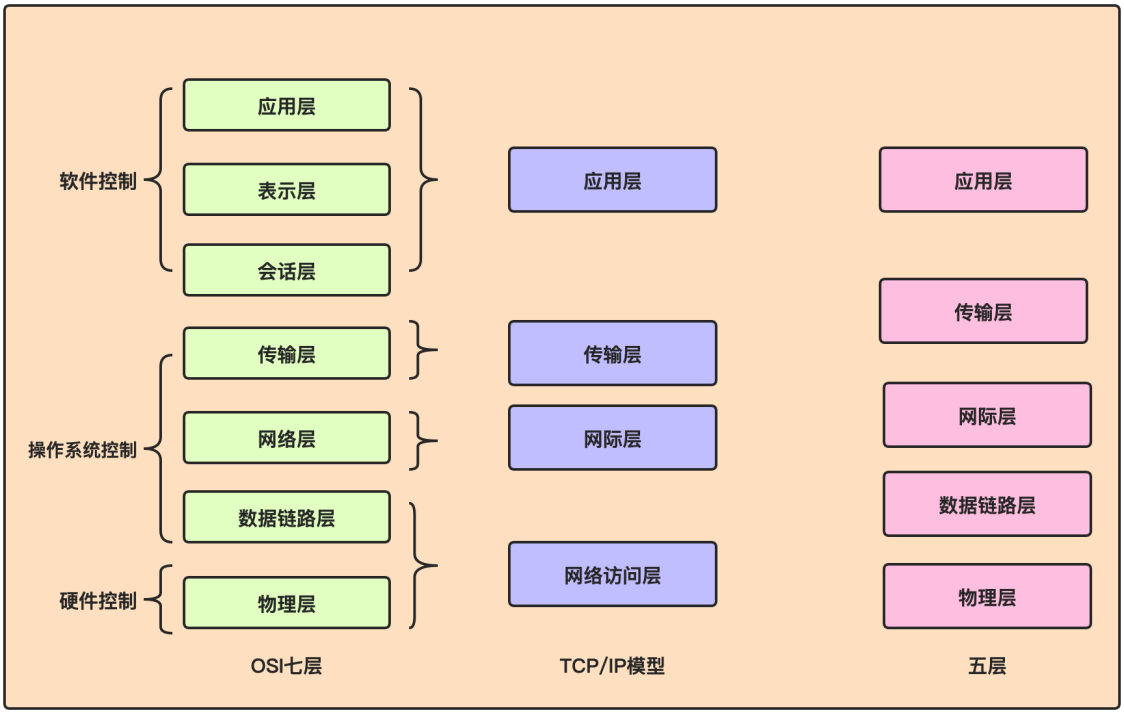
计算机网络相关面试题整理

网络模型



1. OSI七层模型

- 物理层：底层数据传输
- 数据链路层：定义数据的基本格式，如何传输和标识，如MAC地址
- 网络层：定义IP地址和路由功能，如不同设备的数据转发
- 传输层：端到端传输的基本功能，如TCP和UDP
- 会话层：控制应用程序之间的会话能力，如不同软件数据分发给不同软件
- 表示层：数据格式标识，基本压缩功能
- 应用层：各种应用软件，包括web应用

> 传输层数据称为段、网络层数据称为包、数据链路层数据称为帧、物理层称为比特流

总结

- 七层模型是一个标准而非实现
- 四层（TCP/IP）模型是一个实现的应用模型

2. ping命令的原理

- ping是基于网络层的命令，基于ICMP协议。ICMP协议规定：目的主机必须返回ICMP回送应答消息给源主机。如果源主机在一定时间内收到应答，则认为主机可达。
- ping不能完全记录所经过的路由，traceroute可以。首先发一个TTL=1的报文，随后依次增加

3. ARP的原理

> 解决地址问题的协议，以目标IP地址为线索，用来定位下一个应该接收数据分包的网络设备对应的MAC地址

ARP欺骗

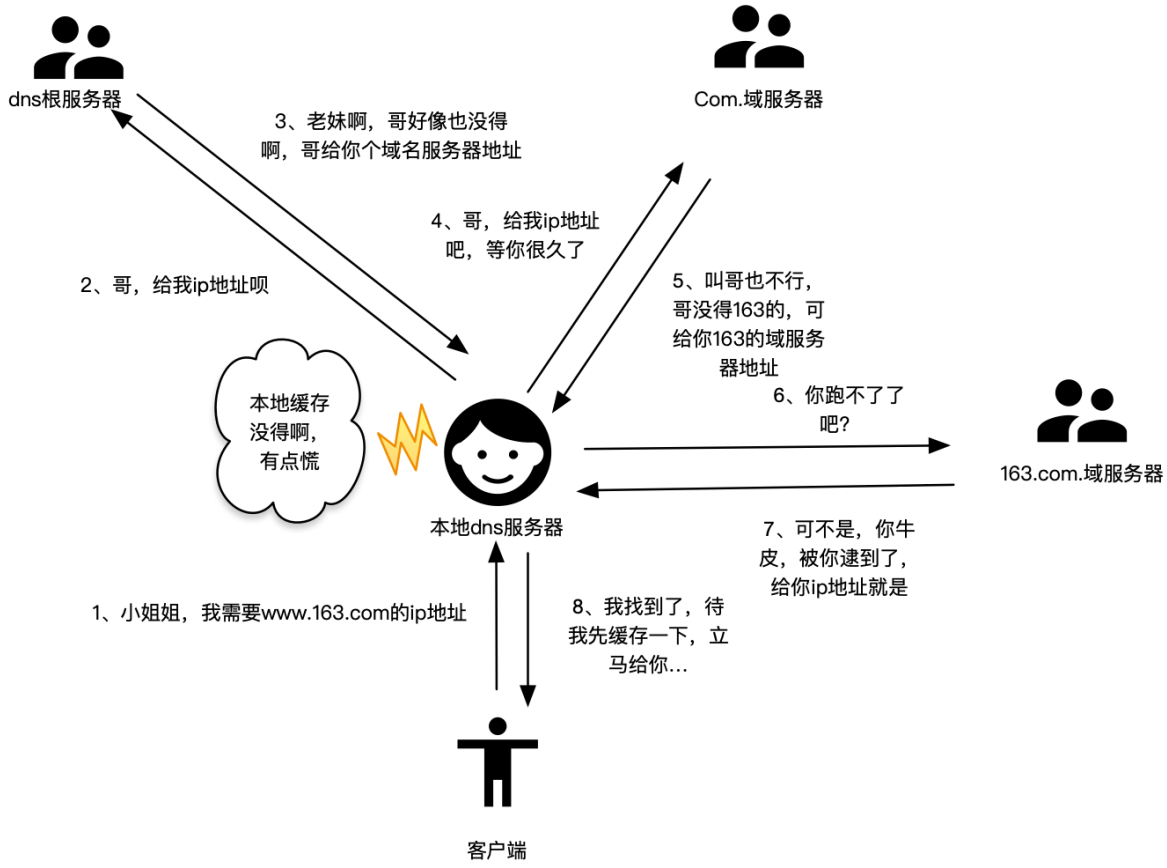
> ARP攻击就是通过伪造IP地址和MAC地址实现ARP欺骗，能够在网络中产生大量的ARP通信量使网络阻塞，攻击者只要持续不断的发出伪造的ARP响应包就能更改目标主机ARP缓存中的IP-MAC条目，造成网络中断或中间人攻击

4. DNS

定义

> Domain Name System(域名解析系统)，因特网上**域名和IP地址相互映射的一个分布式数据库。**通过主机名得到对应IP地址的过程叫做域名解析

解析过程



- 请求顺序：本地dns-根dns-.com域服务器-xxx.com域名服务器
- 先在浏览器找之前有没有缓存过域名对应的ip地址，有的话直接跳过dns解析

DNS查询方式

- 递归解析
- 迭代解析：只能找到相关服务器，不会帮你去查
- **DNS负载均衡**：DNS服务器为同一个主机名配置多个IP地址，在应答DNS查询时，DNS服务器对每个查询将以DNS文件中主机记录的IP地址按顺序返回不同的结果，将客户端的访问引导到不同的机器上去，使得不同的客户端访问不同的服务器
- 为什么DNS用UDP？：UDP快
- 为什么区域传送用TCP？：TCP协议可靠性好

DNS劫持

> 通过劫持DNS服务器，通过某些手段取得某域名的解析记录控制权，进而修改此域名的解析结果，导致对该域名的访问由原IP地址转入到修改后的指定IP，其结果就是对特定的网址不能访问或访问的是假网址

DNS污染

> DNS污染是指一些刻意制造或无意中制造出来的域名服务器分组，把域名指往不正确的IP地址。它是一种让一般用户由于得到虚假目标主机IP而不能与其通信的方法，是一种DNS缓存投毒攻击（DNS cache poisoning）。其工作方式是：由于通常的DNS查询没有任何认证机制，而且DNS查询通常基于的UDP是无连接不可靠的协议，因此DNS的查询非常容易被篡改，

TCP

5. TCP协议

定义

> Transmission Control Protocol 传输控制协议，是一种面向连接的、可靠的、基于字节流的传输层通信协议

TCP头部

图 3-3　TCP 头部结构

- 16位源端口和16位目的端口号

  > 应用程序的端口号和应用程序所在主机的IP地址统称为socket（套接字），一个socket唯一标识一个应用程序

- 32位序列号（Sequence Number）

  > 用于TCP通信某一传输方向上字节流的每个字节编号，确保数据通信的有序性，避免乱序问题。接收端根据编号确认分割的数据段在原始数据包的位置。Sequence Number_x = Acknowledge Number_y（x的序列号=y发给x的确认号）

- 32位确认序列号（Acknowledge Number）

  > 接收端锁期望收到的下一序列号。接收端的确认序列号为上次成功收到的序列号+1，只有当标志位中的ACK为1时，确认序列号才有效，主要用来解决不丢包的问题

- 6位标志位TCP Flag

  > URG,ACK,PSH,RST,SYN,FIN

  - ACK:ACK位0表示接收端还未应答，一旦接收到收到数据之后，就将ACK置为1

- SYN（同步序列号）：TCP握手发送的第一个数据包。当SYN=1，ACK=0连接被响应的时候，SYN=1，ACK=1.通常被用来进行端口扫描
- FIN：表示发送端已经达到数据末尾，没有数据需要传输。发送FIN位标志的TCP数据包后，连接将被断开

Window Size

> TCP header中有一个Window Size字段，指接收端的窗口，即接收窗口，用来告知发送端本身所能接收的数据量，从而达到一部分流控的目的
> 为了得到最优的连接速率，使用TCP窗口来控制流速率，滑动窗口是一种主要机制。这个窗口容许源端在给定链接传送数据分段而不用等待目标端返回ACK
> 滑动的依据就是发送数据已经收到ACK，确认对端收到，才能继续滑动窗口发送新的数据。能够看到窗口大小对于吞吐量有着重要影响
> 发送端收到窗口为0时的报文，启动持续计数器，超时后重新发送零窗口探测报文

拥塞控制

> 慢开始、拥塞避免、快重传、快恢复

慢开始+拥塞避免

> 发送端维护一个拥塞窗口cwnd的状态变量，发送窗口swnd，慢开始门限ssthresh
> 发送方使用拥塞窗口为发送窗口 swnd = cwnd,每次收到确认报文段，有cwnd += 确认报文段数量
> 拥塞避免后，每次 cwnd +=1
> 报文丢失认为有拥塞， ssthresh变为当前cwnd一半，cwnd置为1，

- 当cwnd < ssthresh时，使用慢开始
- 当cwnd > ssthresh时，使用拥塞避免
- 当cwnd = ssthresh时，两中都可以

快重传

- 接收端不要等待自己发送数据时才确认，而是立即发送确认
- 即使收到了失序的报文也要立即对已收到的确认
- 发送方一旦收到3个连续的重复确认，不等待超时重传计数器，立即重传

快恢复

- ssthresh和cwnd变为当前窗口的一半
- 或者cwnd等于新的ssthresh+3

三次握手

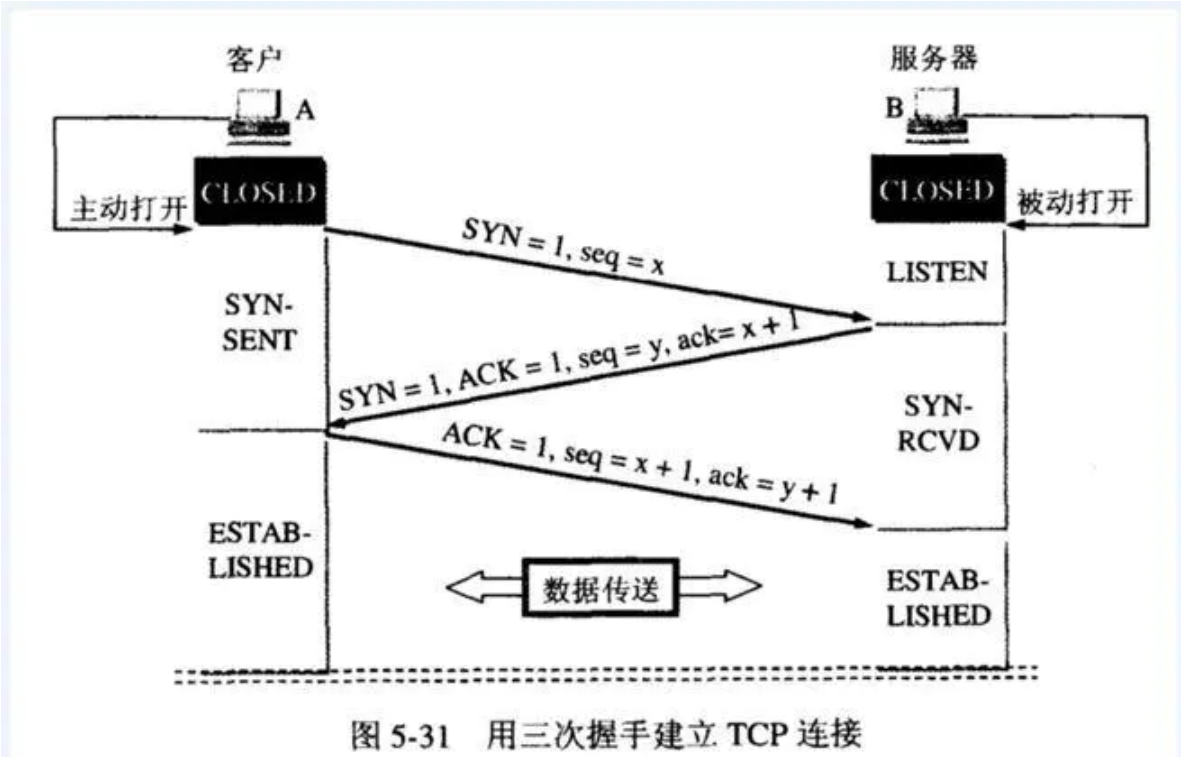
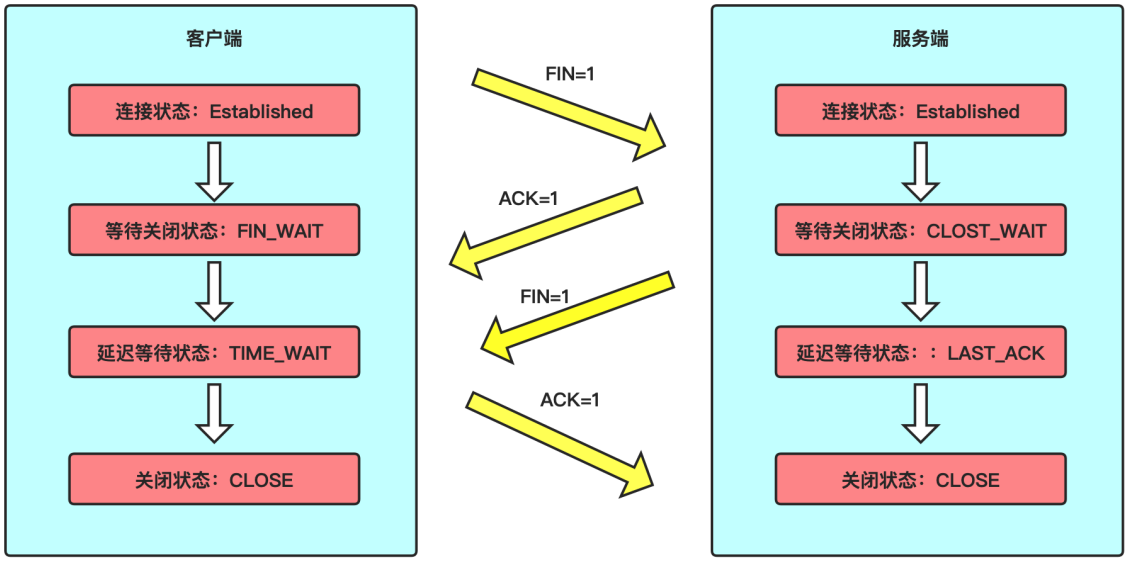


图 5-31　用三次握手建立 TCP 连接

TCP四次分手



TCP的三次握手和四次挥手

- **为什么是三次握手**：为了防止已失效的连接请求报文段突然又传送到了服务端,TCP 需要seq序列号来做可靠重传或接收，而避免连接复用时无法分辨出 seq 是延迟或者是旧链接的 seq，因此需要三次握手来约定确定双方的 ISN（初始 seq 序列号）
- **为什么是四次挥手**：任何一方都可以在数据传输结束后发起连接释放的通知，待对方确认后进入半关闭状态。当对方也无数据再次发送时，则发出连接释放通知，对方确认后就完全关闭TCP连接。
- **为什么有2MSL的等待延迟**：如果最后客户端发送的ACK=1丢失，服务端无法判断是否已经发送完数据，因此再次发起断开连接的请求，一个来回就是2MSL
- **TIME_WAIT过多的危害及解决办法**：占用内存；修改TIME_WAIT相关参数(tw_reuse,te_recycle)

TCP粘包

- TCP是流传输协议,是一种面向连接的、可靠的、基于字节流的传输层通信协议
- TCP没有包的概念，它只负责传输字节序列，UDP是面向数据报的协议，所以不存在拆包粘包问题

- 由应用层来维护消息和消息的边界，即需要一个应用层协议，比如HTTP

> 解决方式：

- 消息长度固定，提前确定包长度，读取的时候也安固定长度读取，适合定长消息包。
- 使用特殊的字符或字符串作为消息的边界，例如 HTTP 协议的 headers 以“\r\n”为字段的分隔符
- 自定义协议，将消息分为消息头和消息体，消息头中包含表示消息总长度

TCP如何保证可靠传输

- 确认和重传
- 数据校验：报文头部有校验和，校验报文是否损坏
- 合理分片和排序
- 拥塞控制
- 流量控制

TCP长连接和短连接

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完成一次读写之后，**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。**

短连接

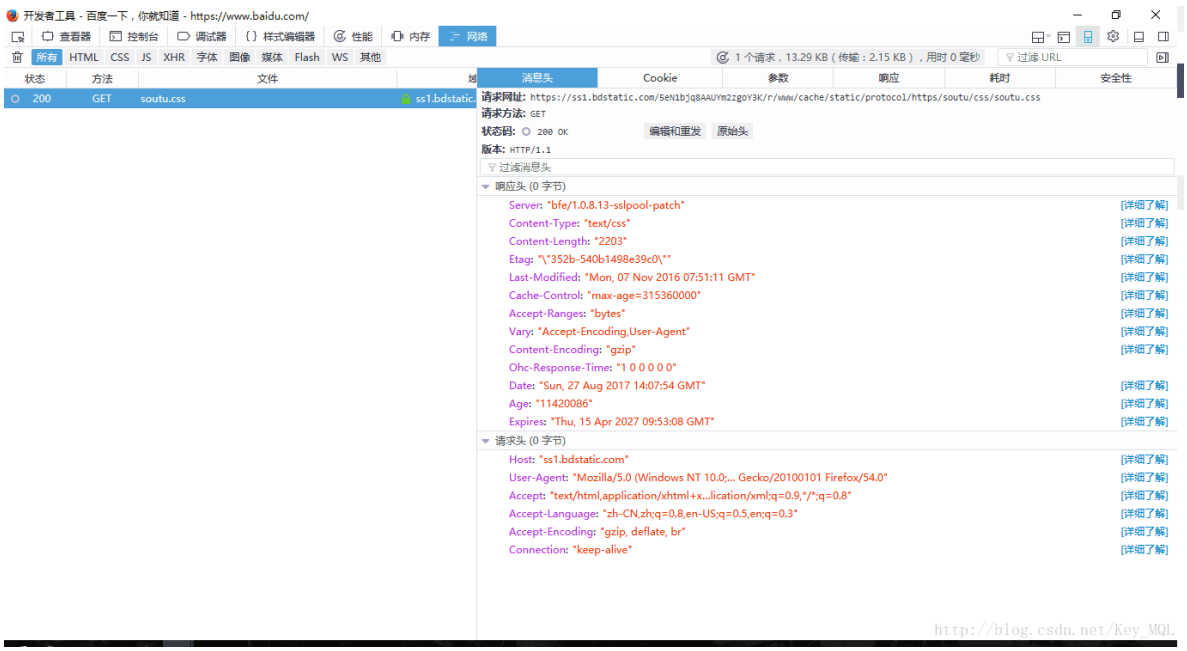
> cient向server发起连接请求，server接到请求，然后双方建立连接。client向server发送消息，server回应client，然后一次读写就完成了，这时候双方任何一个都可以发起close操作

6. HTTP

HTTP的状态码

- **1xx 信息**：100 Continue 表明目前为止正常，客户端可以持续发送或忽略这个相应
- **2xx 成功**
  - 200：OK
  - 204：请求已成功处理，但是不返回数据
  - 206：客户端进行了范围请求，响应报文包含由Content-Range指定范围的实体内容
- **3xx 重定向**
  - 301：永久重定向
  - 302：临时重定向
  - 303：和302类似，但明确由GET获取资源
  - 304：请求报文包含一些条件，不满足则返回304
  - 307：临时重定向，要求浏览器不会把POST改成GET
- **4xx 客户端错误**
  - 400：请求存在语法错误
  - 401：需要有认证信息或认证失败
  - 403：没有权限
  - 404：路由不存在或没找到
- **5xx 服务器错误**
  - 500：服务端正在执行请求时错误
  - 503：服务端无法处理

HTTP头部包含哪些key



GET和POST的区别

- GET使用URL或者Cookie传递参数，而POST将数据存放在BODY中
- GET方式提交的数据有长度限制，而POST的数据可以很大
- POST比GET方式安全，数据在地址栏上不可见
- **本质区别**：GET请求是幂等性的，POST请求不是。幂等性是指一次和多次请求某一个资源应该具有同样的副作用。简单来说意味着对同一URL的多个请求应该返回同样的结果。

> 因此不应该且**不能用get请求做数据的增删改这些有副作用的操作。**因为get请求是幂等的，**在网络不好的隧道中会尝试重试。**如果用get请求增数据，会有**重复操作**的风险，而这种重复操作可能会导致副作用（浏览器和操作系统并不知道你会用get请求去做增操作）。

http是否无状态？如何有状态？

> http本身是无状态协议，每次请求都是独立的，上一次的请求不会影响这一次。通过引入Cookie和Session记录用户信息

Cookies和Session的区别

Cookies

> 服务器发送到用户浏览器并保存到本地的一小块数据，用来告诉服务器多个请求是否来自同一浏览器。主要用途：

- 会话状态管理（如用户登录状态、购物车、游戏分数或其它需要记录的信息）
- 个性化设置（如用户自定义设置、主题等）
- 浏览器行为跟踪（如跟踪分析用户行为等）

Session

> 将用户信息通过Session存储在服务器中

Cookies和Session的选择与区别

- Cookies智能存储ASCII码字符串，Session可以任何类型
- Cookies在浏览器，Session存储在服务器
- 如果大型网站将所有信息都通过Session保存在服务器，开销非常大

JWT

> Json web token是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准
> 只需要服务端生成token，客户端保存后每次携带这个token，服务端认证解析即可

构成

> 第一部分我们称它为**头部（header)**,第二部分我们称其为**载荷（payload)**，第三部分是**签证（signature)**

特性

- 因为json的通用性，所以JWT是可以进行跨语言支持的，像JAVA,JavaScript,NodeJS,PHP等很多语言都可以使用。
- payload部分，JWT可以在自身存储一些其他业务逻辑所必要的非敏感信息。
- 便于传输，jwt的构成非常简单，字节占用很小，所以它是非常便于传输的。它不需要在服务端保存会话信息, 所以它易于应用的扩展。

在浏览器中输入url地址后显示主页的过程?

- 根据域名，进行DNS域名解析；
- 拿到解析的IP地址，建立TCP连接；
- 向IP地址，发送HTTP请求；
- 服务器处理请求；
- 返回响应结果；
- 关闭TCP连接；
- 浏览器解析HTML；
- 浏览器布局渲染；

HTTP1.x的缺点

- 一次只允许在一个TCP连接上发起请求
- 单向，只能由客户端发起
- 请求报文与响应报文首部信息冗余量大
- 数据未压缩

HTTP2.0的改变

- 多路复用允许同时通过单一的HTTP连接发起多重请求响应的消息
- 首部压缩
- 服务端推送
- 二进制分帧

http和RPC的异同

相同点

> 底层都是基于Socket

不同点

- RPC的服务提供方和消费方必须使用同一的RPC框架，调用快处理快
- http无需关注服务提供方的编程语言，只需要按照restful风格的原则请求即可，通用性强

RPC

> Remote Procedure Call远程服务调用

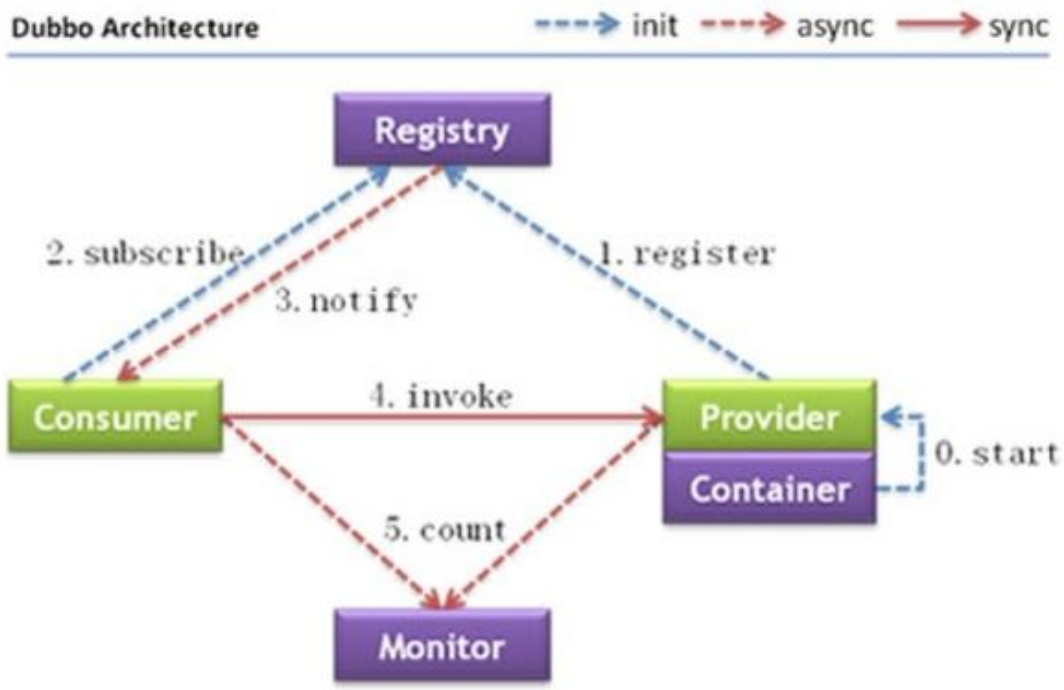
- 客户端client发起服务调用请求。

- client stub 可以理解成一个代理，会将调用方法、参数按照一定格式进行封装，通过服务提供的地址，发起网络请求。
- 消息通过网络传输到服务端。
- server stub接受来自socket的消息
- server stub将消息进行解包、告诉服务端调用的哪个服务，参数是什么
- 结果返回给server stub。
- sever stub把结果进行打包交给socket
- socket通过网络传输消息
- client slub 从socket拿到消息。
- client stub解包消息将结果返回给client。

RPC的序列化方式

- JDK原生序列化
- JSON: 进行序列化的额外空间开销巨大，对于大数据量服务意味着需要巨大的内存和磁盘开销；没有类型
- Hessian：是动态类型、二进制、紧凑的，并且可跨语言移植的一种序列化框架；对Java里面一些常见对象的类型不支持，比如Linked
- Protobuf：Google 公司内部的混合语言数据标准，是一种轻便、高效的结构化数据存储格式；序列化后体积相比 JSON、Hessian 小很多；--IDL 能清晰地描述语义，所以足以帮助并保证应用程序之间的类型不会丢失，无需类似XML 解析器；

拓展（阿里的RPC框架Dubbo）



> 具体的交互流程是 Consumer 一端通过注册中心获取到 Provider 节点后，通过 Dubbo 的客户端 SDK 与 Provider 建立连接，并发起调用。Provider 一端通过 Dubbo 的服务端 SDK 接收到 Consumer 的请求，处理后再把结果返回给 Consumer。

- 核心包括：远程通信、集群容错、自动发现
- 能做什么：透明化的远程方法调用、软负载均衡及容错机制
- 服务自动注册与发现

> Dubbo采用全spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可，Dubbo基于Spring的Schema扩展进行加载。

7. Https

Https的定义

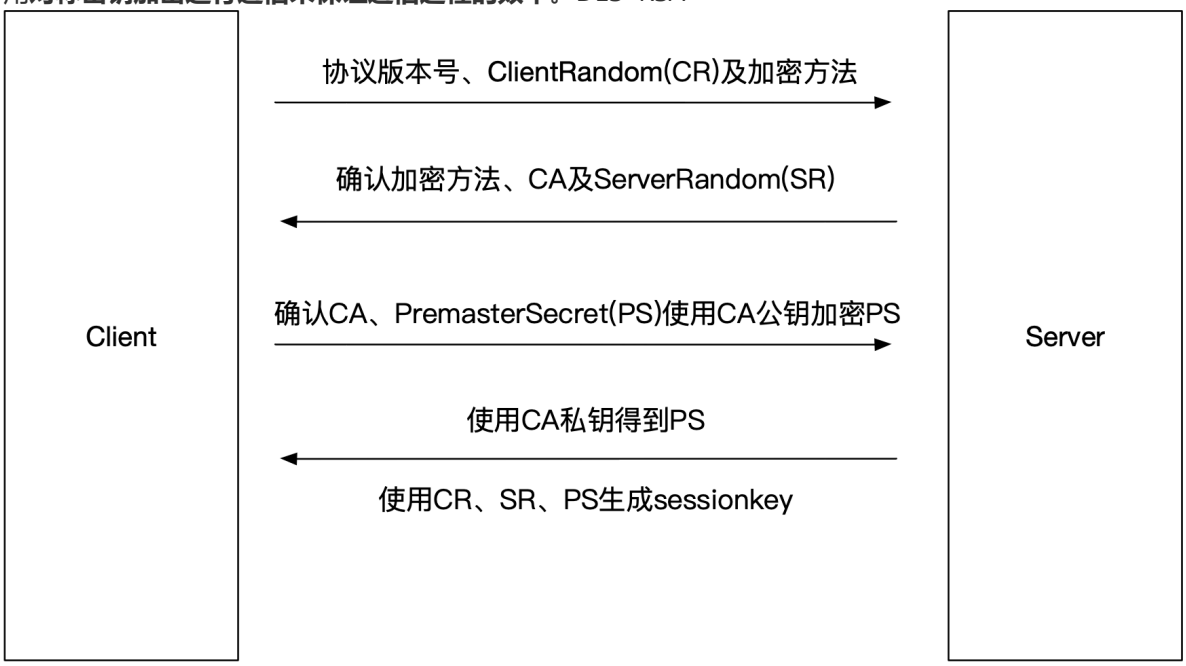
- HTTPS并不是新协议，而是让**HTTP先和SSL（Secure Socket Layer）通信，再由SSL和TCP通信，也就说说HTTPS使用了隧道进行通信**
- 通过使用SSL，HTTPS具备了加密（防窃听）、认证（防伪装）和完整性保护（防篡改）

`HTTP`的缺点

- 使用明文进行通信，内容可能会被窃听
- 不验证通信方的身份，通信方的身份有可能遭遇伪装
- 无法验证报文的完整性，有可能被篡改

https的通信过程和加密过程

HTTPS 采用混合的加密机制，使用**非对称密钥加密用于传输对称密钥来保证传输过程的安全性**，之后使用**对称密钥加密进行通信来保证通信过程的效率。**DES+RSA



确保传输安全过程（其实就是rsa原理）：

- Client给出**协议版本号**、一个客户端生成的**随机数**（Client random），以及客户端支持的**加密方法。**
- Server确认双方使用的**加密方法**，并给出**数字证书**、以及一个服务器生成的**随机数**（Server random）。
- Client确认**数字证书有效**，然后生成一个新的**随机数**（Premaster secret），并使用**数字证书中的公钥，加密这个随机数**，发给Server。
- Server使用自己的**私钥，获取Client发来的随机数**（Premaster secret）。
- Client和Server根据约定的加密方法，使用前面的**三个随机数，生成"对话密钥"**（session key），用来加密接下来的整个对话过程。

怎么判断证书的有效性

数字证书签名流程

- 首先 CA 会把持有者的公钥、用途、颁发者、有效时间等信息打成一个包，然后对这些信息进行 Hash 计算，得到一个 Hash 值；
- 然后 CA 会使用自己的私钥将该 Hash 值加密，生成 Certificate Signature，也就是 CA 对证书做了签名；
- 最后将 Certificate Signature 添加在文件证书上，形成数字证书

验证流程

- 首先客户端会使用同样的 Hash 算法获取该证书的 Hash 值 H1；
- 浏览器和操作系统中集成了 CA 的公钥信息，浏览器收到证书后可以使用 CA 的公钥解密 Certificate Signature 内容，得到一个 Hash 值 H2；
- 最后比较 H1 和 H2，如果值相同，则为可信赖的证书，否则则认为证书不可信。

8. FTP

> 基于TCP协议，工作在应用层

传输模式

- ASCII模式和二进制模式

主动和被动模式

- 主动模式FTP的客户端发送 PORT 命令到FTP服务器。
- 被动模式中FTP的客户端发送 PASV命令到 FTP 服务器。