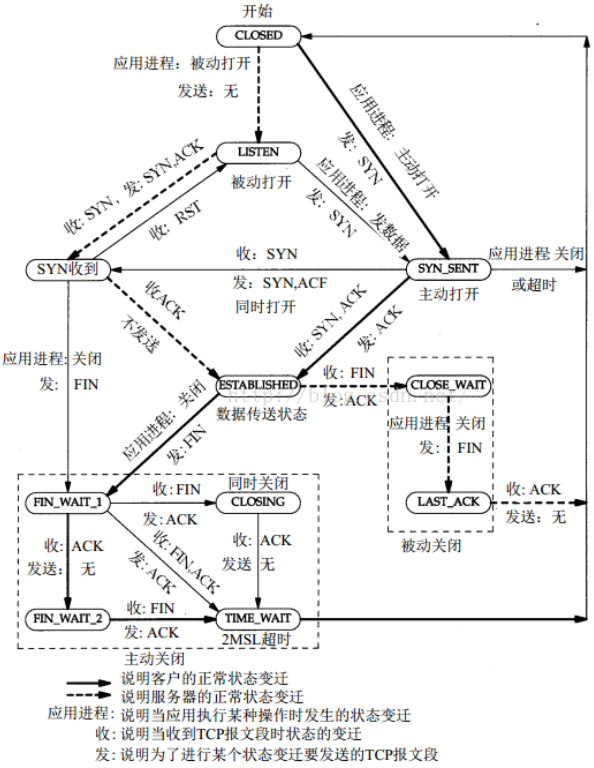
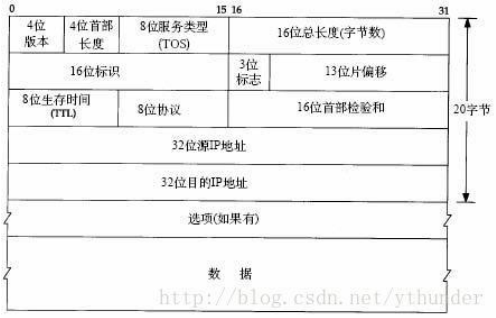
TCP 状态转换图



IP 头部



TCP 头部

UDP 头部

网络 7 层、4 层 、5 层模型(体系结构)

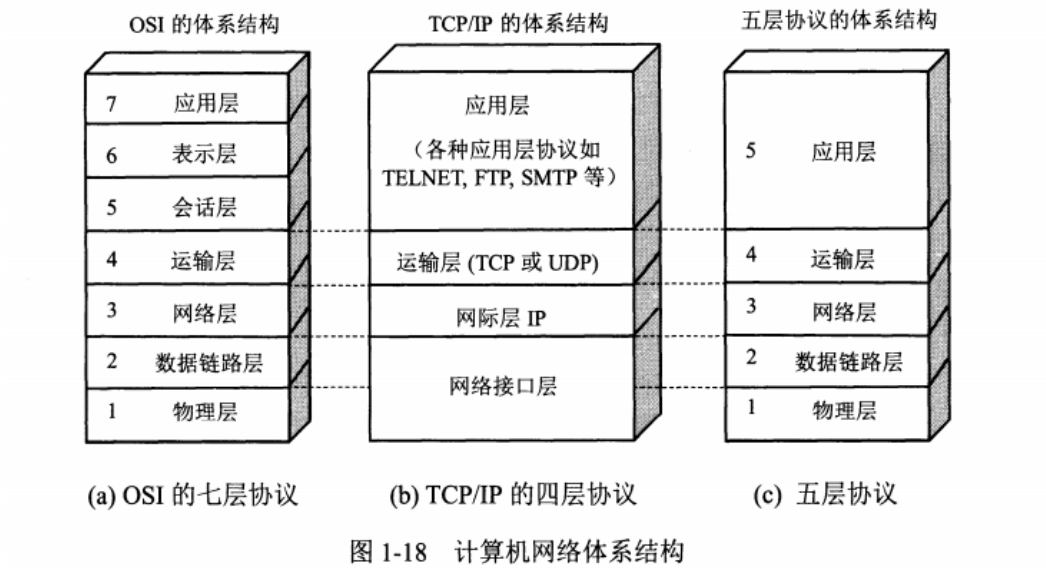
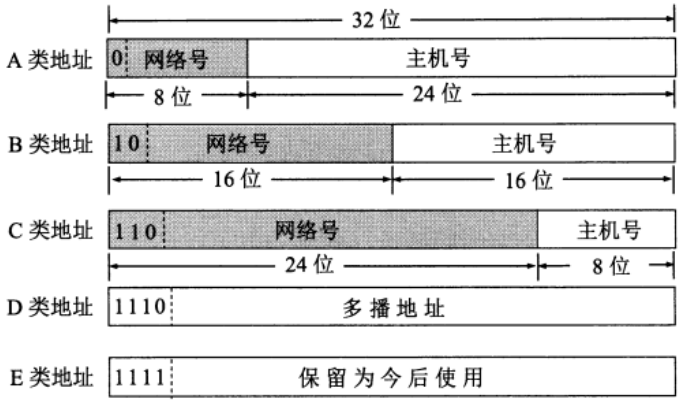


图 1-18　计算机网络体系结构

IP 地址分类



3 次握手(三次握手)

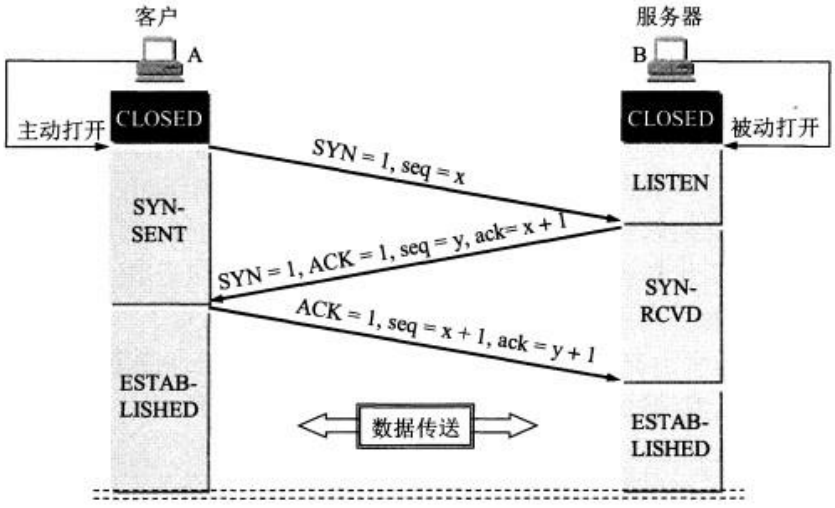


图 5-28 用三报文握手建立 TCP 连接

4 次挥手(四次挥手)

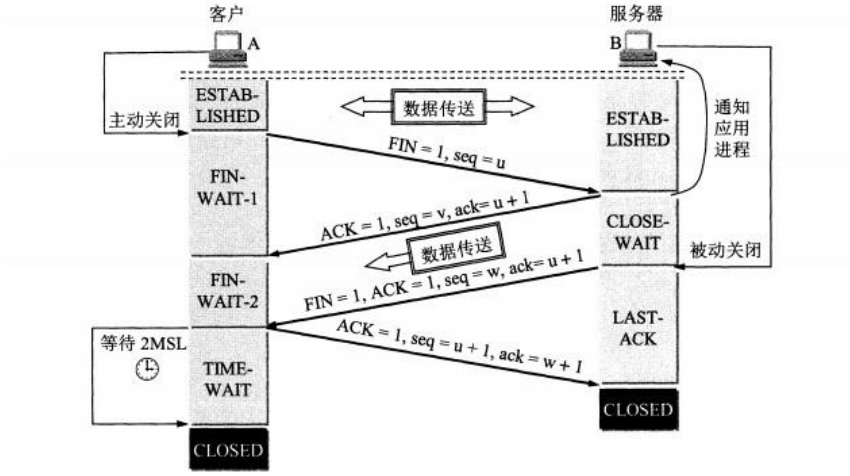


图 5-29　TCP 连接释放的过程

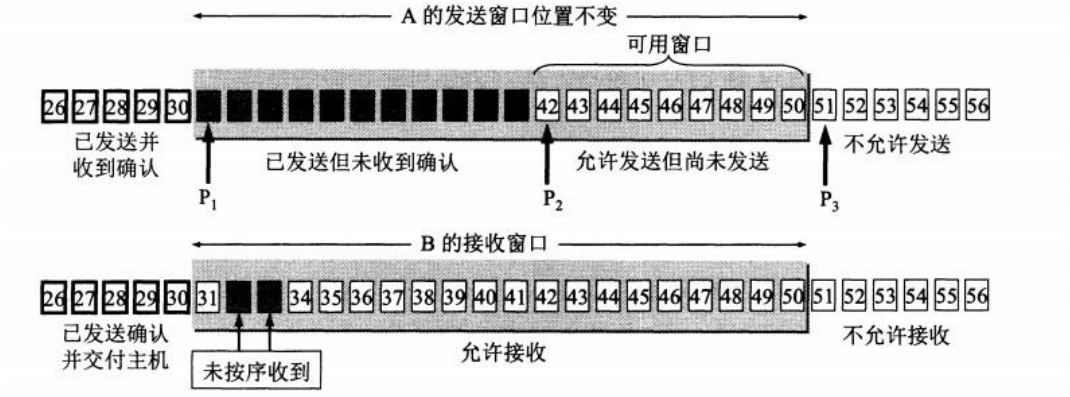
交换机与路由器的区别

- 交换机工作于数据链路层，能识别 MAC 地址，根据 MAC 地址转发链路层数据帧。具有自学机制来维护 IP 地址与 MAC 地址的映射。
- 路由器位于网络层，能识别 IP 地址并根据 IP 地址转发分组。维护着路由表，根据路由表选择最佳路线。

TIME_WAIT(2MSL)

1. 确保最后一个确认报文段能够到达。如果 B 没收到 A 发送来的确认报文段，那么就会重新发送连接释放请求报文段，A 等待一段时间就是为了处理这种情况的发生。
2. 可能存在“已失效的连接请求报文段”，为了防止这种报文段出现在本次连接之外，需要等待一段时间。防止串话。

TCP 滑动窗口



TCP 可靠传输

1、确认和重传：接收方收到报文就会确认，发送方发送一段时间后没有收到确认就重传。

2、数据校验

3、数据合理分片和排序：

UDP：IP 数据报大于 1500 字节,大于 MTU.这个时候发送方 IP 层就需要分片(fragmentation).把数据报分成若干片,使每一片都小于 MTU.而接收方 IP 层则需要进行数据报的重组.这样就会多做许多事情,而更严重的是,由于 UDP 的特性,当某一片数据传送中丢失时,接收方便无法重组数据报.将导致丢弃整个 UDP 数据报.

tcp 会按 MTU 合理分片，接收方会缓存未按序到达的数据，重新排序后再交给应用层。

4、流量控制：当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。

5、拥塞控制：当网络拥塞时，减少数据的发送。

TCP 流量控制
流量控制是为了控制发送方发送速率，保证接收方来得及接收。
接收方发送的确认报文中的窗口字段可以用来控制发送方窗口大小，从而影响发送方的发送速率。例如将窗口字段设置为 0，则发送方不能发送数据。
TCP 拥塞控制

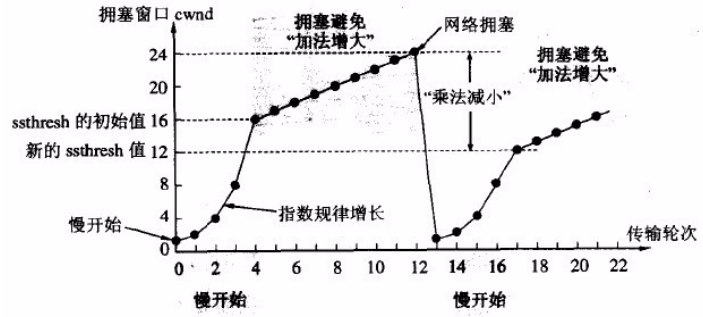


图 5-25 慢开始和拥塞避免算法的实现举例 net/sicofield

ssthresh：处理拥塞时参照的一个参数。例子中初始值为 16，后来变为 12。
当 cwnd> ssthresh，cwnd 以慢开始的方法指数增长；
当 cwnd< ssthresh，cwnd 以拥塞避免的方法线性增长。

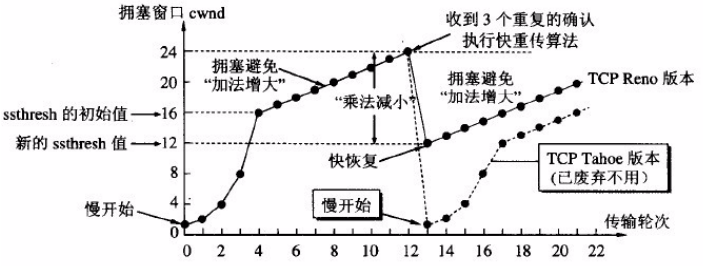


图 5-27 从连续收到三个重复的确认转入拥塞避免 net/sicofield

快重传：收到 3 个同样的确认就立刻重传，不等到超时；
快恢复：cwnd 不是从 1 重新开始。

网络编程一般步骤
- TCP：
  - 服务端：socket -> bind -> listen -> accept -> recv/send -> close。
  - 客户端：socket -> connect -> send/recv -> close。
- UDP：
  - 服务端：socket -> bind -> recvfrom/sendto -> close。
  - 客户端：socket -> sendto/recvfrom -> close。

TCP 和 UDP 区别
- TCP 面向连接（三次握手），通信前需要先建立连接；UDP 面向无连接，通信前不需要连接。
- TCP 通过序号、重传、流量控制、拥塞控制实现可靠传输；UDP 不保障可靠传输，尽最大努力交付。
- TCP 面向字节流传输，因此可以被分割并在接收端重组；UDP 面向数据报传输。

TCP 为什么不是两次握手而是三次
如果仅两次连接可能出现一种情况：客户端发送完连接报文（第一次握手）后由于网络不好，延时很久后报文到达服务端，服务端接收到报文后向客户端发起连接（第二次握手）。此时客户端会认定此报文为失效报文，但在两次握手情况下服务端会认为已经建立起了连接，服务端

会一直等待客户端发送数据，但因为客户端会认为服务端第二次握手的回复是对失效请求的回复，不会去处理。这就造成了服务端一直等待客户端数据的情况，浪费资源。

TCP 为什么挥手是四次而不是三次
TCP 是全双工的，它允许两个方向的数据传输被独立关闭。当主动发起关闭的一方关闭连接之后，TCP 进入半关闭状态，此时主动方可以只关闭输出流。
之所以不是三次而是四次主要是因为被动关闭方将"对主动关闭报文的确认"和"关闭连接"两个操作分两次进行。
对主动关闭报文的确认是为了快速告知主动关闭方，此关闭连接报文已经收到。此时被动方不立即关闭连接是为了将缓冲中剩下的数据从输出流发回主动关闭方（主动方接收到数据后同样要进行确认），因此要把"确认关闭"和"关闭连接"分两次进行。

阻塞和非阻塞 I/O 区别
- 如果内核缓冲没有数据可读时，read()系统调用会一直等待有数据到来后才从阻塞态中返回，这就是阻塞 I/O。
- 非阻塞 I/O 在遇到上述情况时会立即返回给用户态进程一个返回值，并设置 errno 为 EAGAIN。
- 对于往缓冲区写的操作同理。

同步和异步区别
- 同步 I/O 指处理 I/O 操作的进程和处理 I/O 操作的进程是同一个。
- 异步 I/O 中 I/O 操作由操作系统完成，并不由产生 I/O 的用户进程执行。

Reactor 和 Proactor 区别
- Reactor 模式是同步 I/O，处理 I/O 操作的依旧是产生 I/O 的程序；Proactor 是异步 I/O，产生 I/O 调用的用户进程不会等待 I/O 发生，具体 I/O 操作由操作系统完成。
- 异步 I/O 需要操作系统支持，Linux 异步 I/O 为 AIO，Windows 为 IOCP。

域名系统 DNS (使用 UDP)
1. 层次结构
域名服务器可以分为以下四类：
(1) 根域名服务器：解析顶级域名；
(2) 顶级域名服务器：解析二级域名；
(3) 权限域名服务器：解析区内的域名；
区和域的概念不同，可以在一个域中划分多个区。图 b 在域 abc.com 中划分了两个区：abc.com 和 y.abc.com
因此就需要两个权限域名服务器：
(4) 本地域名服务器：也称为默认域名服务器。可以在其中配置高速缓存。
2. 解析过程
主机向本地域名服务器解析的过程采用递归，而本地域名服务器向其它域名服务器解析可以使用递归和迭代两种方式。
迭代的方式下，本地域名服务器向一个域名服务器解析请求解析之后，结果返回到本地域名服务器，然后本地域名服务器继续向其它域名服务器请求解析；而递归地方式下，结果不是直接返回的，而是继续向前请求解析，最后的结果才会返回。

Web 页面请求过程(URL 请求过程)
浏览器中输入 URL，首先浏览器要将 URL 解析为 IP 地址，解析域名就要用到 DNS 协议，首先主机会查询 DNS 的缓存，如果没有就给本地 DNS 发送查询请求。DNS 查询分为两种方式，一种是递归查询，一种是迭代查询。如果是迭代查询，本地的 DNS 服务器，向根域名服务器发送查询请求，根域名服务器告知该域名的一级域名服务器，然后本地服务器给该一级域名服务器发送查询请求，然后依次类推直到查询到该域名的 IP 地址。DNS 服务器是基于 UDP 的，因此会用到 UDP 协议。
得到 IP 地址后，浏览器就要与服务器建立一个 http 连接。因此要用到 http 协议，http 协议报文格式上面已经提到。http 生成一个 get 请求报文，将该报文传给 TCP 层处理。如果采用 https 还会先对 http 数据进行加密。TCP 层如果有需要先将 HTTP 数据包分片，分片依据路径 MTU 和 MSS。TCP 的数据包然后会发送给 IP 层，用到 IP 协议。IP 层通过路由选路，一跳一跳发送到

目的地址。当然在一个网段内的寻址是通过以太网协议实现(也可以是其他物理层协议，比如 PPP，SLIP)，以太网协议需要直到目的 IP 地址的物理地址，有需要 ARP 协议。

ARP 协议

应用接受用户提交的数据，触发 TCP 建立连接，TCP 的第一个 SYN 报文通过 connect 函数到达 IP 层，IP 层通过查询路由表：

如果目的 IP 和自己在同一个网段：

当 IP 层的 ARP 高速缓存表中存在目的 IP 对应的 MAC 地址时，则调用网络接口 send 函数（参数为 IP Packet 和目的 MAC））将数据提交给网络接口，网络接口完成 Ethernet Header + IP + CRC 的封装，并发送出去；

当 IP 层的 ARP 高速缓存表中不存在目的 IP 对应的 MAC 地址时，则 IP 层将 TCP 的 SYN 缓存下来，发送 ARP 广播请求目的 IP 的 MAC，收到 ARP 应答之后，将应答之中的<IP 地址，对应的 MAC>对缓存在本地 ARP 高速缓存表中，然后完成 TCP SYN 的 IP 封装，调用网络接口 send 函数（参数为 IP Packet 和目的 MAC））将数据提交给网络接口，网络接口完成 Ethernet Header + IP + CRC 的封装，并发送出去；。

如果目的 IP 地址和自己不在同一个网段，就需要将包发送给默认网关(网关是默认的数据出口)，这需要知道默认网关的 MAC 地址：

当 IP 层的 ARP 高速缓存表中存在默认网关对应的 MAC 地址时，则调用网络接口 send 函数（参数为 IP Packet 和默认网关的 MAC）将数据提交给网络接口，网络接口完成 Ethernet Header + IP + CRC

当 IP 层的 ARP 高速缓存表中不存在默认网关对应的 MAC 地址时，则 IP 层将 TCP 的 SYN 缓存下来，发送 ARP 广播请求默认网关的 MAC，收到 ARP 应答之后，将应答之中的<默认网关地址，对应的 MAC>对缓存在本地 ARP 高速缓存表中，然后完成 TCP SYN 的 IP 封装，调用网络接口 send 函数（参数为 IP Packet 和默认网关的 MAC）将数据提交给网络接口，网络接口完成 Ethernet Header + IP + CRC 的封装，并发送出去。

ARP 的位置

OSI 模型有七层，TCP 在第 4 层传输层，IP 在第 3 层网络层，而 ARP 在第 2 层数据链路层。高层对低层是有强依赖的，所以 TCP 的建立前要进行 ARP 的请求和应答。

ARP 高速缓存表在 IP 层使用。如果每次建立 TCP 连接都发送 ARP 请求，会降低效率，因此在主机、交换机、路由器上都会有 ARP 缓存表。建立 TCP 连接时先查询 ARP 缓存表，如果有效，直接读取 ARP 表项的内容进行第二层数据包的发送；只有表失效时才进行 ARP 请求和应答进行 MAC 地址的获取，以建立 TCP 连接。

ARP 的作用

要了解 ARP 的作用，首先要分清两个“地址”：

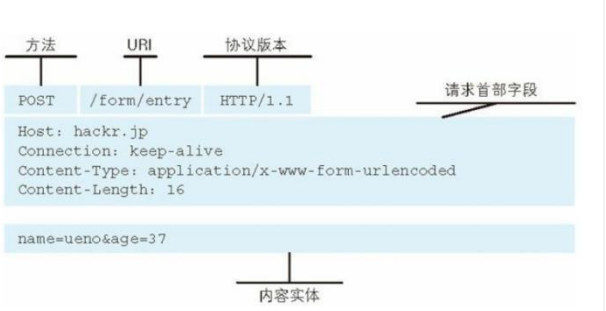
（1）TCP/IP 的 32bit IP 地址。仅知道主机的 IP 地址不能让内核发送数据帧给主机。

（2）网络接口的硬件地址，它是一个 48bit 的值，用来标识不同的以太网或令牌环网络接口。在硬件层次上，进行数据交换必须有正确的接口地址，内核必须知道目的端的硬件地址才能发送数据。

简言之，就是在以太网中，一台主机要把数据帧发送到同一局域网上的另一台主机时，设备驱动程序必须知道以太网地址才能发送数据。而我们只知道 IP 地址，这时就需要采用 ARP 协议将 IP 地址映射为以太网地址。

要注意一点，一般认为 ARP 协议只使适用于局域网。

HTTP 请求报文



HTTP 响应报文

HTTP 方法
GET：获取资源
POST：传输实体主体
POST 主要目的不是获取资源，而是传输实体主体数据。
GET 和 POST 的请求都能使用额外的参数，但是 GET 的参数是以查询字符串出现在 URL 中，而 POST 的参数存储在实体主体部分。

GET 的传参方式相比于 POST 安全性较差，因为 GET 传的参数在 URL 是可见的，可能会泄露私密信息。并且 GET 只支持 ASCII 字符，如果参数为中文则可能会出现乱码，而 POST 支持标准字符集。
HEAD：获取报文首部
和 GET 方法一样，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。
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，因此存在安全性问题，一般 WEB 网站不使用该方法。
DELETE：删除文件
与 PUT 功能相反，并且同样不带验证机制。
OPTIONS：查询支持的方法
查询指定的 URL 能够支持的方法。
会返回 Allow: GET, POST, HEAD, OPTIONS 这样的内容。
TRACE：追踪路径
服务器会将通信路径返回给客户端。
发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器就会减 1，当数值为 0 时就停止传输。

HTTP 状态码
服务器返回的响应报文中第一行为状态行，包含了状态码以及原因短语，来告知客户端请求的结果。

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）	需要进行附加操作以完
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

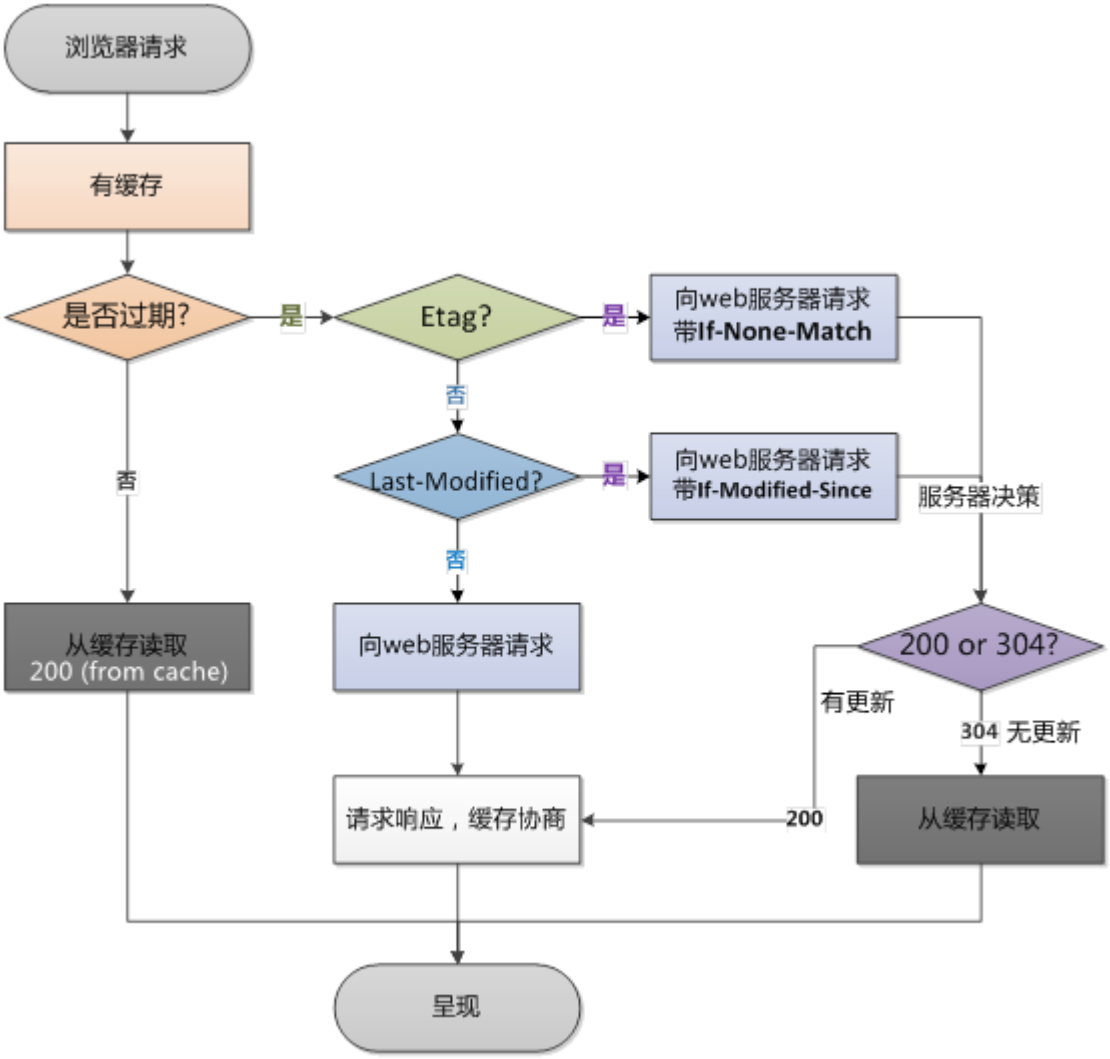
2XX 成功
200 OK
204 No Content：请求已经成功处理，但是返回的响应报文不包含实体的主体部分。一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。
206 Partial Content
3XX 重定向
301 Moved Permanently：永久性重定向
302 Found：临时性重定向
303 See Other
注：虽然 HTTP 协议规定 301、302 状态下重定向时不允许把 POST 方法改成 GET 方法，但是大多数浏览器都会把 301、302 和 303 状态下的重定向把 POST 方法改成 GET 方法。
304 Not Modified：如果请求报文首部包含一些条件，例如：If-Match，If-ModifiedSince，If-None-Match，If-Range，If-Unmodified-Since，但是不满足条件，则服务器会返回 304 状态码。
307 Temporary Redirect：临时重定向，与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 POST 方法改成 GET 方法。
4XX 客户端错误
400 Bad Request：请求报文中存在语法错误
401 Unauthorized：该状态码表示发送的请求需要有通过 HTTP 认证（BASIC 认证、DIGEST 认证）的认证信息。如果之前已进行过一次请求，则表示用户认证失败。

403 Forbidden：请求被拒绝，服务器端没有必要给出拒绝的详细理由。
404 Not Found
5XX 服务器错误
500 Internal Server Error：服务器正在执行请求时发生错误
503 Service Unavilable：该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

HTTP 常用字段
Cache-Control、Connection、Accept-Charset、Accept-Encoding、Accept-Language、Host、If-Match、If-Modified-Since、Allow

Session 和 Cookie 区别
Session 是服务器用来跟踪用户的一种手段，每个 Session 都有一个唯一标识：Session ID。当服务器创建了一个 Session 时，给客户端发送的响应报文就包含了 Set-Cookie 字段，其中有一个名为 sid 的键值对，这个键值对就是 Session ID。客户端收到后就把 Cookie 保存在浏览器中，并且之后发送的请求报文都包含 Session ID。HTTP 就是 Session 和 Cookie 这两种方式一起合作来实现跟踪用户状态的，而 Session 用于服务器端，Cookie 用于客户端。

HTTP 缓存



1、Exprires: Expires 是 Web 服务器响应消息头字段，在响应 http 请求时告诉浏览器在过期时间前浏览器可以直接从浏览器缓存取数据，而无需再次请求。
2、Cache-Control: Cache-Control 与 Expires 的作用一致，都是指明当前资源的有效期，控制浏览器是否直接从浏览器缓存取数据还是重新发请求到服务器取数据。 只不过 Cache-Control 的选择更多，设置更细致，如果同时设置的话，其优先级高于 Expires。
3、Last-Modified/If-Modified-Since：Last-Modified/If-Modified-Since 要配合 Cache-Control 使用。 Last-Modified：标示这个响应资源的最后修改时间。 If-Modified-Since：当资源过期时（使用 Cache-Control 标识的 max-age），发现资源具有 Last-Modified 声明，则再次向 web 服务器请求时带上头 If-Modified-Since，表示请求时间。
4、Etag/If-None-Match：Etag/If-None-Match 也要配合 Cache-Control 使用。 Etag：web 服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。 Apache 中，ETag 的值，默认是对文件的索引节（INode），大小（Size）和最后修改时间（MTime）进行 Hash 后得到的。 If-None-Match：当资源过期时（使用 Cache-Control 标识的 max-age），发现资源具有 Etage 声明，则再次向 web 服务器请求时带上头 If-None-Match （Etag 的值）。

HTTP 分块传输
分块传输（Chunked Transfer Coding）可以把数据分割成多块，让浏览器逐步显示页面。

HTTP 范围请求

如果网络出现中断，服务器只发送了一部分数据，范围请求使得客户端能够只请求未发送的那部分数据，从而避免服务器端重新发送所有数据。
在请求报文首部中添加 Range 字段，然后指定请求的范围，例如 Range : bytes = 5001-10000。
请求成功的话服务器发送 206 Partial Content 状态。

HTTP 内容协商
通过内容协商返回最合适的内容，例如根据浏览器的默认语言选择返回中文界面还是英文界面。
涉及以下首部字段：Accept、Accept-Charset、Accept-Encoding、Accept-Language、Content-Language。

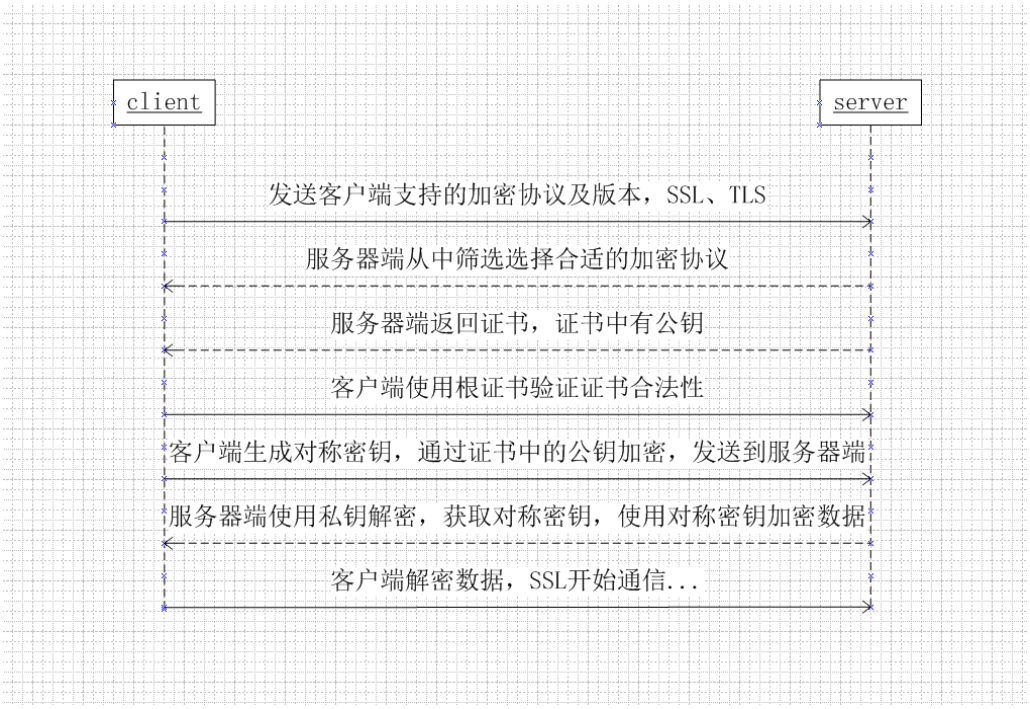
HTTP 虚拟主机
使用虚拟主机技术，使得一台服务器拥有多个域名，并且在逻辑上可以看成多个服务器。

HTTP 通信数据转发
代理
代理服务器接受客户端的请求，并且转发给其它服务器。代理服务器一般是透明的，不会改变 URL。
使用代理的主要目的是：缓存、网络访问控制以及记录访问日志。

网关
与代理服务器不同的是，网关服务器会将 HTTP 转化为其它协议进行通信，从而其它非 HTTP 服务器的服务。

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

HTTPs



HTTP 有以下安全性问题：

1. 通信使用明文，内容可能会被窃听；
2. 不验证通信方的身份，因此有可能遭遇伪装；
3. 无法证明报文的完整性，所以有可能已遭篡改。

HTTPs 并不是新协议，而是 HTTP 先和 SSL（Secure Socket Layer）通信，再由 SSL 和 TCP 通信。通过使用 SSL，HTTPs 提供了加密、认证和完整性保护。

认证

通过使用 证书 来对通信方进行认证。证书中有公开密钥数据，如果可以验证公开密钥的确属于通信方的，那么就可以确定通信方是可靠的。

数字证书认证机构（CA，Certificate Authority）颁发的公开密钥证书，可以通过 CA 对其进行验证。

进行 HTTPs 通信时，服务器会把证书发送给客户端，客户端取得其中的公开密钥之后，就可以开始加密过程。

使用 OpenSSL 这套开源程序，每个人都可以构建一套属于自己的认证机构，从而自己给自己颁发服务器证书。浏览器在访问该服务器时，会显示“无法确认连接安全性”或“该网站的安全证书存在问题”等警告消息。

HTTP1.1 和 1.0 的区别

默认持久连接节省通信量，只要客户端服务端任意一端没有明确提出断开 TCP 连接，就一直保持连接，可以发送多次 HTTP 请求

管线化，客户端可以同时发出多个 HTTP 请求，而不用一个个等待响应

断点续传

HTTP 1.1 增加 host 字段

100(Continue) Status(节约带宽) 客户端事先发送一个只带头域的请求，如果服务器因为权限拒绝了请求，就回送响应码 401（Unauthorized）

HTTP/1.1 在 1.0 的基础上加入了一些 cache 的新特性，当缓存对象的 Age 超过 Expire 时变为 stale 对象，cache 不需要直接抛弃 stale 对象，而是与源服务器进行重新激活（revalidation）。

HTTP2.0 的特点

a、HTTP/2 采用二进制格式而非文本格式

b、HTTP/2 是完全多路复用的，而非有序并阻塞的——只需一个 HTTP 连接就可以实现多个请求响应

c、使用报头压缩，HTTP/2 降低了开销

d、HTTP/2 让服务器可以将响应主动“推送”到客户端缓存中

CSRF(跨站请求伪造)、XSS(跨站脚本攻击)

xss：用户过分信任网站，放任来自浏览器地址栏代表的那个网站代码在自己本地任意执行。如果没有浏览器的安全机制限制，xss 代码可以在用户浏览器为所欲为；

csrf：网站过分信任用户，放任来自所谓通过访问控制机制的代表合法用户的请求执行网站的某个特定功能。

xss(盗用用户的身份)原理上利用的是浏览器**可以拼接成任意的 javascript**，然后**黑客拼接好 javascript** 让浏览器自动地给服务器端发出多个请求（get、post 请求）。

csrf(服务端程序员背锅，没验证好)原理上利用的是网站服务器端所有参数都是**可预先构造**的原理，然后**黑客拼接好具体请求 url**，可以引诱你提交他构造好的请求。

操作系统

进程间通信方式

管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。
命名管道 (FIFO)： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。
信号量：信号量用于实现进程间的互斥与同步，也可以用在线程上，主要有 posix 信号量和 System V 信号量，posix 信号量一般用在线程上，System V 信号量一般用在进程上，posix 信号量的函数一般都在下划线。
消息队列(message queue)： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。(优先级，大小)
共享内存(shared memory)：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。
信号 (sinal)： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生，常见的信号。
套接字(socket)： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

常见信号有哪些？：SIGINT，SIGKILL(不能被捕获)，SIGSTOP(不能被捕获)、SIGTERM(可以被捕获)，SIGSEGV，SIGCHLD，SIGALRM

标准 io 和文件 io 的区别

1.定义

标准ＩＯ：具有一定的可移植性。标准 IO 库处理很多细节。例如缓存分配，以优化长度执行 IO 等。标准的 IO 提供了三种类型的缓存。

（1）全缓存：当填满标准 IO 缓存后才进行实际的 IO 操作。

（2）行缓存：当输入或输出中遇到新行符时，标准 IO 库执行 IO 操作。

（3）不带缓存：stderr 就是了。

文件ＩＯ：文件ＩＯ称之为不带缓存的 IO（unbuffered I/O)。不带缓存指的是每个 read，write 都调用内核中的一个系统调用。也就是一般所说的低级 I/O——操作系统提供的基本 IO 服务，与 os 绑定，特定于 Unix 平台。

2.区别

首先：两者一个显著的不同点在于，标准 I/O 默认采用了缓冲机制，比如调用 fopen 函数，不仅打开一个文件，而且建立了一个缓冲区（读写模式下将建立两个缓冲区），还创建了一个包含文件和缓冲区相关数据的数据结构(FILE *)。低级 I/O 一般没有采用缓冲，需要自己创建缓冲区，不过其实在 linux 系统中，都是有使用称为内核缓冲的技术用于提高效率，读写调用是在内核缓冲区和进程缓冲区之间进行的数据复制。使用标准 IO 就不需要自己维护缓冲区了，标准 IO 库会根据 stdin/stdout 来选择缓冲类型，也就是说当你使用标准 IO 的时候，要清楚它的 stdin/stdou 是什么类型以及其默认的缓冲模式，如果不合适，你需要用 setvbuf 先设置，再使用，例如协同进程的标准输入和输出的类型都是管道，所以其默认的缓冲类型是全缓冲的，如果要使用标准 IO，就需要现设置行缓冲。对于文件 IO，只要你自己能维护好缓冲区，完全可以不用标准 IO。

其次从名字上来区分，文件 I/O 主要针对文件操作，读写硬盘等，标准 I/O，主要是打印输出到屏幕等。因为他们设备不一样，文件 io 针对的是文件，标准 io 是对控制台，操作的是字符流。对于不同设备得特性不一样，必须有不同 api 访问才最高效。

wait 和 waitpid 区别

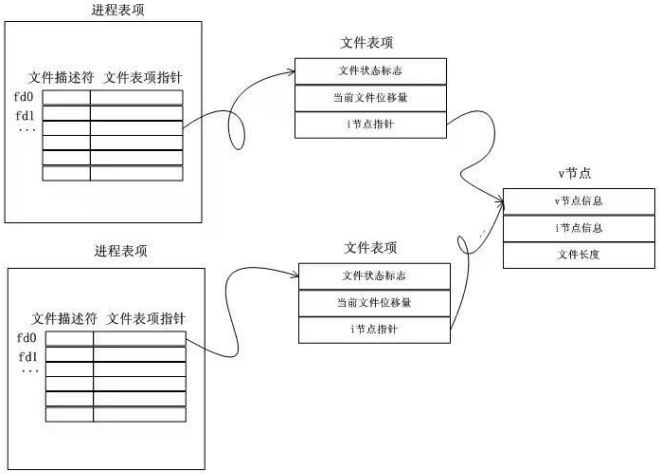
wait 会令调用者阻塞直至某个子进程终止；

waitpid 则可以通过设置一个选项来设置为非阻塞，另外 waitpid 并不是等待第一个结束的进程而是等待参数中 pid 指定的进程。

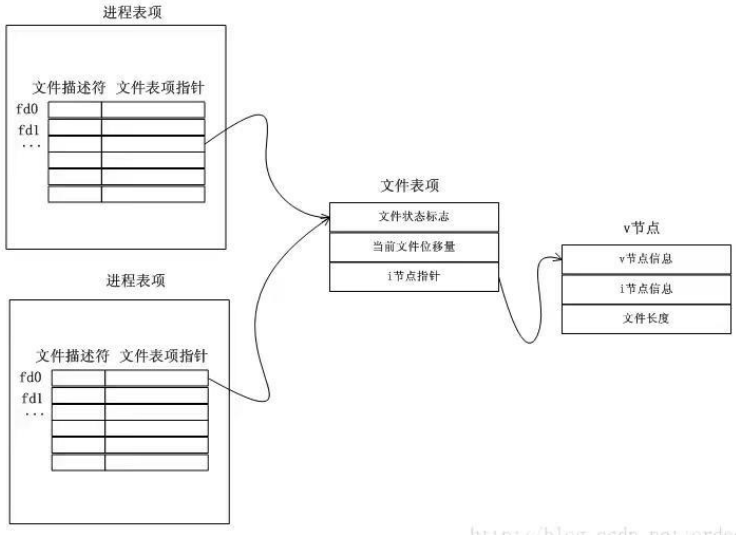
waitpid 中 pid 的含义依据其具体值而变：

pid==-1 等待任何一个子进程，此时 waitpid 的作用与 wait 相同
pid >0 等待进程 ID 与 pid 值相同的子进程
pid==0 等待与调用者进程组 ID 相同的任意子进程
pid<-1 等待进程组 ID 与 pid 绝对值相等的任意子进程
waitpid 提供了 wait 所没有的三个特性：
1 waitpid 使我们可以等待指定的进程
2 waitpid 提供了一个无阻塞的 wait
3 waitpid 支持工作控制

父进程 fork 后父子进程共享的内容
fork 之后，子进程会拷贝父进程的数据空间、堆和栈空间（实际上是采用写时复制技术），二者共享代码段。 所以在子进程中修改全局变量（局部变量，分配在堆上的内存同样也是）后，父进程的相同的全局变量不会改变。
共享 fd，以及 fd 对应的文件表项。
不同进程打开同一个文件



进程的 fork 与文件描述符的拷贝 进程的所打开文件和在 fork 后的结构图如下所示，子进程是共享父进程的文件表项。



ELF 文件的理解
section 是被链接器使用的，但是 segments 是被加载器所使用的。加载器会将所需要的 segment 加载到内存空间中运行。
1) 可重定位的对象文件(Relocatable file)(没有 segments)

这是由汇编器汇编生成的 .o 文件。后面的链接器(link editor)拿一个或一些 Relocatable object files 作为输入，经链接处理后，生成一个可执行的对象文件 (Executable file) 或者一个可被共享的对象文件(Shared object file)。我们可以使用 ar 工具将众多的 .o Relocatable object files 归档(archive)成 .a 静态库文件。
2) 可执行的对象文件(Executable file)
3) 可被共享的对象文件(Shared object file)
这些就是所谓的动态库文件，也即 .so 文件。

在 ELF 文件里面，每一个 sections 内都装载了性质属性都一样的内容，比方：
1) .text section 里装载了可执行代码；
2) .data section 里面装载了被初始化的数据；
3) .bss section 里面装载了未被初始化的数据；
4) 以 .rec 打头的 sections 里面装载了重定位条目；
5) .symtab 或者 .dynsym section 里面装载了符号信息；
6) .strtab 或者 .dynstr section 里面装载了字符串信息；
7) 其他还有为满足不同目的所设置的 section，比方满足调试的目的、满足动态链接与加载的目的等等。

把带有相同属性(比方都是只读并可加载的)的 section 都合并成所谓 segments(段)。最重要的是三个 segment：代码段，数据段和堆栈段。

linux 程序启动过程
当你在 shell 中敲入一个命令要执行时，内核会帮我们创建一个新的进程，它在往这个新进程的进程空间里面加载进可执行程序的代码段和数据段后，也会加载进动态连接器(在 Linux 里面通常就是 /lib/ld-linux.so 符号链接所指向的那个程序，它本省就是一个动态库)的代码段和数据。在这之后，内核将控制传递给动态链接库里面的代码。动态连接器接下来负责加载该命令应用程序所需要使用的各种动态库。加载完毕，动态连接器才将控制传递给应用程序的 main 函数。如此，你的应用程序才得以运行。(过程链接表（PLT）, Global Offset Table（GOT）)

产生死锁的四个必要条件：
（1） 互斥条件：一个资源每次只能被一个进程使用。
（2） 占有且等待：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
（3）不可强行占有:进程已获得的资源，在末使用完之前，不能强行剥夺。
（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。
这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

处理死锁的基本方法：
*死锁预防：通过设置某些限制条件，去破坏死锁的四个条件中的一个或几个条件，来预防发生死锁。但由于所施加的限制条件往往太严格，因而导致系统资源利用率和系统吞吐量降低。
*死锁避免：允许前三个必要条件，但通过明智的选择，确保永远不会到达死锁点，因此死锁避免比死锁预防允许更多的并发。
*死锁检测：不须实现采取任何限制性措施，而是允许系统在运行过程发生死锁，但可通过系统设置的检测机构及时检测出死锁的发生，并精确地确定于死锁相关的进程和资源，然后采取适当的措施，从系统中将已发生的死锁清除掉。

*死锁解除：与死锁检测相配套的一种措施。当检测到系统中已发生死锁，需将进程从死锁状态中解脱出来。常用方法：撤销或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程。死锁检测盒解除有可能使系统获得较好的资源利用率和吞吐量，但在实现上难度也最大。

vfork 函数
vfork() 子进程与父进程共享数据段. vfork()保证子进程先运行，在她调用 exec 或_exit 之后父进程才可能被调度运行。如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。

线程进程的区别体现在几个方面
1.因为进程拥有独立的堆栈空间和数据段，所以每当启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，系统开销比较大，而线程不一样，线程拥有独立的堆栈空间，但是共享数据段，它们彼此之间使用相同的地址空间，共享大部分数据，切换速度也比进程快，效率高，但是正由于进程之间独立的特点，使得进程安全性比较高，也因为进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。一个线程死掉就等于整个进程死掉。
2.体现在通信机制上面，正因为进程之间互不干扰，相互独立，进程的通信机制相对很复杂，譬如管道，信号，消息队列，共享内存，套接字等通信机制，而线程由于共享数据段所以通信机制很方便。。
3.属于同一个进程的所有线程共享该进程的所有资源，包括文件描述符。而不同过的进程相互独立。
4.线程必定也只能属于一个进程，而进程可以拥有多个线程而且至少拥有一个线程；

进程与线程的选择取决以下几点
1、需要频繁创建销毁的优先使用线程；因为对进程来说创建和销毁一个进程代价是很大的。
2、线程的切换速度快，所以在需要大量计算，切换频繁时用线程，还有耗时的操作使用线程可提高应用程序的响应
3、因为对 CPU 系统的效率使用上线程更占优，所以可能要发展到多机分布的用进程，多核分布用线程；
4、并行操作时使用线程，如 C/S 架构的服务器端并发线程响应用户的请求；
5、需要更稳定安全时，适合选择进程；需要速度时，选择线程更好。

守护进程
守护进程是运行在后台的一种特殊进程，不受终端控制，Linux 系统的大多数服务器就是通过守护进程实现的。一个守护进程的父进程是 init 进程。

1）创建子进程，父进程退出
2）在子进程中创建新会话
3）改变当前目录为根目
4）重设文件权限掩码
5) 关闭文件描述符

Linux 开机流程
加载 BIOS 的硬件信息与进行自我测试，并依据设置取得第一个可启动设备；
读取并执行第一个启动设备内 MBR（主引导分区）的 Boot Loader（即是 gurb 等程序）；
依据 Boot Loader 的设置加载 Kernel，Kernel 会开始检测硬件与加载驱动程序；
在硬件驱动成功后，Kernel 会主动调用 init 进程（/sbin/init），而 init 会取得 runlevel 信息；
init 执行/etc/rc.d/rc.sysinit 文件来准备软件的操作环境（如网络、时区等）；
init 执行 runlevel 的各个服务的启动（script 方式）；

init 执行/etc/rc.d/rc.local 文件；

init 执行终端机模拟程序 mingetty 来启动 login 程序，最后等待用户登录。

SQL

事务的 ACID 属性

一般来说，事务是必须满足 4 个条件（ACID）：：原子性（**A**tomicity，或称不可分割性）、一致性（**C**onsistency）、隔离性（**I**solation，又称独立性）、持久性（**D**urability）。

原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。

持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

> *在 MySQL 命令行的默认设置下，事务都是自动提交的，即执行 SQL 语句后就会马上执行 COMMIT 操作。因此要显式地开启一个事务务须使用命令 BEGIN 或 START TRANSACTION，或者执行命令 SET AUTOCOMMIT=0，用来禁止使用当前会话的自动提交。*

四种隔离级别

Read Uncommitted（读取未提交内容）

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

Read Committed（读取提交内容）

它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的 commit，所以同一 select 可能返回不同结果。

Repeatable Read（可重读）

这是 MySQL 的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB 和 Falcon 存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

Serializable（可串行化）

这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

事务的并发问题

脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个 RollBack 了操作，则后一个事务所读取的数据就会是不正确的。

不可重复读(Non-repeatable read):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是

可重复读（repeatable-read）	否	否	是()
串行 my 化（serializable）	否	否	否

数据库的各种锁的总结：

1.共享锁（又称读锁）、排它锁（又称写锁）：

InnoDB 引擎的锁机制：**InnoDB 支持事务，支持行锁和表锁用的比较多，Myisam 不支持事务，只支持表锁。**

共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。

排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的 IS 锁。

意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的 IX 锁。

2.乐观锁、悲观锁：

悲观锁：悲观锁，正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）

乐观锁：

乐观锁（ Optimistic Locking ） 相对悲观锁而言，乐观锁假设认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做（**一般是回滚事务**）。那么我们如何实现乐观锁呢，一般来说有以下 2 种方式：

两段加锁协议

1.扩展阶段

在对任何数据项的读、写之前，要申请并获得该数据项的封锁。

2.收缩阶段

每个事务中，所有的封锁请求必须先于解锁请求。

InnoDB 与 MyISAM 的比较

InnoDB 是事务型的。实现了四个标准的隔离级别，默认级别是可重复读。表是基于聚簇索引建立的，它对主键的查询性能有很高的提升。

InnoDB 支持在线热备份。

MyISAM 设计简单

MyISAM 只支持表级锁，而 InnoDB 还支持行级锁。

MyISAM 支持全文索引，地理空间索引；

InnoDB 支持外键，MyISAM 不支持

索引结构(B 树 B+树，哈希，空间，全文)

1.1 B-Tree 索引

B-Tree 索引是大多数 MySQL 存储引擎的默认索引类型。

B-树

是一种多路搜索树（并不是二叉的）：

1.定义任意非叶子结点最多只有 M 个儿子；且 M>2；

2.根结点的儿子数为[2, M]；

3.除根结点以外的非叶子结点的儿子数为[M/2, M]；

4.每个结点存放至少 M/2-1（取上整）和至多 M-1 个关键字；（至少 2 个关键字）

5.非叶子结点的关键字个数=指向儿子的指针个数-1；

6.非叶子结点的关键字：K[1], K[2], …, K[M-1]；且 K[i] < K[i+1]；

7.非叶子结点的指针：P[1], P[2], …, P[M]；其中 P[1]指向关键字小于 K[1]的
子树，P[M]指向关键字大于 K[M-1]的子树，其它 P[i]指向关键字属于(K[i-1], K[i])的子树；

8.所有叶子结点位于同一层；

B+树

B+树是 B-树的变体，也是一种多路搜索树：

1.其定义基本与 B-树同，除了：

2.非叶子结点的子树指针与关键字个数相同；

3.非叶子结点的子树指针 P[i]，指向关键字值属于[K[i], K[i+1])的子树（B-树是开区间）；

5.为所有叶子结点增加一个链指针；

6.所有关键字都在叶子结点出现；

1.2 哈希索引

基于哈希表实现，优点是查找非常快。

在 MySQL 中只有 Memory 引擎显式支持哈希索引。

哈希索引只包含哈希值和行指针，而不存储字段值，所以不能使用索引中的值来避免读取行。

1.3. 空间索引（R-Tree）

MyISAM 存储引擎支持空间索引，可以用于地理数据存储。

空间索引会从所有维度来索引数据，可以有效地使用任意维度来进行组合查询。

1.4 全文索引

MyISAM 存储引擎支持全文索引，用于查找文本中的关键词，而不是直接比较索引中的值。

索引分类(聚簇索引和 非聚簇索引)

单值索引

即一个索引只包含单个列，一个表可以有多个单列索引。

唯一索引

索引列的值必须唯一，但允许有空值。

复合索引

即一个索引包含多个列。

SQL 优化

3. 索引优化

3.1 独立的列

在进行查询时，索引列不能是表达式的一部分，也不能是函数的参数，否则无法使用索引。

例如下面的查询不能使用 actor_id 列的索引：

```
SELECT actor_id FROM sakila.actor WHERE actor_id + 1 = 5;
```

3.2 前缀索引

对于 BLOB、TEXT 和 VARCHAR 类型的列，必须使用前缀索引，只索引开始的部分字符。

对于前缀长度的选取需要根据 索引选择性 来确定：不重复的索引值和记录总数的比值。选择性越高，查询效率也越高。最大值为 1 ，此时每个记录都有唯一的索引与其对应。

3.3 多列索引

在需要使用多个列作为条件进行查询时，使用多列索引比使用多个单列索引性能更好。例如下面的语句中，最好把 actor_id 和 file_id 设置为多列索引。

```
SELECT file_id, actor_ id FROM sakila.film_actor WhERE actor_id = 1 OR film_id = 1;
```

3.4 索引列的顺序

让选择性最强的索引列放在前面。

3.5 聚簇索引

聚簇索引并不是一种索引类型，而是一种数据存储方式。
术语“聚簇”表示数据行和相邻的键值紧密地存储在一起，InnoDB 的聚簇索引的数据行存放在 B-Tree 的叶子页中。
因为无法把数据行存放在两个不同的地方，所以一个表只能有一个聚簇索引。
优点
1. 可以把相关数据保存在一起，减少 I/O 操作；
2. 因为数据保存在 B-Tree 中，因此数据访问更快。
缺点
1. 聚簇索引最大限度提高了 I/O 密集型应用的性能，但是如果数据全部放在内存，就没必要用聚簇索引。
2. 插入速度严重依赖于插入顺序，按主键的顺序插入是最快的。
3. 更新操作代价很高，因为每个被更新的行都会移动到新的位置。
4. 当插入到某个已满的页中，存储引擎会将该页分裂成两个页面来容纳该行，页分裂会导致表占用更多的磁盘空间。
5. 如果行比较稀疏，或者由于页分裂导致数据存储不连续时，聚簇索引可能导致全表扫描速度变慢。
3.6 覆盖索引
索引包含所有需要查询的字段的值。

哪些情况需要创建索引
①主键自动建立唯一索引
②频繁作为查询条件的字段应该创建索引
③查询中与其他表关联的字段，外键关系建立索引
④频繁更新的字段不适合建立索引，因为每次更新不单单是更新了记录还会更新索引
⑤WHERE 条件里用不到的字段不创建索引
⑥单键/组合索引的选择问题，who?(在高并发下倾向创建组合索引)
⑦查询中排序的字段，排序的字段若通过索引去访问将大大提高排序速度
⑧查询中统计或者分组字段
哪些情况不要创建索引
①表记录太少
②经常增删改的表
提高了查询速度，同时却会降低更新表的速度，如对表进行 INSERT、UPDATE、和 DELETE。
因为更新表时，MySQL 不仅要保存数据，还要保存一下索引文件。
数据重复且分布平均的表字段，因此应该只为最经常查询和最经常排序的数据建立索引。
③注意，如果某个数据列包含许多重复的内容，为它建立索引就没有太大的实际效果。

查询性能优化(Explain 及其它)
1. Explain
用来分析 SQL 语句，分析结果中比较重要的字段有：
- select_type : 查询类型，有简单查询、联合查询和子查询
- key : 使用的索引
- rows : 扫描的行数
2. 减少返回的列
慢查询主要是因为访问了过多数据，除了访问过多行之外，也包括访问过多列。
最好不要使用 SELECT * 语句，要根据需要选择查询的列。
3. 减少返回的行
最好使用 LIMIT 语句来取出想要的那些行。

还可以建立索引来减少条件语句的全表扫描。例如对于下面的语句，不适用索引的情况下需要进行全表扫描，而使用索引只需要扫描几行记录即可，使用 Explain 语句可以通过观察 rows 字段来看出这种差异。

```
SELECT * FROM sakila.film_actor WHERE film_id = 1;
```

4. 拆分大的 DELETE 或 INSERT 语句

如果一次性执行的话，可能一次锁住很多数据、占满整个事务日志、耗尽系统资源、阻塞很多小的但重要的查询。

```
DELEFT FROM messages WHERE create < DATE_SUB(NOW(), INTERVAL 3 MONTH);
rows_affected = 0 do { rows_affected = do_query( "DELETE FROM messages WHERE create < DATE_SUB(NOW(), INTERVAL 3 MONTH) LIMIT 10000") } while rows_affected > 0
```

分库与分表

1. 分表与分区的不同

分表，就是讲一张表分成多个小表，这些小表拥有不同的表名；而分区是将一张表的数据分为多个区块，这些区块可以存储在同一个磁盘上，也可以存储在不同的磁盘上，这种方式下表仍然只有一个。

2. 使用分库与分表的原因

随着时间和业务的发展，数据库中的表会越来越多，并且表中的数据量也会越来越大，那么读写操作的开销也会随着增大。

3. 垂直切分

将表按功能模块、关系密切程度划分出来，部署到不同的库上。例如，我们会建立商品数据库 payDB、用户数据库 userDB 等，分别用来存储项目与商品有关的表和与用户有关的表。

4. 水平切分

把表中的数据按照某种规则存储到多个结构相同的表中，例如按 id 的散列值、性别等进行划分，

5. 垂直切分与水平切分的选择

如果数据库中的表太多，并且项目各项业务逻辑清晰，那么垂直切分是首选。

如果数据库的表不多，但是单表的数据量很大，应该选择水平切分。

6. 水平切分的实现方式

最简单的是使用 merge 存储引擎。

7. 分库与分表存在的问题

(1) 事务问题

在执行分库分表之后，由于数据存储到了不同的库上，数据库事务管理出现了困难。如果依赖数据库本身的分布式事务管理功能去执行事务，将付出高昂的性能代价；如果由应用程序去协助控制，形成程序逻辑上的事务，又会造成编程方面的负担。

(2) 跨库跨表连接问题

在执行了分库分表之后，难以避免会将原本逻辑关联性很强的数据划分到不同的表、不同的库上。这时，表的连接操作将受到限制，我们无法连接位于不同分库的表，也无法连接分表粒度不同的表，导致原本只需要一次查询就能够完成的业务需要进行多次才能完成。

MongoDB 与项目

MongoDB 索引类型

单字段索引、复合索引(多个字段) 、多 key 索引(数组) 、哈希索引是指按照某个字段的 hash 值来建立索引，hash 索引只能满足字段完全匹配的查询，不能满足范围查询等。地理位置索引、文本索引(倒排索引，不支持中文)能解决快速文本查找的需求，比如有一个博客文章集合，需要根据博客的内容来快速查找，则可以针对博客内容建立文本索引。

性能分析函数（explain）

```
> db.person.find({"name":"hxc"+10000})
{ "_id" : ObjectId("4f4cee61c31a64c0b29bab31"), "name" : "hxc10000", "age" : 10000 }
>
> db.person.find({"name":"hxc"+10000}).explain()
{
        "cursor" : "BasicCursor",
        "nscanned" : 100000,
        "nscannedObjects" : 100000,
        "n" : 1,
        "millis" : 114,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "isMultiKey" : false,
        "indexOnly" : false,
        "indexBounds" : {

        }
}
```

cursor: 这里出现的是”BasicCursor",什么意思呢，就是说这里的查找采用的是“表扫描”，也就是顺序查找，很悲催啊。

nscanned: 这里是 10w，也就是说数据库浏览了 10w 个文档，很恐怖吧，这样玩的话让人受不了啊。

n: 这里是 1，也就是最终返回了 1 个文档。

millis: 这个就是我们最最最....关心的东西，总共耗时 114 毫秒。

性能分析 Profiling (显示的结果和 explain 差不多)

level 有三种级别

0 – 不开启

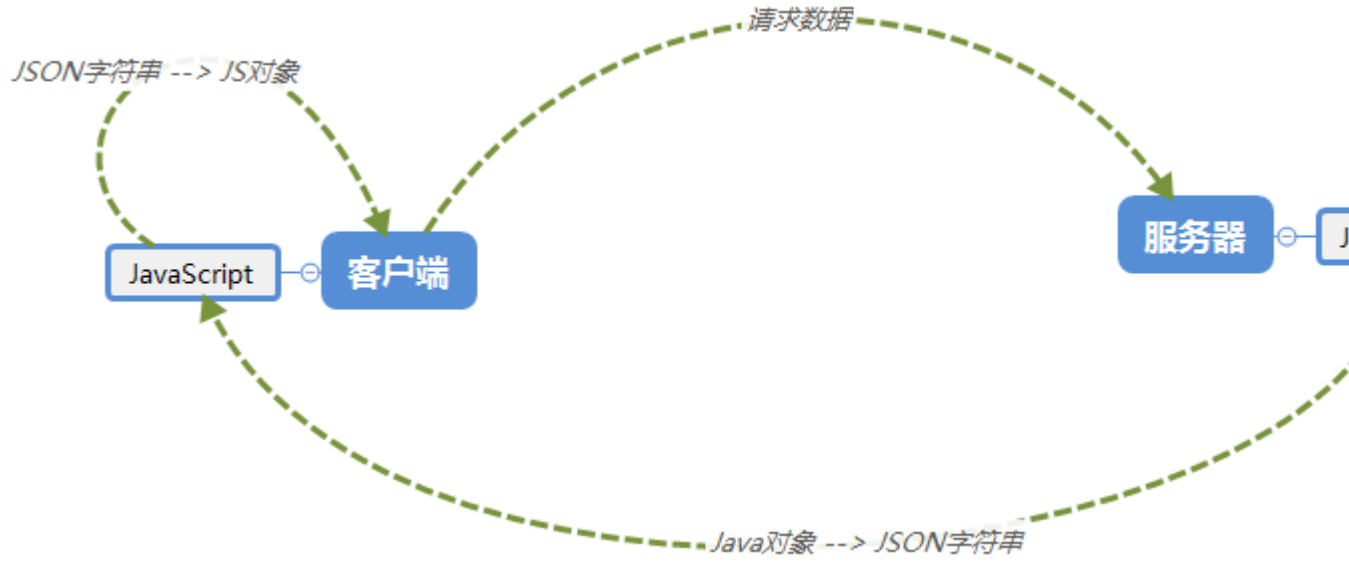
1 – 记录慢命令 (默认为>100ms)

2 – 记录所有命令

Mongodb Profile 记录是直接存在系统 db 里的，记录位置 system.profile ，我们只要查询这个 Collection 的记录就可以获取到我们的 Profile 记录了。

执行查询，然后执行 profile

BSON



- null: 表示为 null
- boolean: 表示为 true 或 false
- number: 一般的浮点数表示方式
- string: 表示为 "..."
- array: 表示为 [...]
- object: 表示为 { ... }

Collections：在 mongodb 中叫做集合，是文档的集合。无模式，可以存储各种各样的文档。类似 mysql 中的表。
Document：这里的 user 集合（“表”）有一个 document（document 可以理解为 mysql 中的记录）。
GridFS：因为 bson 对象的大小有限制，不适合存储大型文件，GridFS 文件系统为大型文件提供了存储的方案，GridFS 下的 fs 保存的是图片、视屏等大文件。

SQL 与 NoSQL 的区别
存储方式： SQL 数据存在特定结构的表中；而 NoSQL 则更加灵活和可扩展，存储方式可以省是 JSON 文档、哈希表或者其他方式。SQL 通常以数据库表形式存储数据。
表/数据集合的数据的关系： 在 SQL 中，必须定义好表和字段结构后才能添加数据，例如定义表的主键(primary key)，索引(index),触发器(trigger),存储过程(stored procedure)等。表结构可以在被定义之后更新，但是如果有比较大的结构变更的话就会变得比较复杂。在 NoSQL 中，数据可以在任何时候任何地方添加，不需要先定义表。 NoSQL 可能更加适合初始化数据还不明确或者未定的项目中。
SQL 提供了 Join 查询，可以将多个关系数据表中的数据用一条查询语句查询出来。NoSQL 没有提供。
SQL 删数据为了数据完整性，比如外键，不能随便删，而 NoSQL 是可以随意删除的。
在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用 NoSQL 数据库。
NoSQL 缺点：事务支持较弱，join 等复杂操作能力较弱，通用性较差

CAP 理论：
Consistency(一致性), 数据一致更新，所有数据变动都是同步的
Availability(可用性), 好的响应性能
Partition tolerance(分区容忍性) 可靠性
BASE：
Basically Availble --基本可用
Soft-state --软状态/柔性事务
MongoDB Wiredtiger 存储引擎实现原理
按照 Mongodb 默认的配置，WiredTiger 的写操作会先写入 Cache，并持久化到 WAL(Write ahead log)，每 60s 或 log 文件达到 2GB 时会做一次 Checkpoint，将当前的数据持久化，产生一个新的快照。Wiredtiger 连接初始化时，首先将数据恢复至最新的快照状态，然后根据 WAL 恢复数据，以保证存储可靠性。

mongodb 不支持事务，所以，在你的项目中应用时，要注意这点。无论什么设计，都不要要求 mongodb 保证数据的完整性。
但是 mongodb 提供了许多原子操作，比如文档的保存，修改，删除等，都是原子操作。

MongoDB 的锁
MongoDB 读采用的是共享锁，写采用的是排它锁。
对于大部分的读写操作，WiredTiger 使用的都是乐观锁，在全局、数据库、集合级别，WiredTiger 使用的是意向锁。当引擎探测到两个操作之间发生了冲突，将会产生一个写冲突，mongodb 将会重新执行操作。只有如删除集合等操作需要排它锁。

包括 S 锁（Shared lock），X 锁（Exclusive lock), IS 锁(Intent Share lock), IX(Intent Exclusive lock)
mongoDB 没有完整事务支持，操作原子性只到单个 document 级别。无论什么设计，都不要要求 mongodb 保证数据的完整性。但是 mongodb 对单个文档的操作提供了许多原子操作，比如文档的保存，修改(包括对单个文档修改多个字段)，删除等，都是原子操作。
针对多文档，MongoDB 是不支持事务的，只能在应用层代码层面进行控制，每写一步，判断一下。

性能监控
ifstat 看网络带宽
iostat 和 vmstat 都是 xxstat 间隔 次数
iostat 能查看到系统 IO 状态信息，从而确定 IO 性能是否存在瓶颈。
vmstat 虚拟内存统计。 可对操作系统的虚拟内存、进程、IO 读写、CPU 活动等进行监视。它是对系统的整体情况进行统计，不足之处是无法对某个进程进行深入分析。
netstat 命令是一个监控 TCP/IP 网络的非常有用的工具，它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息。-t tcp

C++语言
static 作用是什么？在 C 和 C++中有何区别？
- static 可以修饰局部变量（静态局部变量）、全局变量（静态全局变量）和函数，被修饰的变量存储位置在静态区。对于静态局部变量，相对于一般局部变量其生命周期长，直到程序运行结束而非函数调用结束，且只在第一次被调用时定义；对于静态全局变量，相对于全局变量其可见范围被缩小，只能在本文件中可见；修饰函数时作用和修饰全局变量相同，都是为了限定访问域。
- C++的 static 除了上述两种用途，还可以修饰类成员（静态成员变量和静态成员函数），静态成员变量和静态成员函数不属于任何一个对象，是所有类实例所共有。
- static 的数据记忆性可以满足函数在不同调用期的通信，也可以满足同一个类的多个实例间的通信。
- 未初始化时，static 变量默认值为 0。

指针和引用区别？
- 引用只是别名，不占用具体存储空间，只有声明没有定义；指针时具体变量，需要占用存储空间。
- 引用在声明时必须初始化为另一变量；指针声明和定义可以分开，可以先只声明指针变量而不初始化，等用到时再指向具体变量。
- 引用一旦初始化之后就不可以再改变（变量可以被引用为多次，但引用只能作为一个变量引用）；指针变量可以重新指向别的变量。
- 不存在指向空值的引用，必须有具体实体；但是存在指向空值的指针。

宏定义和内联函数(inline)区别？
- 在使用时，宏只做简单字符串替换（编译前）。而内联函数可以进行参数类型检查（编译时），且具有返回值。
- 内联函数本身是函数，强调函数特性，具有重载等功能。
- 内联函数可以作为某个类的成员函数，这样可以使用类的保护成员和私有成员。而当一个表达式涉及到类保护成员或私有成员时，宏就不能实现了。

C 的 restrict 关键字：
restrict 是 c99 标准引入的，它只可以用于限定和约束指针，并表明指针是访问一个数据对象的唯一且初始的方式. 即它告诉编译器，所有修改该指针所指向内存中内容的操作都必须通过该指针来修改, 而不能通过其它途径(其它变量或指针)来修改;这样做的好处是,能帮助编译器进行更好的优化代码,生成更有效率的汇编代码。
现在程序员用 restrict 修饰一个指针，意思就是“只要这个指针活着，我保证这个指针独享这片内存，没有‘别人’可以修改这个指针指向的这片内存，所有修改都得通过这个指针来”。由于这

个指针的生命周期是已知的，编译器可以放心大胆地把这片内存中前若干字节用寄存器 cache 起来。

Bloom 过滤器
假设布隆过滤器有 m 比特，里面有 n 个元素，每个元素对应 k 个指纹的 Hash 函数

$$(1-e^{-\frac{kn}{m}})^k$$