C++面试题目

1. 指针和引用的区别

相同点：

指针指向一块内存，它的内容是所指内存的地址；而引用则是某块内存的别名。

不同点：

●指针是一个实体，而引用仅是个别名；

●引用只能在定义时被初始化一次，之后不可变；指针可变；引用“从一而终”，指针可以“见异思迁”；

●引用没有const，指针有const，const的指针不可变；

●引用不能为空，指针可以为空；

●“sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身的大小；

●指针和引用的自增(++)运算意义不一样；

●引用是类型安全的，而指针不是 (引用比指针多了类型检查

2. 堆和栈的区别、 自由存储区、全局/静态区、常量存储区

1. 申请方式，堆（程序员），栈（系统）。
2. 申请后系统响应，堆（遍历空闲链表，找到第一个大于的。），栈（只要剩余空间大于所申请，系统提供，否则栈溢出。）
3. 申请大小限制，windows下面，栈是向低地址扩展很小（1M或2M，编译时确定），堆是向高地址。
4. 申请效率，堆慢，栈快。

全局/静态区：全局变量、静态变量。 常量存储：const。 自由存储区: new /delete 堆：malloc

全局变量和静态变量的区别

全局变量，作用域为整个项目，不管在哪个文件中，只要声明后都可以使用。

静态全局变量，作用域为定义改变量的所在文件。

3. new和delete是如何实现的，new 与 malloc的异同处

mallco实现：向进程的堆获取想要的大小，没有就向操作系统申请，失败则返回null。

free将内存控制块设定为可用。

new：先调用mallco，不成功返回bad_allco。用回调函数来处理失败的情况。 复杂类型会先mallco，然后调用构造。

delete：先析构，然后free。

4. Struct和class的区别

在C中Struct不能有函数，在c++中 权限不同。

5. define 和const的区别

1. 宏定义常量没有类型，const定义常量有类型。（宏在预处理时替换，多份拷贝，const只有一个拷贝，消耗内存小）
2. define的常量不能用指针改变去向，const可改变。
3. define可定义函数，const不可以。

编译器处理：d：预处理阶段替换。c：编译时确定值。 内存：d：不分配内存，多少次使用，多少次拷贝。c：静态存储区。 范围：d：限于当前文件。c：只在文件内有效，个个文件单独，想共享可以用extern。

6. const,static

static 成员，在类内声明，在类外初始化。 const 要成员，不能在定义出初始化，只能通过构造函数进行。static const和static一样初始化。

cosnt成员函数主要目的是防止成员函数修改对象的内容。即const成员函数不能修改成员变量的值，但可以访问成员变量。当方法成员函数时，该函数只能是const成员函数。（重载函数，用const对象调用）

static成员函数主要目的是作为类作用域的全局函数。不能访问类的非静态数据成员。类的静态成员函数没有this指针，这导致：1、不能直接存取类的非静态成员变量，调用非静态成员函数2、不能被声明为virtual

static const A; // 又可以在类内初始化，又可以类外定义。

7. 计算类大小

class A {} sizeof(A) = 1;

class A { virtual Fun(){} };: sizeof(A) = 4(32位机器)/8(64位机器);

class A { static int a; };: sizeof(A) = 1;

class A { int a; };: sizeof(A) = 4;

class A { static int a; int b; };: sizeof(A) = 4;

编译器往往会给一个空类隐含的加一个字节。

8. STL

hash表: 用开链法实现。链到一定长度要扩展数组长度。 hash函数自定。

unordered_map: 不会排序。定义==（数组+链表）

map:需要排序定义operator <. （用红黑树）

vector: 采用的数据结构很简单：线性的连续空间。不够了就重新申请拷贝。（引起空间重新配置时，指向原来的迭代器就不行了）

list：是一个环状双向链表。

deque 双向队列：一段一段的连续空间构成，由迭代器维护整体连续的假象。中控器为一个连续数组空间（映射/map），每个元素都是指针，指向一段连续线性空间（缓冲区），缓冲区大小一致。（在deque上排序很慢，可将数据拷贝至vector ，排序后拷贝回deque）

stack底层默认以deque实现，不提供迭代器。

queue底层默认是deque实现，不提供迭代器。

priority queue优先队列默认是最大堆实现。最大堆默认是vector实现的完全二叉树。

9. 重载和重写

重写 (override):

> 父类与子类之间的多态性。子类重新定义父类中有相同名称和参数的虚函数。

1) 被重写的函数不能是 static 的。必须是 virtual 的 (即函数在最原始的基类中被声明为 virtual) 。

2) 重写函数必须有相同的类型，名称和参数列表 (即相同的函数原型)

3) 重写函数的访问修饰符可以不同。尽管 virtual 是 private 的，派生类中重写改写为 public,protected 也是可以的

重载 (overload):

> 指函数名相同，但是它的参数表列个数或顺序，类型不同。但是不能靠返回类型来判断。

1、方法的重写是子类和父类之间的关系，是垂直关系；方法的重载是同一个类中方法之间的关系，是水平关

系。

2、重写要求参数列表相同；重载要求参数列表不同。

3、重写关系中，调用那个方法体，是根据对象的类型（对象对应存储空间类型）来决定；重载关系，是根据调用时的实参表与形参表来选择方法体的。

10. 虚函数实现,纯虚函数(动态绑定、静态)

虚函数表，放在对象的最前面。（为了确定偏移量）

继承中，父类虚函数放在子类前面。若有覆盖关系，则覆盖。若有多个父亲，多张表。

造成问题： 1. 安全性。 父类指针可以访问子类虚函数，访问private函数用子类指针。

析构函数需要virtual，在多态时。 构造函数不能virtual。(因为不知道儿子，还是调用自己的函数。和普通函数没区别。)

纯虚函数：接口。

不要重新定义，virtual函数的缺省参数值。 引用和指针都可以实现动态绑定。w

11.对象复用

“Is－A”代表一个类是另外一个类的一种； (继承) “Has-A”代表一个类是另外一个类的一个角色，而不是另外一个类的特殊种类。（组合）

12.零拷贝

用mmap 代替 read。

零拷贝(zero-copy)是只使用内存映射技术实现在内核与应用层之间的数据传递，由于内核与应用层使用的是同一块内存，取消了内核向用户空间的拷贝过程，会很大程度上提高系统效率。

13. 拷贝构造函数调用时机

3.何时调用复制构造函数新建一个对象并将其初始化为同类现有对象时，复制构造函数都将被调用。这在很多情况下都可能发生，最常见的情况是新对象显式地初始化为现有的对象。

例如，假设motto是一个StringBad对象，则下面4种声明都将调用复制构造函数：

```
StringBad ditto(motto);
StringBad metoo = motto;
StringBad also = StringBad(motto);
StringBad * pStringBad = new StringBad(motto);
```

其中中间的2种声明可能会使用拷贝构造函数直接创建metoo和also，也可能使用拷贝构造函数生成一个临时对象，然后将临时对象的内容赋给metoo和also，这取决于具体的实现。 最后一种声明使用motto初始化一个匿名对象，并将新对象的地址赋给pStringBad指针

14.内存对齐、泄露

1. 要和 min(#pragram pack() , 长度最长的数据成员)；对齐
2. 再和结构体对齐。

作用： 1. 平台移植，不是所有硬件能够访问任意地址上的数据。 2. 内存对其后，CPU访问速度大大提升。

用动态存储分配函数开辟空间后，使用完毕未释放，程序运行越长，整个系统崩溃。

防止方法：1. 用stl 不用数组。 2. 用智能指针。 3. new 、delete配对，尽量使用只能指针。4. 用内存池。

15.调试程序方法

1. 单元测试。
2. 打断点，日志信息。

用valgrind (内存泄露方法)

遇到coredump要怎么调试。（打断点，看线程栈信息）

16.模板

1. 重载，减少代码量。

17.成员初始化列表

1. 减少赋值操作，省去临时变量存在。
2. 如果有const成员，必须在初始化列表中做初始化。
3. 某些成员没有赋值构造，只能用初始化。

18. C11的特性

1. 右值引用,move.
2. 泛化常量表达式.
3. 外部模板 extern template，不实例化。
4. 初始化列表。
5. auto推到。 decltype（在编译器确定表达式类型）
6. 基于范围的for。
7. lamda表达式， final（不准改变） override（不重写基类） nullptr
8. 模板别名，union可定义构造函数。

9. 可变参数模板
10. static assert，控制对象对齐，实现垃圾回收。
11. tuple类型，多线程库。unorderedmap，正则表达式，智能指针。
12. 随机数，封装引用。 。。太多了不举例了。

19. C++函数调用惯例

1) 名字查找（Name Lookup）

2) 模板参数类型推导（Template Argument Deduction）

3) 重载决议（Overload Resolution）

4) 访问控制（Access Control）

5) 动态绑定（Dynamic Binding）

using namespace std;的坏处

```
#include <iostream>
using namespace std;
class Polluted {
    public:
    Polluted& operator<<(const char*)
    {
    return *this;
    }
};
int main()
{
    Polluted cout;
    cout << "Hello, Core C++!\n";
}
```

20.C++的四种强制转换

dynamic_cast

1.dynamic_cast是在运行时检查的，用于在集成体系中进行安全的向下转换downcast(当然也可以向上转换，但没必要，因为可以用虚函数实现)

即：基类指针/引用 -> 派生类指针/引用 如果源和目标没有继承/被继承关系，编译器会报错！

2.dynamic_cast是4个转换中唯一的RTTI操作符，提供运行时类型检查。

3.dynamic_cast不是强制转换，而是带有某种”咨询“性质的，如果不能转换，返回NULL。这是强制转换做不

到的。

4.源类中必须要有虚函数，保证多态，才能使用 dynamic_cast(expression)

static_cast 用法：static_cast < type-id > (expression)

该运算符把expression转换为type-id类型，在编译时使用类型信息执行转换，在转换执行必要的检测（指针越界，类型检查），其操作数相对是安全的。

但没有运行时类型检查来保证转换的安全性。

reinterpret_cast

仅仅是复制n1的比特位到d_r, 没有进行必要的分析.interpret_cast是为了映射到一个完全不同类型的意思，这个关键词在我们需要把类型映射回原有类型时用到它。我们映射到的类型仅仅是为了故弄玄虚和其他目的，这是所有映射中最危险的。(这句话是C++编程思想中的原话)

const_cast

去除const常量属性，使其可以修改 volatile属性的转换 易变类型<->不同类型.

网络面试题目

1. 建立TCP服务器的各个系统调用

TCP各层调用细节

Socket:通过分配新描述符创建新的套接字，将新的描述符返回到调用进程。任何后续的系统调用都使用创建的套接字标识。socket 系统调用还向创建的套接字描述符分配协议

Bind：将本地网络传输地址与套接字关联。

Listen：调用指示协议，服务器进程准备接受套接字上任何新传入的连接。

Accept：等待传入连接的阻塞调用。处理连接请求后，accept 将返回新的套接字描述符。将此新的套接字连接到客户端，使另外一个套接字 s 保持 LISTEN 状态，以接受进一步连接。

Connect：客户端进程通常调用 connect 系统调用，以连接到服务器进程。

Shutdown：关闭连接的任意一端或两端。如果需要关闭读取部分，则会丢弃接收缓冲区中存在的任何数据，并关闭该端的连接。对写入部分，TCP 发送任何剩余的数据，然后终止连接的写入端。

Close：可关闭或中止套接字上任何挂起的连接。

Send、Receive： 发、收。

2. 半连接、半打开、半关闭

1. 半连接

   发生在TCP3次握手中。如果A向B发起TCP请求，B也按照正常情况进行响应了，但是A不进行第3次握手，这就是半连接。

2. 半打开

   如果一方已经关闭或异常终止连接，而另一方却不知道。 我们将这样的TCP连接称为半打开（Half-Open）。

3. 半关闭

   TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力，这就是TCP的半关闭。 客户端发送FIN，另一端发送对这个FIN的ACK报文段。 此时客户端就处于半关闭。

3.内部网关协议IGP

IGP（Interior Gateway Protocol,内部网关协议）是在一个自治网络内网关（主机和路由器）间交换路由信息的协议。(RIP、OSPF、IS-IS、IGRP、EIGRP)

两类：距离矢量路由协议和链路状态路由协议。

RIP:跳来跳去。到一定跳数删除。（距离矢量）

OSPF:(链路状态) 最短路径，还可以分区。（IP层）支持简单认证，和MD5认证。 IS-IS:ISIS更适合运营商级的网络，而OSPF非常适合企业级网络。支持更多网络协议比OSPF。（链路层）

IGRP是一种距离向量型的内部网关协议（IGP）。距离向量路由协议要求每个路由器以规则的时间间隔向其相邻的路由器发送其路由表的全部或部分。随着路由信息在网络上扩散， 路由器就可以计算到所有节点的距离。

EIGRP是由距离矢量和链路状态两种路由协议混合，因此可以像距离矢量协议那样，从它的相邻路由器那里得到更新信息;也像链路状态协议那样，保存着一个拓扑表，然后通过自己的DUAL算法选择一个最优的无环路径

4. TCP与UDP区别，报文头部

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接。

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付。

3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的。UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信。

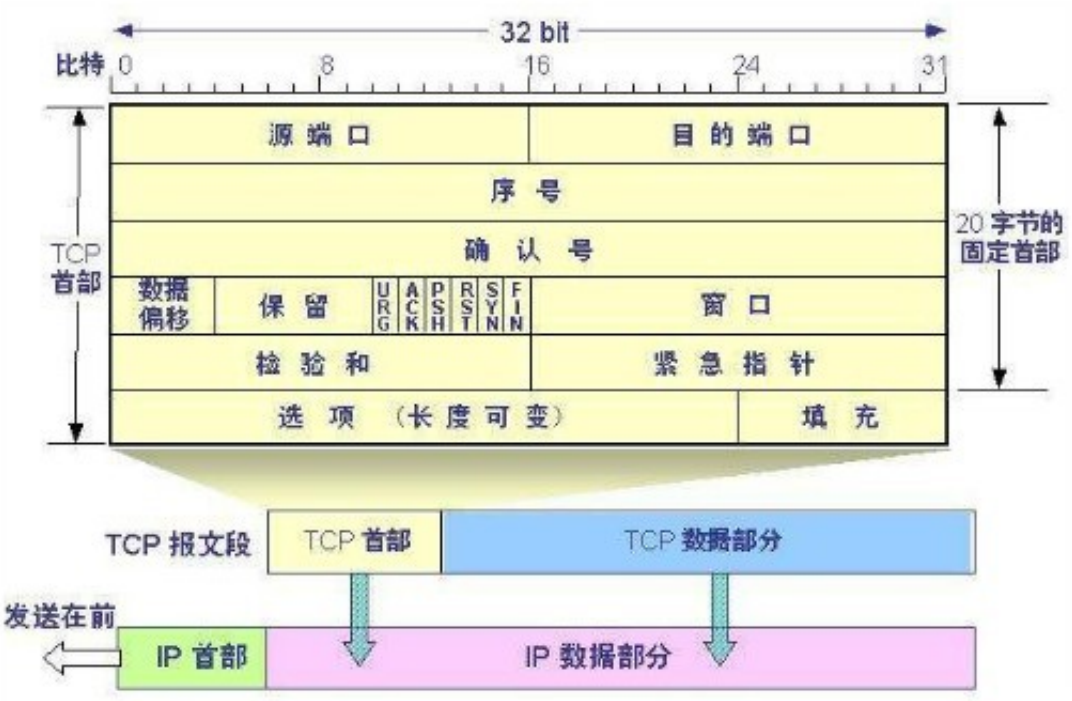
5、TCP首部开销20字节;UDP的首部开销小，只有8个字节

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道。

TCP端口	用途
21	FTP 文件传输服务
23	TELNET 终端仿真服务
25	SMTP 简单邮件传输服务
80	HTTP 超文本传输服务
110	POP3 “邮局协议版本3”使用的端口
443	HTTPS 加密的超文本传输服务
1521	Oracle数据库服务
1863	MSN Messenger的文件传输功能所使用的端口
3389	Microsoft RDP 微软远程桌面使用的端口
5631	Symantec pcAnywhere 远程控制数据传输时使用的端口
5000	MS SQL Server使用的端口

UDP端口	用途
53	DNS 域名解析服务
161	SNMP管理网络设备
5632	Symantec pcAnywhere 主控端扫描被控端时使用的端口
8000	腾讯QQ

TCP报文（20字节）



UDP报文（8字节）

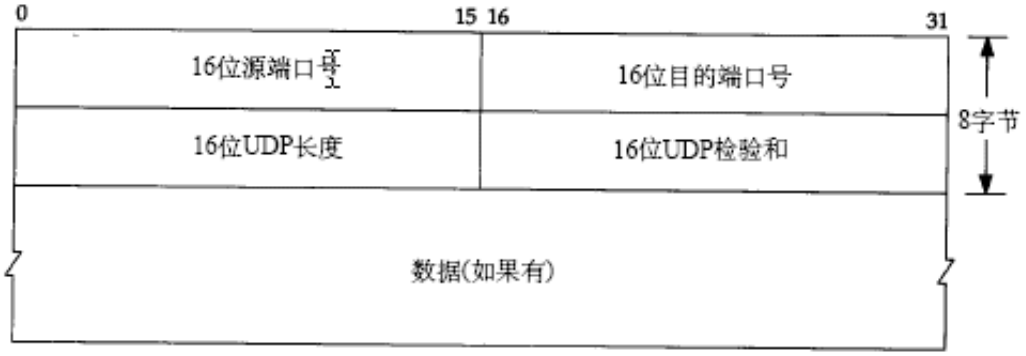
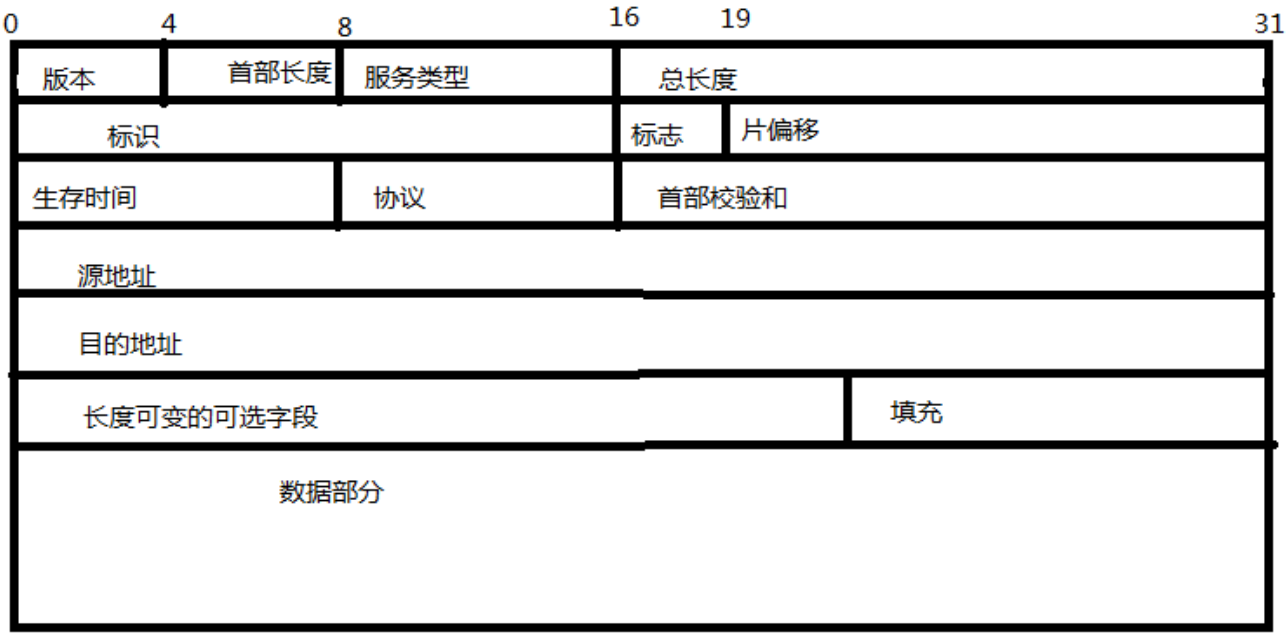


图2：UDP首部格式

IP报文（20字节）



七层模型

5.网页解析的过程与实现方法、URL的组成

URL由三部分组成：协议类型,主机名 和 路径及文件名

1.在浏览器里输入要网址。

2.浏览器查找域名的IP地址。

- 浏览器缓存 – 浏览器会缓存DNS记录一段时间。 有趣的是，操作系统没有告诉浏览器储存DNS记录的时间，这样不同浏览器会储存个自固定的一个时间（2分钟到30分钟不等）。
- 系统缓存 – 如果在浏览器缓存里没有找到需要的记录，浏览器会做一个系统调用（windows里是gethostbyname）。这样便可获得系统缓存中的记录。
- 路由器缓存 – 接着，前面的查询请求发向路由器，它一般会有自己的DNS缓存。
- ISP DNS 缓存 – 接下来要check的就是ISP缓存DNS的服务器。在这一般都能找到相应的缓存记录。
- 递归搜索 – 你的ISP的DNS服务器从跟域名服务器开始进行递归搜索，从.com顶级域名服务器到Facebook的域名服务器。一般DNS服务器的缓存中会有.com域名服务器中的域名，所以到顶级服务器的匹配过程不是那么必要了。

（一个域名对应多个IP，一个ip对应多个域名）

- 循环 DNS 是DNS查找时返回多个IP时的解决方案。举例来说， Facebook.com 实际上就对应了四个IP地址。
- 负载平衡器 是以一个特定IP地址进行侦听并将网络请求转发到集群服务器上的硬件设备。 一些大型的站点一般都会使用这种昂贵的高性能负载平衡器。
- 地理 DNS 根据用户所处的地理位置，通过把域名映射到多个不同的IP地址提高可扩展性。这样不同的服务器不能够更新同步状态，但映射静态内容的话非常好。
- Anycast是一个IP地址映射多个物理主机的路由技术。 美中不足，Anycast与TCP协议适应的不是很好，所以很少应用在那些方案中。

3.浏览器给web服务器发送一个HTTP请求

4.facebook服务的永久重定向响应

5.浏览器跟踪重定向地址

6.服务器“处理”请求

7.服务器发回一个HTML响应

8.浏览器开始显示HTML

9.浏览器发送获取嵌入在HTML中的对象

10.浏览器发送异步（AJAX）请求

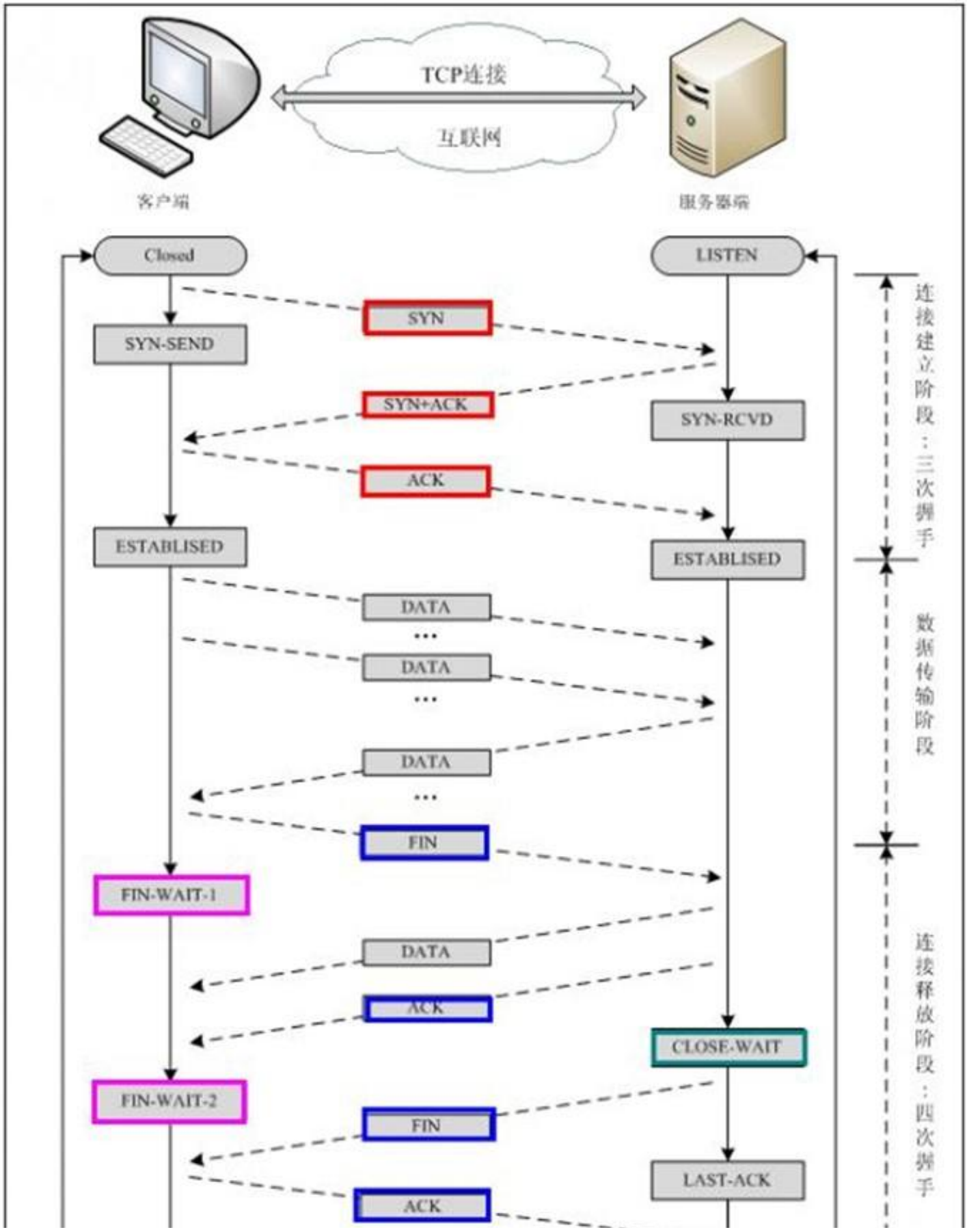
6.网络数据分片

数据报的分段和分片确实发生，分段发生在传输层，分片发生在网络层。但是对于分段来说，这是经常发生在UDP传输层协议上的情况，对于传输层使用TCP协议的通道来说，这种事情很少发生。

TCP在传输层进行分段（MSS Maxitum Segment Size）。IP在网络层进行分片。MTU（Maximum

Transmission Unit，MTU），最大传输单元

7. 三次握手中11种状态，TCP各种特性



为什么不两次握手或四次?

三次握手是最完美的建立tcp通信的过程。两次握手并不能使两台主机都确认对方已经为tcp通信做好准备，而四次握手则没有必要。

TIME_WAIT的意义（为什么要等于2MSL）？

1. 防止上一次连接中的包，迷路后重新出现，影响新连接(经过2MSL，上一次连接中所有的重复包都会消失)
2. 可靠的关闭TCP连接。在主动关闭方发送的最后一个 ack(fin) ，有可能丢失，这时被动方会重新发 fin, 如果这时主动方处于 CLOSED 状态 ，就会响应 rst 而不是 ack。所以 主动方要处于 TIME_WAIT 状态，而不能是 CLOSED 。

四次分手可以是三次嘛?

可以，是因为Server在收到Client的FIN报文后会立即ACK，然后再通知应用层来决定Server端到Client端的单向连接是否要关闭，如果应用层没数据要发则Server此时再发一个FIN，这就导致比TCP建立多了一步，因为FIN和ACK不是在同一报文中发送。如果Server的FIN和ACK在同一报文中发送，则跟TCP建立一样，是3-way的。

TCP怎么保证可靠性（面向字节流，超时重传，应答机制，滑动窗口，拥塞控制，校验等）

面向字节流:TCP给上层提供了正序的保证送达且不重复服务，正序、不重复和可靠都是TCP层要完成的工作。所以上层看起来就是一个不间断的数据流。

应答机制:Ack，序列号机制。

TCP校验

首部校验：TCP的首部字段中有一个字段是校验和，发送方将伪首部、TCP首部、TCP数据使用累加和校验的方式计算出一个数字，然后存放在首部的校验和字段里，接收者收到TCP包后重复这个过程，然后将计算出的校验和和接收到的首部中的校验和比较，如果不一致则说明数据在传输过程中出错。

MD5 最简单的就是使用MD5校验，在发送数据前将数据使用MD5加密，并将MD5摘要一起发送，接收端接收数据后将数据再次用MD5加密，如果得到的摘要和收到的摘要一致说明数据正确。 上文亚马逊的处理方式就是这样。

tcp拥塞控制和流量控制有什么区别?

拥塞控制：防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。拥塞控制所要做的都有一个前提：网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

流量控制：指点对点通信量的控制，是端到端正的问题。流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收.

拥塞控制

超时重传机制：在发送一个数据之后，就开启一个定时器，若是在这个时间内没有收到发送数据的ACK确认报文，则对该报文进行重传，在达到一定次数还没有成功时放弃并发送一个复位信号。

TCP慢启动：是TCP的一个拥塞控制机制，慢启动算法的基本思想是当TCP开始在一个网络中传输数据或发现数据丢失并开始重发时，首先慢慢的对网路实际容量进行试探，避免由于发送了过量的数据而导致阻塞。（cwnd可以是翻倍也可以是+1 门限值ssthresh限制）

快速重传和快速恢复算法：这是数据丢包的情况下给出的一种修补机制。一般来说，重传发生在超时之后，但是如果发送端接受到3个以上的重复ACK的情况下（上面的图中第二个包丢失了，就收到了两个相同的ack=11），就应该意识到，数据丢了，需要重新传递。

流量控制

TCP：滑动窗口来进行传输控制，滑动窗口的大小意味着接收方还有多大的缓冲区可以用于接收数据。发送方可以通过滑动窗口的大小来确定应该发送多少字节的数据。当滑动窗口为0时，发送方一般不能再发送数据报，但有两种情况除外，一种情况是可以发送紧急数据，例如，允许用户终止在远端机上的运行进程。另一种情况是发送方可以发送一个1字节的数据报来通知接收方重新声明它希望接收的下一字节及发送方的滑动窗口大小。

糊涂窗口综合证： TCP接收方的缓存已满，而交互式的应用进程一次只从接收缓存中读取1字节（这样就使接收缓存空间仅腾出1字节），然后向发送方发送确认，并把窗口设置为1个字节（但发送的数据报为40字节的的话）。

8. HTTP与TCP

HTTP/1.0和HTTP/1.1都把TCP作为底层的传输协议。

1.0与1.1区别

1 可扩展性（增加版本号、增加了OPTIONS方法、Upgrade头域）

2 缓存（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）

3 带宽优化 （HTTP/1.1中在请求消息中引入了range头域，它允许只请求资源的某个部分）

4 长连接 （HTTP 1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。）

5 消息传递 （HTTP/1.1中引入了Chunkedtransfer-coding，发送方将消息分割成若干个任意大小的数据块，每个数据块在发送时都会附上块的长度，最后用一个零长度的块作为消息结束的标志。）

6 Host头域 （HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。此外，服务器应该接受以绝对路径标记的资源请求。）

7 错误提示（16->24）

8 内容协商(语言因子)

get和post的基本区别

Get是最常用的方法，通常用于请求服务器发送某个资源，而且应该是安全的和幂等的。GET请求的数据会附在URL之后。

POST方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。

提交两次产生两份，应该使用POST方法，提交后一次覆盖前一次，应该使用PUT方法。

HTTP状态码

1xx	*Information*	
2xx	*Success*	
200	OK	一切都很好
201	Created	已生成一个资源
204	No Content	这个请求已被处理，但没有需要返回的内容
3xx	*Redirect*	
301	Moved	永久重定向：客户端应该更新它们的链接
302	Found	通常代表一个重写规则或类似的结果，这里表示你请求的内容，但它是在不同的地方找到的
304	Not Modified	这关系到缓存而且通常使用空的正文告诉客户端使用它们的缓存版本
307	Temporary Redirect	这个内容已迁移，但不是永久的，因此不需要更新你的链接
4xx	*Failure*	
400	Bad Request	通用的来自服务器的“不知道”消息
401	Not Authorized	你需要提供一些凭证来访问
403	Forbidden	你提供了凭证，但没有访问权限
404	Not Found	在这个 URL 中什么都没有
406	Not Acceptable	服务器无法提供适合这个请求文件头的内容
5xx	*Server Error*	
500	Internal Server Error	对于 PHP 应用程序而言，PHP 出现了某些错误并且没有给 Apache 提供关于这些错误的任何信息
503	Service Unavailable	通常用 API 显示的一个临时错误信息

http和https的区别，由http升级为https需要做哪些操作

HTTPS和HTTP的区别主要如下：

1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

客户端在使用HTTPS方式与Web服务器通信时有以下几个步骤：

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。

HTTPS优点:

（1）使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；

（2）HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

（3）HTTPS是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

（4）谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。

HTTPS缺点:

（1）HTTPS协议握手阶段比较费时，会使页面的加载时间延长近50%，增加10%到20%的耗电；

（2）HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

（3）SSL证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

（4）SSL证书通常需要绑定IP，不能在同一IP上绑定多个域名，IPv4资源不可能支撑这个消耗。

（5）HTTPS协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL证书的信用链体系并不安全，特别是在某些国家可以控制CA根证书的情况下，中间人攻击一样可行。

如何切换到HTTPS

1. 购买SSL证书。
2. 备份原网站（防止完蛋）。
3. 新版本代码改动。改为https。
4. 检查新版本页面源代码。
5. 全站301转向。全站做http到https的301转向

一个机器能够使用的端口号上限是多少，为什么？可以改变吗？那如果想要用的端口超过这个限制怎么办?

Linux上连接数，理论上可以达到没有上限，但实际上由于Linux中一切都是文件，Linux允许打开的文件的句柄数的上限为65535.

可以修改系统配置文件，修改 vi /etc/sysctl.conf

8. 对称密码和非对称密码体系、数字证书

密码博客

对称加密(也叫私钥加密):指加密和解密使用相同密钥的加密算法。有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，同时解密密钥也可以从加密密钥中推算出来。而在大多数的对称算法中，加密密钥和解密密钥是相同的，所以也称这种加密算法为秘密密钥算法或单密钥算法。

非对称加密算法需要两个密钥:公开密钥(publickey)和私有密钥(privatekey)。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密;如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。 （非对称加密算法实现机密信息交换的基本过程是:甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开;得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方;甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。）

数字证书

使用数字证书可以确保公钥不被冒充。数字证书是经过权威机构（CA）认证的公钥，通过查看数字证书，可以知道该证书是由那家权威机构签发的，证书使用人的信息，使用人的公钥。

数字签名，是用私钥将文件hash值进行加密。 收到文件的人用公钥对hash值进行解密，解开说明是BOB发的，hash值一致说明确认文件没有改动。

冒充者，将自己的公钥发给 别人，再把数字签名发给被人。别人以为他是其他人。所以得逞。 数字证书可以

证明公钥的持有人。

根证书

根证书是CA认证中心给自己颁发的证书,是信任链的起始点。安装根证书意味着对这个CA认证中心的信任。

MD5加密

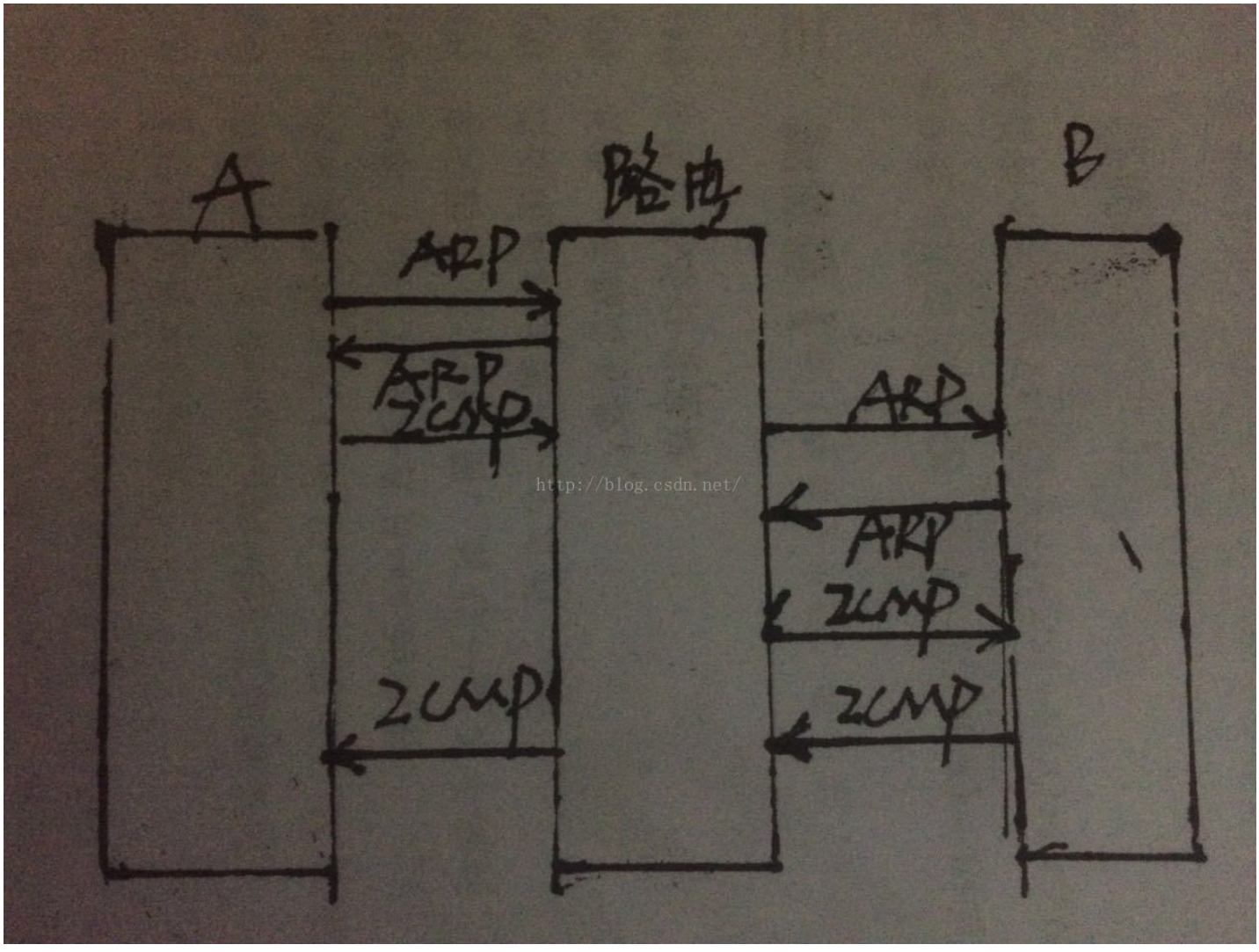
MD5（单向散列算法）的全称是Message-Digest Algorithm 5（信息-摘要算法），MD5算法的使用不需要支付任何版权费用。

RSA加密

RSA是第一个比较完善的公开密钥算法，它既能用于加密，也能用于数字签名。（可以通过公钥和私钥一起解出来）

9.一次完整的Ping过程 分包、粘包

涉及到的协议： UDP ICMP ARP OSPF DNS ARP协议把IP地址转换为MAC地址。



ICMP是（Internet Control Message Protocol）Internet控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。

分包、粘包

1.正常情况：如果Socket Client 发送的数据包，在Socket Server端也是一个一个完整接收的，那个就不会出现粘包和分包情况，数据正常读取。

2.粘包情况：Socket Client发送的数据包，在客户端发送和服务器接收的情况下都有可能发送，因为客户端发送的数据都是发送的一个缓冲buffer，然后由缓冲buffer最后刷到数据链路层的，那么就有可能把数据包2的一部分数据结合数据包1的全部被一起发送出去了，这样在服务器端就有可能出现这样的情况，导致读取的数据包包含了数据包2的一部分数据，这就产生粘包，当然也有可能把数据包1和数据包2全部读取出来。

3.分包情况：意思就是把数据包2或者数据包1都有可能被分开一部分发送出去，接着发另外的部分，在服务器端有可能一次读取操作只读到一个完整数据包的一部分。

4.在数据包发送的情况下，有可能后面的数据包分开成2个或者多个，但是最前面的部分包，黏住在前面的一个完整或者部分包的后面，也就是粘包和分包同时产生了。

数据库

1. 关系型和非关系型数据库的区别

非关系型数据库的优势：1. 性能NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。2. 可扩展性同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。关系型数据库的优势：1. 复杂查询可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。2. 事务支持使得对于安全性能很高的数据访问要求得以实现。

2. 常用SQL语句

DDL – Data Definition Language

CREATE，ALTER，DROP，TRUNCATE，COMMENT

DML – Data Manipulation Language

SELECT,INSERT,UPDATE,DELETE,CALL,EXPLAIN PLAN.

DCL – Data Control Language

TCL – Transaction Control Language

COMMIT,SAVEPOINT,ROLLBACK,SET TRANSACTION

3. 数据库中join的类型与区别

连接操作博客

inner join: 内连接是最常见的一种连接，它也被称为普通连接，只连接匹配的行（仅对满足连接条件的CROSS中的列）。它又分为等值连接（连接条件运算符为"="）和不等值连接（连接条件运算符不为"="，例如between...and）

Select * from t1,t2 where t1.id=t2.c1

full outer join:包含左、右两个表的全部行，不管另外一边的表中是否存在与它们匹配的行。

Select * from t1 full outer join t2 ont1.id = t2.c1

left outer join:包含左边表的全部行（不管右边的表中是否存在与它们匹配的行），以及右边表中全部匹配的行。

Select * from t1 left outer join t2 ont1.id =t2.c1;

right outer join: 包含右边表的全部行（不管左边的表中是否存在与它们匹配的行），以及左边表中全部匹配的行。

Select * from t1 right outer join t2 ont1.id =t2.c1;

cross join:卡尔乘积（所有可能的行对），交叉连接用于对两个源表进行纯关系代数的乘运算。它不使用连接条件来限制结果集合，而是将分别来自两个数据源中的行以所有可能的方式进行组合

Select * from t1 cross join t2;

natural join:自然连接是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉；而等值连接不会去掉重复的属性列（靠名字去匹配）

Select * from t1 naturaljoin t3;

self join：某个表和其自身连接，连接方式可以是内连接，外连接，交叉连接

Select * from t1 as t1_s, t1;

4. 数据库索引

mysql索引类型：

1. 普通索引
2. 唯一索引 （索引列值是唯一的，组合也必须唯一）
3. 主键索引 (在主键上创索引)
4. 组合索引 （多字段）

5. 全文索引 (对char varchar text上，配合match against使用)

聚集与非聚集索引的区别

SQL SERVER提供了两种索引：聚集索引和非聚集索引。

我们可以这样区分：聚集索引的叶节点就是最终的数据节点，而非聚集索引的叶节仍然是索引节点，但它有一个指向最终数据的指针。

聚集索引表示表中存储的数据按照索引的顺序存储，检索效率比非聚集索引高，但对数据更新影响较大。

非聚集索引表示数据存储在一个地方，索引存储在另一个地方，索引带有指针指向数据的存储位置，非聚集索引检索效率比聚集索引低，但对数据更新影响较小。

动作描述	使用聚集索引	使用非聚集索引
列经常被分组排序	应	应
返回某范围内的数据	应	不应
一个或极少不同值	不应	不应
小数目的不同值	应	不应
大数目的不同值	不应	应
频繁更新的列	不应	应
外键列	应	应
主键列	应	应
频繁修改索引列	不应	应

唯一性索引和主码索引的区别

1. 主键一定是唯一性索引，唯一性索引并不一定就是主键。
2. 一个表中可以有多个唯一性索引，但只能有一个主键。
3. 主键列不允许空值，而唯一性索引列允许空值。

数据库采用B+树而不是B-树 b树的原因

B树在提高了磁盘IO性能的同时并没有解决元素遍历的效率低下的问题。正是为了解决这个问题，B+树应运而生。

B+树只要遍历叶子节点就可以实现整棵树的遍历。而且在数据库中基于范围的查询是非常频繁的，而B树需要遍历整棵树，效率太低。

B+树还有一个最大的好处，方便扫库，B树必须用中序遍历的方法按序扫库，而B+树直接从叶子结点挨个扫一遍就完了，B+树支持range-query非常方便，而B树不支持。

B+树是B树的变形，它把所有的附属数据都放在叶子结点中，只将关键字和子女指针保存于内结点，内结点

完全是索引的功能，最大化了内结点的分支因子。

因为索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。

从上面分析可以看到，d越大索引的性能越好，而出度的上限取决于节点内key和data的大小，由于B+Tree内节点去掉了data域，因此可以拥有更大的出度，拥有更好的性能。

树应用

AVL树，最早的平衡二叉树，windows对进程地址空间的管理用AVL树。

红黑树，平衡二叉树，用于STL中，map和set就是用红黑树。

B/B+树：用于磁盘文件组织 数据索引和数据库索引。 Trie（字典树）:用在统计和排序大量字符串，如自动机。

最左前缀原则

组合索引，只会用最左边的列。（state, city, zip）对state有用，对city没用。

5. Mysql优化、锁

1. 字段宽度小一点。设置成notnull（不用比较null值）。
2. 索引列上不要用函数操作。
3. 类型相同字段比较。

MyISAM 和InnoDB 讲解

InnoDB和MyISAM是许多人在使用MySQL时最常用的两个表类型，这两个表类型各有优劣，视具体应用而定。基本的差别为：MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快，但是不提供事务支持，而InnoDB提供事务支持以及外部键等高级数据库功能

两种类型最主要的差别就是Innodb 支持事务处理与外键和行级锁。 INNODB的索引文件和数据文件是在一起的，MYISAM是分开的。

表锁： 开销小，加锁快；不会出现死锁；锁定力度大，发生锁冲突概率高，并发度最低。（表共享读锁（Table Read Lock）和表独占写锁（Table Write Lock））

行锁： 开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高。

页锁： 开销和加锁速度介于表锁和行锁之间；会出现死锁；锁定粒度介于表锁和行锁之间，并发度一般。

还有乐观锁，就是用版本号，再修改后再去检查是否有人动了。悲观锁是指不管如何先加锁。

数据库中事务的ACID 与分布式CAP

原子性（Atomicity）:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。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（Consistency）:数据库的约束 级联和触发机制Trigger都必须满足事务的一致性。也就是说，通过各种途径包括外键约束等任何写入数据库的数据都是有效的，不能发生表与表之间存在外键约束，但是有数据却违背这种约束性。所有改变数据库数据的动作事务必须完成，没有事务会创建一个无效数据状态。

隔离性（Isolation）:隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。

持久性（Durability）:在事务完成以后，该事务对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

目前主要有两种方式实现ACID：第一种是Write ahead logging，也就是日志式的方式(现代数据库均基于这种方式)。第二种是Shadow paging（copy数据，多的截断）。

- Consistent一致性: 一致性，这个和数据库ACID的一致性类似，但这里关注的所有数据节点上的数据一致性和正确性，而数据库的ACID关注的是在在一个事务内，对数据的一些约束。系统在执行过某项操作后仍然处于一致的状态。在分布式系统中，更新操作执行成功后所有的用户都应该读取到最新值。
- Available可用性: 每一个操作总是能够在一定时间内返回结果。需要注意“一定时间”和“返回结果”。“一定时间”是指，系统结果必须在给定时间内返回。“返回结果”是指系统返回操作成功或失败的结果。
- Partition Tolerant分区容错性: 在两个复制系统之间，如果发生了计划之外的网络连接问题，对于这种情况，有一套容错性设计来保证，是否可以对数据进行分区。这是考虑到性能和可伸缩性。

隔离性问题

	脏读	不可重复读	幻读	说明
Read uncommitted(读未提交)	√	√	√	还没提交事务就能读到（脏读）
Read committed(读已提交)	×	√	√	一个事务读取第一次后再读第二次发现不对(未加行读锁)(不可重复读)
Repeatable read（重复读）	×	×	√	在读时不能加写锁。两次读肯定是对的（加了行读锁，未加表级别锁）
Serializable （序列化）	×	×	×	第一个事务对全表修改，第二个事务追加。最后第一个事务发现有记录没被修改。（幻读）

更新丢失（Lost Update） 同时写，不加写锁的原因。

mysql：可重复读（默认） oracle：读已提交（默认） 、序列化（支持）。 sqlserver：读已提交（默认）。

数据库连接池的作用以及配置

连接池技术的主要优点包括：

（1） 缩短了连接创建时间

创建新的JDBC连接会导致联网操作和一定的JDBC驱动开销，如果这类连接是“循环”使用的，使用该方式，可避免这类不利因素。

（2）简化的编程模型

使用连接池技术时，每个单独线程能够像创建了自己的JDBC连接那样进行操作，从而允许使用直接的JDBC编程技术。

（3）受控的资源使用

如果不使用连接池技术，而是在每次需要时为线程创建新的连接，那么应用程序的资源使用将十分浪费，而且在负载较重的情况下会导致无法预期的结果。

Mysql Innodb的两种表空间方式:共享表空间方式(大文件)、独立表空间方式（小文件）

范式

1NF的定义为：符合1NF的关系中的每个属性都不可再分。

2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。

3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖

6. 分布式事务

数据一致性模型

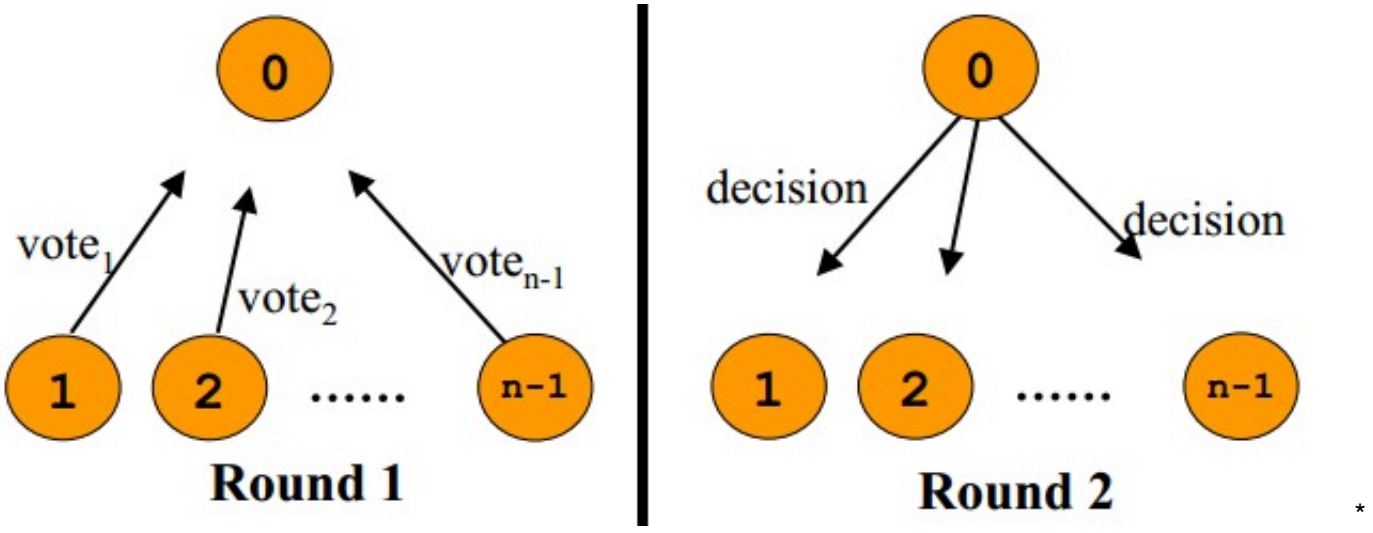
一些分布式系统通过复制数据来提高系统的可靠性和容错性，并且将数据的不同的副本存放在不同的机器。

强一致性： 当更新操作完成之后，任何多个后续进程或者线程的访问都会返回最新的更新过的值。这种是对用户最友好的，就是用户上一次写什么，下一次就保证能读到什么。根据 CAP 理论，这种实现需要牺牲可用性。

弱一致性： 系统并不保证续进程或者线程的访问都会返回最新的更新过的值。用户读到某一操作对系统特定数据的更新需要一段时间，我们称这段时间为“不一致性窗口”。系统在数据写入成功之后，不承诺立即可以读到最新写入的值，也不会具体的承诺多久之后可以读到。

最终一致性： 是弱一致性的一种特例。系统保证在没有后续更新的前提下，系统最终返回上一次更新操作的值。在没有故障发生的前提下，不一致窗口的时间主要受通信延迟，系统负载和复制副本的个数影响。DNS是一个典型的最终一致性系统。

2PC 和3PC



两个阶段的执行

1.请求阶段（commit-request phase，或称表决阶段，voting phase） 在请求阶段，协调者将通知事务参与者准备提交或取消事务，然后进入表决过程。 在表决过程中，参与者将告知协调者自己的决策：同意（事务参与者本地作业执行成功）或取消（本地作业执行故障）。

2.提交阶段（commit phase） 在该阶段，协调者将基于第一个阶段的投票结果进行决策：提交或取消。 当且仅当所有的参与者同意提交事务协调者才通知所有的参与者提交事务，否则协调者将通知所有的参与者取消事务。 参与者在接收到协调者发来的消息后将执行响应的操作。

- 两阶段提交的缺点

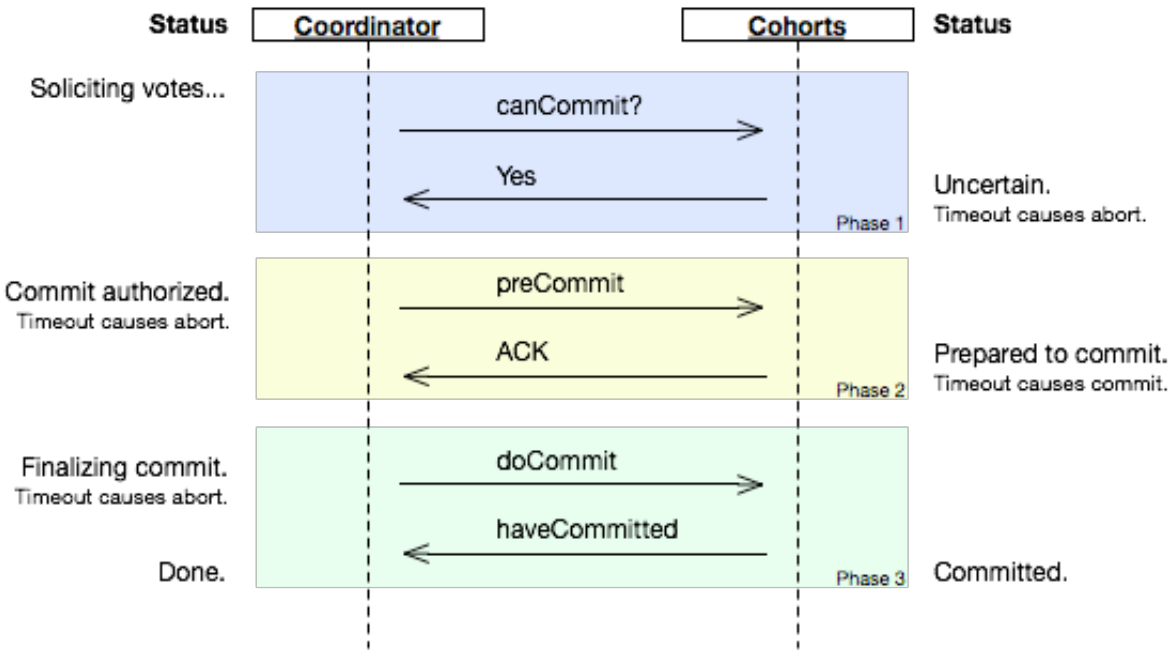
1.同步阻塞问题。执行过程中，所有参与节点都是事务阻塞型的。 当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态。

2.单点故障。由于协调者的重要性，一旦协调者发生故障。 参与者会一直阻塞下去。尤其在第二阶段，协调者发生故障，那么所有的参与者还都处于锁定事务资源的状态中，而无法继续完成事务操作。（如果是协调者挂掉，可以重新选举一个协调者，但是无法解决因为协调者宕机导致的参与者处于阻塞状态的问题）

3.数据不一致。在二阶段提交的阶段二中，当协调者向参与者发送commit请求之后，发生了局部网络异常或者在发送commit请求过程中协调者发生了故障，这回导致只有一部分参与者接受到了commit请求。 而在这部分参与者接到commit请求之后就会执行commit操作。但是其他部分未接到commit请求的机器则无法执行事务提交。于是整个分布式系统便出现了数据部一致性的现象。

- 两阶段提交无法解决的问题

当协调者出错，同时参与者也出错时，两阶段无法保证事务执行的完整性。 考虑协调者再发出commit消息之后宕机，而唯一接收到这条消息的参与者同时也宕机了。 那么即使协调者通过选举协议产生了新的协调者，这条事务的状态也是不确定的，没人知道事务是否被已经提交。



- 三个阶段的执行

1.CanCommit阶段

3PC的CanCommit阶段其实和2PC的准备阶段很像。 协调者向参与者发送commit请求，参与者如果可以提交就返回Yes响应，否则返回No响应。

2.PreCommit阶段

Coordinator根据Cohort的反应情况来决定是否可以继续事务的PreCommit操作。 根据响应情况，有以下两种可能。 A.假如Coordinator从所有的Cohort获得的反馈都是Yes响应，那么就会进行事务的预执行： 发送预提交请求。Coordinator向Cohort发送PreCommit请求，并进入Prepared阶段。 事务预提交。Cohort接收到PreCommit请求后，会执行事务操作，并将undo和redo信息记录到事务日志中。 响应反馈。如果Cohort成功的执行了事务操作，则返回ACK响应，同时开始等待最终指令。

B.假如有任何一个Cohort向Coordinator发送了No响应，或者等待超时之后，Coordinator都没有接到Cohort的响应，那么就中断事务： 发送中断请求。Coordinator向所有Cohort发送abort请求。 中断事务。Cohort收到来自Coordinator的abort请求之后（或超时之后，仍未收到Cohort的请求），执行事务的中断。

3.DoCommit阶段

该阶段进行真正的事务提交，也可以分为以下两种情况:

执行提交

A.发送提交请求。Coordinator接收到Cohort发送的ACK响应，那么他将从预提交状态进入到提交状态。并向所有Cohort发送doCommit请求。 B.事务提交。Cohort接收到doCommit请求之后，执行正式的事务提交。并在完成事务提交之后释放所有事务资源。 C.响应反馈。事务提交完之后，向Coordinator发送ACK响应。 D.完成事务。Coordinator接收到所有Cohort的ACK响应之后，完成事务。

中断事务

Coordinator没有接收到Cohort发送的ACK响应（可能是接受者发送的不是ACK响应，也可能响应超时），那么就会执行中断事务。

- 三阶段提交协议和两阶段提交协议的不同 对于协调者(Coordinator)和参与者(Cohort)都设置了超时机制（在2PC中，只有协调者拥有超时机制，即如果在一定时间内没有收到cohort的消息则默认失败）。 在2PC的准备阶段和提交阶段之间，插入预提交阶段，使3PC拥有CanCommit、PreCommit、DoCommit三个阶段。 PreCommit是一个缓冲，保证了在最后提交阶段之前各参与节点的状态是一致的。
- 三阶段提交协议的缺点

如果进入PreCommit后，Coordinator发出的是abort请求，假设只有一个Cohort收到并进行了abort操作， 而其他对于系统状态未知的Cohort会根据3PC选择继续Commit，此时系统状态发生不一致性。

7. 数据库视图

1. 简化应用程序。
2. 可以实现一定的权限控制

Linux基础

1. linux同步与异步、阻塞与非阻塞概念以及五种IO模型

同步：就是在发出一个功能调用时，在没有得到结果之前，该调用就不返回。

异步：当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者。

阻塞：指调用结果返回之前，当前线程会被挂起（线程进入非可执行状态，在这个状态下，cpu不会给线程分配时间片，即线程暂停运行）。函数只有在得到结果之后才会返回。

非阻塞：指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。

同步IO和异步IO的区别就在于：数据拷贝的时候进程是否阻塞。

阻塞IO和非阻塞IO的区别就在于：应用程序的调用是否立即返回。

5种模型

1)阻塞I/O（blocking I/O）：在调用recv()/recvfrom（）函数时，发生在内核中等待数据和复制数据的过程。

2)非阻塞I/O （nonblocking I/O）：非阻塞IO通过进程反复调用IO函数（多次系统调用，并马上返回，没有就返回错误）；在数据拷贝的过程中，进程是阻塞的。

3)I/O复用(select 、poll、epoll) （I/O multiplexing）

复用流程图

多路复用就是为了使进程能够从多个阻塞I/O中获得自己想要的数据并继续执行接下来的任务。其主要的思路就是同时监视多个文件描述符，如果有文件描述符的设定状态的被触发，就继续执行进程，如果没有任何一个文件描述符的设定状态被触发，进程进入sleep。

select:在使用select的时候，有一个叫做fd*set的集合，这个集合用来存放所有需要进行复用的I/O（文件描述符），select首先会将这个fd*set集合拷贝进内核，然后内核就会遍历这个集合（阻塞）直到某一个文件描述符满足条件（可读、可写、异常）才返回。

poll:就是给定一段时间，在这一段时间内程序处于睡眠状态一直等待某一个资源，它会在两种情况下返回①时间到了②等到了资源。

epoll:谁准备好了就提醒一下。

epoll的边缘触发和条件触发

条件触发(LT)：一有事件就会注册。并且同时支持block和no-block socket.

边缘触发(ET)：客户端只注册一次，服务端要一次性搞完。只支持no-block socket.

4)信号驱动I/O （signal driven I/O (SIGIO)）

对于给定的I/O口（一般就是对于文件描述符）设定为信号驱动I/O，则当I/O口准备好之后（读：有数据可

读；写：有空间可写），向注册它的进程发送事先约定好的信号，进程收到信号后触发signal handler进行I/O处理。

5)异步I/O （asynchronous I/O (the POSIX aio_functions)）

当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者的输入输出操作

2. 文件系统的理解（EXT4，XFS，BTRFS）

ext4 还有一些明显的限制。最大文件大小是 16 tebibytes（大概是 17.6 terabytes），这比普通用户当前能买到的硬盘还要大的多。使用 ext4 能创建的最大卷/分区是 1 exbibyte（大概是 1,152,921.5 terabytes）。通过使用多种技巧， ext4 比 ext3 有很大的速度提升。类似一些最先进的文件系统，它是一个日志文件系统，意味着它会对文件在磁盘中的位置以及任何其它对磁盘的更改做记录。纵观它的所有功能，它还不支持透明压缩、重复数据删除或者透明加密。技术上支持了快照，但该功能还处于实验性阶段。

btrfs 有很多不同的叫法，例如 Better FS、Butter FS 或者 B-Tree FS。它是一个几乎完全从头开发的文件系统。btrfs 出现的原因是它的开发者起初希望扩展文件系统的功能使得它包括快照、持久化、校验以及其它一些功能。用户当然可以继续选择创建多个分区从而无需镜像任何东西。考虑到这种情况，btrfs 能横跨多种硬盘，和 ext4 相比，它能支持 16 倍以上的磁盘空间。btrfs 文件系统一个分区最大是 16 exabyte，最大的文件大小也是 16 exabyte。

XFS 文件系统是扩展文件系统的一个扩展。XFS 是 64 位高性能日志文件系统。对 XFS 的支持大概在 2002 年合并到了 Linux 内核，到了 2009 年，红帽企业版 Linux 5.4 也支持了 XFS 文件系统。对于 64 位文件系统，XFS 支持最大文件系统大小为 8 exabytes。XFS 文件系统有一些缺陷，例如它不能压缩，删除大量文件时性能低下。目前RHEL 7.0 文件系统默认使用 XFS。

3. Linux三剑客grep,awk,sed

grep 命令 ： 强大的文本’搜索’工具

在file_name文件中找到word所在的所有行并显示。-n 为显示行号。

```
grep   -n   'word'  file_name
```

在file_name文件中找到wall 或者是well 所在的所有行并显示

```
grep   'w[ea]ll'   file_name
```

在file_name文件中找到”非well” 所在的所有行并显示

```
grep    'w[^e]ll'   file_name
```

在file_name文件中找到以The开头的所有行并显示（请与上一条命令进行区别）

```
grep   '^The'   file_name
```

在file_name文件中找到goo(任意字符)(任意字符)le的所有行并显示(即总共七个字符)

```
grep   'goo..le'   file_name
```

在file_name文件中找到g , gg , ggg等的所有行并显示（*代表重复前一个字符0～～无穷多次）

```
grep    'g*g'     file_name
```

在file_name文件中找到Tyy,Tyyy的所有行并显示(注意{}在 shell中有特殊含义，故需要转义)

```
grep   'Ty\{2,3\}'    file_name
```

sed :实现数据的替换，删除，增加，选取等(以行为单位进行处理)

删除file_name文件的2到4行

```
sed    '2,4d'    file_name
```

在第二行下新增这样两行

```
sed  '2a liu .....\
shengxi is shuai !!!'  file_name
```

把file_name文件的2到4行，替换为ni han ma a !!

```
sed   '2,4c    ni han ma   a !!!'    file_name
```

把file_name文件的2到5行打印出来(不用-n 参数，就会重复输出2到5行)

```
sed   '2,5p' -n  file_name
```

把file_name文件中的a[t]. 全部替换为p_temp-> (-i 会直接将修改写入文件，[ ] 和 . 是特殊符号，需要用\来转义一下)

```
sed  -i  's/a\[t\]\./p_temp->/g'     file_name
```

awk：以字段为单位进行处理(其实就是把一行的数据分割，然后进行处理)

把file_name 文件中的前五行的第一列，第二列的数据列出来 （以[tab]或空格键分隔）

```
awk      'NR<6{print   $1  "\t"  $2 }'     file_name
```

把/etc/passwd文件的第一列与第三列列出来（BEGIN可以让我们自己设置的分隔字符立即生效）

```
awk      'BEGIN{FS=":"}  $3<10{print $1 "\t" $3}'     /etc/passwd
```

linux常用命令 ps、 free、top

free命令

```
total：表示物理内存总量。
used：表示总计分配给缓存（包含buffers 与cache ）使用的数量，但其中可能部分缓存并未实际使用。
free：未被分配的内存。
shared：共享内存，一般系统不会用到，这里也不讨论。
buffers：系统分配但未被使用的buffers 数量。
cached：系统分配但未被使用的cache 数量
```

buffers和cached的区别：（buffer用于写，cached用于读）

a. buffers是供单个进程使用的，而cached可供多个进程使用

b. buffers是一次性的，而cached是可以反复使用的

buffers和cached的共同之处就在于它们都是临时性的存储，如果后续有进程需要使用这些内存空间，Linux会释放(free)这些临时性占用的内存。

top命令

-u<用户名>：指定用户名；

-p<进程号>：指定进程；

ps，kill，find，cp

ps 它能捕获系统在某时间的进程状态 -aux是用内存和CPU排序。

kill -9 （无条件终止）

find / -name xxx

cp 拷贝

4. 硬链接与软链接

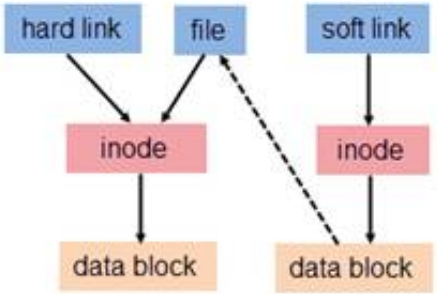
为解决文件的共享使用，Linux 系统引入了两种链接：硬链接 (hard link) 与软链接（又称符号链接，即 soft link 或 symbolic link）。链接为 Linux 系统解决了文件的共享使用，还带来了隐藏文件路径、增加权限安全及节省存储等好处。

硬连接:

- 文件有相同的 inode 及 data block；
- 只能对已存在的文件进行创建；
- 不能交叉文件系统进行硬链接的创建；
- 不能对目录进行创建，只可对文件创建；
- 删除一个硬链接文件并不影响其他有相同 inode 号的文件。

软连接:

- 软链接有自己的文件属性及权限等；
- 可对不存在的文件或目录创建软链接；
- 软链接可交叉文件系统；
- 软链接可对文件或目录创建；
- 创建软链接时，链接计数 i_nlink 不会增加；
- 删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。



软链接有不同的inode，硬连接是相同的inode

5. 文件权限、文件的三种时间、检测带宽

ls -l filename

其中： 最前面那个 - 代表的是类型
中间那三个 rw- 代表的是所有者（user）
然后那三个 rw- 代表的是组群（group）
最后那三个 r-- 代表的是其他人（other）

然后我再解释一下后面那9位数：
r 表示文件可以被读（read）
w 表示文件可以被写（write）
x 表示文件可以被执行（如果它是程序的话）
- 表示相应的权限还没有被授予

> 访问时间（access time 简写为atime）显示的是文件中的数据最后被访问的时间，比如系统的进程直接使用或通过一些命令和脚本间接使用
>
> 修改时间（modify time 简写为mtime）显示的是文件内容被修改的最后时间，比如用vi编辑时就会被改变。
>
> 状态修改时间(change time 简写为ctime)显示的是文件的权限、拥有者、所属的组、链接数发生改变时的时间。当然当内容改变时也会随之改变（即inode内容发生改变和Block内容发生改变时）

监控总体带宽使用——nload、bmon、slurm、bwm-ng、cbm、speedometer和netload

监控总体带宽使用（批量式输出）——vnstat、ifstat、dstat和collectl

每个套接字连接的带宽使用——iftop、iptraf、tcptrack、pktstat、netwatch和trafshow

每个进程的带宽使用——nethogs

操作系统

1. 进程与线程

进程:程序执行时的一个实例，即它是程序已经执行到课中程度的数据结构的汇集。从内核的观点看，进程的目的就是担当分配系统资源（CPU时间、内存等）的基本单位。

线程:进程的一个执行流，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。一个进程由几个线程组成（拥有很多相对独立的执行流的用户程序共享应用程序的大部分数据结构），线程与同属一个进程的其他的线程共享进程所拥有的全部资源。

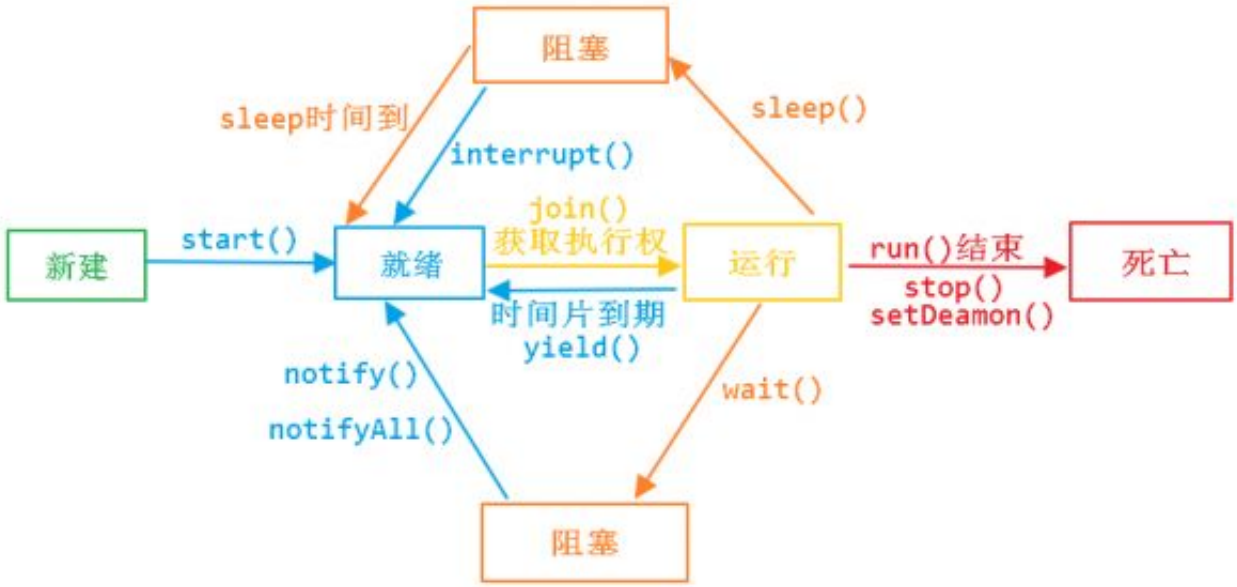
为什么用多线程

1. 和进程相比，它是一种非常"节俭"的多任务操作方式。
2. 线程间方便的通信机制。

一个进程最多可以创建线程的数目

1. 32位大约2048个（理论上） 默认1M一个线程。

线程状态图



2. 进程通信方法、线程通信方法（Linux和windows下）

进程：

（1） 管道（pipe）：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有血缘关系的进程间使用。进程的血缘关系通常指父子进程关系。

（2）有名管道（named pipe）：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通信。

（3）信号量（semophore）：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它通常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

（4）消息队列（message queue）：消息队列是由消息组成的链表，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。

（5）信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某一事件已经发生。

（6）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的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，如信号量配合使用，来实现进程间的同步和通信。

（7）套接字（socket）：套接口也是一种进程间的通信机制，与其他通信机制不同的是它可以用于不同及其

间的进程通信。

线程：

1、锁机制

1.1 互斥锁：提供了以排它方式阻止数据结构被并发修改的方法。

1.2 读写锁：允许多个线程同时读共享数据，而对写操作互斥。

1.3 条件变量：可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

2、信号量机制：包括无名线程信号量与有名线程信号量

3、信号机制：类似于进程间的信号处理。

线程间通信的主要目的是用于线程同步，所以线程没有象进程通信中用于数据交换的通信机制。

3. 一个C语言程序到执行完文件的全过程

（1）预处理（Pre-Processing）宏定义指令/条件编译指令/头文件包含指令/特殊符号（去注释、替代宏值、打开头文件和条件编译）。

（2）编译（Compiling）编译、优化阶段，经过预编译得到的输出文件中，只有常量;如数字、字符串、变量的定义，以及C语言的 关键字，main,if,else,for,while,{,}, +,-,*,\等等。

（3）汇编（Assembling）汇编过程主要的作用是汇编过程实际上指把汇编语言代码翻译成目标机器指令的过程。目标文件有：文本区(text)，数据区(data)和bss区。文本区一般包括程序的代码和常量，数据区通常存放全局变量等内容，bss区用于存放未初始化的变量或作为公共变量存储空间。在一个目标文件中，其text区从地址 0 开始，随后是data区，再后面是bss区。而要运行程序，必须装载到内存中，所以这些区的地址需要在内存中重新安排，也就是重定位。

（4）链接（Linking）将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够被操作系统装入执行的统一整体。

静态链接 ： 函数的代码将从其所在地静态链接库中被拷贝到最终的可执行程序中。这样该程序在被执行时这些代码将被装入到该进程的虚拟地址空间中。静态链接库实际上是一个目标文件的集合，其中的每个文件含有库中的一个或者一组相关函数的代码。

动态链接 ： 函数的代码被放到称作是动态链接库或共享对象的某个目标文件中。链接程序此时所作的只是在最终的可执行程序中记录下共享对象的名字以及其它少量的登记信息。在此可执行文件被执行时，动态链接库的全部内容将被映射到运行时相应进程的虚地址空间。动态链接程序将根据可执行程序中记录的信息找到相应的函数代码。

4. 进程调度方法详细介绍

进程三种状态

1. 等待态：等待某个事件的完成；
2. 就绪态：等待系统分配处理器以便运行；
3. 运行态：占有处理器正在运行。

运行态→等待态 往往是由于等待外设，等待主存等资源分配或等待人工干预而引起的。

等待态→就绪态 则是等待的条件已满足，只需分配到处理器后就能运行。

运行态→就绪态 不是由于自身原因，而是由外界原因使运行状态的进程让出处理器，这时候就变成就绪态。例如时间片用完，或有更高优先级的进程来抢占处理器等。

就绪态→运行态 系统按某种策略选中就绪队列中的一个进程占用处理器，此时就变成了运行态。

进程调度的方式

非剥夺方式　　分派程序一旦把处理机分配给某进程后便让它一直运行下去，直到进程完成或发生 某事件而阻塞时，才把处理机分配给另一个进程。

剥夺方式　　当一个进程正在运行时，系统可以基于某种原则，剥夺已分配给它的处理机，将之分配给其它进程。剥夺原则有：优先权原则、短进程优先原则、时间片原则。

进程调度算法

1. 先进先出算法（FIFO）
2. 最短CPU运行期优先调度算法(SCBF--Shortest CPU Burst First)
3. 时间片轮转法
4. 多级反馈队列

引起进程调度的原因

(1)正在执行的进程执行完毕。这时，如果不选择新的就绪进程执行，将浪费处理机资源。

(2)执行中进程自己调用阻塞原语将自己阻塞起来进入睡眠等状态。

(3)执行中进程调用了P原语操作，从而因资源不足而被阻塞；或调用了v原语操作激活了等待资源的进程队列。

(4)执行中进程提出I/O请求后被阻塞。

(5)在分时系统中时间片已经用完。

(6)在执行完系统调用等系统程序后返回用户进程时，这时可看作系统进程执行完毕，从而可调度选择一新的用户进程执行。

以上都是在不可剥夺方式下的引起进程调度的原因。在CPU执行方式是可剥夺时．还有

(7)就绪队列中的某进程的优先级变得高于当前执行进程的优先级，从而也将引发进程调度。

5. 页面置换方法详细介绍

地址映射过程中，若在页面中发现所要访问的页面不再内存中，则产生缺页中断。当发生缺页中断时操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。而用来选择淘汰哪一页的规则叫做页面置换算法。

1. 最佳置换算法（OPT）（理想置换算法）

   这是一种理想情况下的页面置换算法，但实际上是不可能实现的。该算法的基本思想是：发生缺页时，选择内存中最后要被访问的页面置换出去。这个算法唯一的一个问题就是它无法实现。

2. 先进先出置换算法（FIFO）

   最简单的页面置换算法是先入先出（FIFO）法。总是选择在主存中停留时间最长（即最老）的一页置换，即先进入内存的页，先退出内存。

3. 最近最久未使用（LRU）算法

   1. 计数器,最简单的情况是使每个页表项对应一个使用时间字段，并给CPU增加一个逻辑时钟或计数器。
   2. 栈,每当访问一个页面时，就把它从栈中取出放在栈顶上。这样一来，栈顶总是放有目前使用最多的页，而栈底放着目前最少使用的页。

4. 最近未使用页面置换算法（NRU）

   一种LRU近似算法是最近未使用算法（Not Recently Used，NUR）。它在存储分块表的每一表项中增加一个引用位，操作系统定期地将它们置为0。当某一页被访问时，由硬件将该位置1。过一段时间后，通过检查这些位可以确定哪些页使用过，哪些页自上次置0后还未使用过。就可把该位是0的页淘汰出去，因为在最近一段时间里它未被访问过。

5. 第二次机会页面置换算法

   第二次机会页面置换算法的基本思想是与FIFO相同的，但是有所改进，避免把经常使用的页面置换出去。当选择置换页面时，检查它的访问位。如果是0，就淘汰这页；如果访问位是1，就给它第二次机会，并选择下一个FIFO页面。

6. Clock置换算法

   对于第二次机会算法虽然比较合理，但是要移动链表，降低了效率，一个更好的方法把所有页面都保存在一个类似于钟面的环形链表中，表指针指向最老的页面，当发生缺页中断时，判断标志位是否为0，如果为0则淘汰该页面，把新的页面插入该位置，并设置标志位为0；如果为1，把标志为设为0，然后指针前移重复找要淘汰的页面.

7. 最少使用（LFU）置换算法

   在采用最少使用置换算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录该页面被访问的频率。该置换算法选择在最近时期使用最少的页面作为淘汰页。由于存储器具有较高的访问速度，例如100 ns，在1 ms时间内可能对某页面连续访问成千上万次，因此，通常不能直接利用计数器来记录某页被访问的次数，而是采用移位寄存器方式。每次访问某页时，便将该移位寄存器的最高位置1，再每隔一定时间(例如100 ns)右移一次。这样，在最近一段时间使用最少的页面将是∑Ri最小的页。

3. 死锁

产生死锁必要条件

1. 互斥条件。即某个资源在一段时间内只能由一个进程占有，不能同时被两个或两个以上的进程占有。
2. 不可抢占条件。进程所获得的资源在未使用完毕之前，资源申请者不能强行地从资源占有者手中夺取资源，而只能由该资源的占有者进程自行释放。
3. 占有且申请条件。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外进程占有，此时该进程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。
4. 循环等待条件。存在一个进程等待序列{P1，P2，...，Pn}，其中P1等待P2所占有的某一资源，P2等待P3所占有的某一源，.....，而Pn等待P1所占有的的某一资源，形成一个进程循环等待环。

避免死锁

1. 安全序列

   我们首先引入安全序列的定义：所谓系统是安全的，是指系统中的所有进程能够按照某一种次序分配资源，并且依次地运行完毕，这种进程序列{P1，P2，...，Pn}就是安全序列。如果存在这样一个安全序列，则系统是安全的；如果系统不存在这样一个安全序列，则系统是不安全的

2. 银行家算法

   综上所述，银行家算法是从当前状态出发，逐个按安全序列检查各客户谁能完成其工作，然后假定其完成工作且归还全部贷款，再进而检查下一个能完成工作的客户，.....。如果所有客户都能完成工作，则找到一个安全序列，银行家才是安全的。

哲学家问题解法

1. 破坏请求保持条件

   利用原子思想完成。即只有拿起两支筷子的哲学家才可以进餐，否则，一支筷子也不拿。

2. 环路等待

   1. 奇数号哲学家先拿他左边的筷子，偶数号哲学家先拿他右边的筷子。这样破坏了同方向环路，一个哲学家拿到一只筷子后，就阻止了他邻座的一个哲学家吃饭。

2. 至多允许四位哲学家进餐，将最后一个哲学家停止申请资源，断开环路。最终能保证有一位哲学家能进餐，用完释放两只筷子，从而使更多的哲学家能够进餐。
3. 哲学家申请资源总是按照资源序号先大后小的顺序，这样0．3号哲学家先右后左，但是4号哲学家先左后右，改变方向，破坏了环路。

读者写者问题

读者写者是一个非常著名的同步问题。读者写者问题描述非常简单，有一个写者很多读者，多个读者可以同时读文件，但写者在写文件时不允许有读者在读文件，同样有读者在读文件时写者也不去能写文件。

主要区别在于有写者时，新读者来了能不能读。

```
读者优先算法
1.写者、读者互斥访问文件资源。
2.多个读者可以同时访问文件资源。
3.只允许一个写者访问文件资源。

int count=0;  //用于记录当前的读者数量
semaphore mutex=1;  //用于保护更新count变量时的互斥
semaphore rw=1;  //用于保证读者和写者互斥地访问文件
writer () {  //写者进程
    while (1){
        P(rw); // 互斥访问共享文件
        Writing;  //写入
        V(rw) ;  //释放共享文件
    }
}

reader () {  // 读者进程
    while(1){
        P (mutex) ;  //互斥访问count变量
        if (count==0)  //当第一个读进程读共享文件时
            P(rw);  //阻止写进程写
        count++;  //读者计数器加1
        V (mutex) ;  //释放互斥变量count
        reading;  //读取
        P (mutex) ;  //互斥访问count变量
        count--; //读者计数器减1
        if (count==0)  //当最后一个读进程读完共享文件
            V(rw) ;  //允许写进程写
        V (mutex) ;  //释放互斥变量 count
    }
}
```

写者优先
1.写者线程的优先级高于读者线程。
2.当有写者到来时应该阻塞读者线程的队列。
3.当有一个写者正在写时或在阻塞队列时应当阻塞读者进程的读操作，直到所有写者进程完成写操作时放开读者进
4.当没有写者进程时读者进程应该能够同时读取文件。

```
int count = 0;  //用于记录当前的读者数量
semaphore mutex = 1;  //用于保护更新count变量时的互斥
semaphore rw=1;  //用于保证读者和写者互斥地访问文件
semaphore w=1;  //用于实现“写优先”

writer(){
    while(1){
        P(w);  //在无写进程请求时进入
        P(rw);  //互斥访问共享文件
        writing;  //写入
        V(rw);  // 释放共享文件
        V(w) ;  //恢复对共享支件的访问
    }
}

reader () {  //读者进程
    while (1){
        P (w) ;  // 在无写进程请求时进入
        P (mutex);  // 互斥访问count变量

        if (count==0)  //当第一个读进程读共享文件时
            P(rw);  //阻止写进程写

        count++;  //读者计数器加1
        V (mutex) ;  //释放互斥变量count
        V(w);  //恢复对共享文件的访问
        reading;  //读取
        P (mutex) ; //互斥访问count变量
        count--;  //读者计数器减1

        if (count==0)  //当最后一个读进程读完共享文件
            V(rw);  //允许写进程写

        V (mutex);  //释放互斥变量count
    }
}
```

公平竞争
1.优先级相同。
2.写者、读者互斥访问。
3.只能有一个写者访问临界区。

4.可以有多个读者同时访问临界资源。

```
/* 读者队列初始值为0，其他资源初始值为1*/
int readCount = 0;
semaphore keySignal = 1;
semaphore OneSignal = 1;
semaphore readCountSignal = 1;

reader()
{
    while(true)
    {
      wait(keySignal);         //申请令牌
      wait(readCountSignal);  //申请计数器资源
      if(!readCount)           //为零则申请文件资源
         wait(fileSrc);
      readCount++;
      signal(readCountSignal); //释放计数器资源
      signal(keySignale);      //释放令牌

      ...
      perform read operation  //执行临界区代码
      ...

      wait(readCountSignal);  //申请计数器资源
      readCount--;
      if(!readCount)                 //为零则释放文件资源
         signal(fileSrc);
    signal(readCountSignal); //释放读者计数器资源
    }
}

writer()
{
     while(true)
     {
        wait(OneSignal);     //申请令牌资源
        wait(keySignal);     //申请令牌
        wait(fileSrc);       //申请文件资源

        ...
        perform write operation //执行临界区代码
        ...

        signal(fileSrc);   //释放文件资源
        signal(keysignal); //释放令牌
        signal(OneSignal); //释放令牌资源
     }
}
```

生产者消费者模式

生产者生产数据到缓冲区中，消费者从缓冲区中取数据。

如果缓冲区已经满了，则生产者线程阻塞；

如果缓冲区为空，那么消费者线程阻塞。

4. bitmap

所谓的Bit-map就是用一个bit位来标记某个元素对应的Value， 而Key即是该元素。由于采用了Bit为单位来存储数据，因此在存储空间方面，可以大大节省。

优点：

1. 运算效率高，不许进行比较和移位；
2. 占用内存少，比如N=10000000；只需占用内存为N/8=1250000Byte=1.25M。
3. 空间复杂度不随原始集合内元素的个数增加而增加 缺点：

所有的数据不能重复。即不可对重复的数据进行排序和查找。

空间复杂度随集合内最大元素增大而线性增大。

bitmap表为：

```
a[0]--------->0-31
a[1]--------->32-63
a[2]--------->64-95
a[3]--------->96-127
void setVal(int val)
{
    bmap[val/32] |= (1<<(val%32));
    //bmap[val>>5] != (val&0x1F);//这个更快?
}

bool testVal(int val)
{
    return bmap[val/32] & (1<<(val%32));
    //return bmap[val>>5] & (val&0x1F);
}
```

4. 布隆过滤器

布隆过滤器是由一个很长的二进制向量和一系列随机映射函数组成。它可以用于检索一个元素是否在一个集合中。

优点

空间效率和查询时间都远远超过一般的算法，布隆过滤器存储空间和插入 / 查询时间都是常数O(k)。另外, 散列函数相互之间没有关系，方便由硬件并行实现。布隆过滤器不需要存储元素本身，在某些对保密要求非常严格的场合有优势

缺点

误算率是其中之一。随着存入的元素数量增加，误算率随之增加。但是如果元素数量太少，则使用散列表足矣。

另外，一般情况下不能从布隆过滤器中删除元素。

5. 读写文件时系统调用

读文件

1、进程调用库函数向内核发起读文件请求；

2、内核通过检查进程的文件描述符定位到虚拟文件系统的已打开文件列表表项；

3、调用该文件可用的系统调用函数read()

4、read()函数通过文件表项链接到目录项模块，根据传入的文件路径，在目录项模块中检索，找到该文件的inode；

5、在inode中，通过文件内容偏移量计算出要读取的页；

6、通过inode找到文件对应的address_space；

7、在address_space中访问该文件的页缓存树，查找对应的页缓存结点：

（1）如果页缓存命中，那么直接返回文件内容；

（2）如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过inode找到文件该页的磁盘地址，读取相应的页填充该缓存页；重新进行第6步查找页缓存；

8、文件内容读取成功。

写文件

前5步和读文件一致，在address_space中查询对应页的页缓存是否存在

6、如果页缓存命中，直接把文件内容修改更新在页缓存的页中。写文件就结束了。这时候文件修改位于页缓

存，并没有写回到磁盘文件中去。

7、如果页缓存缺失，那么产生一个页缺失异常，创建一个页缓存页，同时通过inode找到文件该页的磁盘地址，读取相应的页填充该缓存页。此时缓存页命中，进行第6步。

8、一个页缓存中的页如果被修改，那么会被标记成脏页。脏页需要写回到磁盘中的文件块。有两种方式可以把脏页写回磁盘：

（1）手动调用sync()或者fsync()系统调用把脏页写回

（2）pdflush进程会定时把脏页写回到磁盘

同时注意，脏页不能被置换出内存，如果脏页正在被写回，那么会被设置写回标记，这时候该页就被上锁，其他写请求被阻塞直到锁释放。

6. 线程池

可缓存线程池

- 线程数无限制。
- 有空闲线程则复用空闲线程，若无空闲线程则新建线程。
- 一定程序减少频繁创建/销毁线程，减少系统开销。

定长线程池

- 可控制线程最大并发数（同时执行的线程数）
- 超出的线程会在队列中等待

定长线程池

- 支持定时及周期性任务执行。

单线程化的线程池

- 有且仅有一个工作线程执行任务
- 所有任务按照指定顺序执行，即遵循队列的入队出队规则

线程池调度

thread pool blog

排队有三种通用策略

1. 直接提交。工作队列的默认选项是 SynchronousQueue，它将任务直接提交给线程而不保持它们。在此，如果不存在可用于立即运行任务的线程，则试图把任务加入队列将失败，因此会构造一个新的线程。此策略可以避免在处理可能具有内部依赖性的请求集时出现锁。直接提交通常要求无界

maximumPoolSizes 以避免拒绝新提交的任务。当命令以超过队列所能处理的平均数连续到达时，此策略允许无界线程具有增长的可能性。

2. 无界队列，即队列容量超出余下任务量。使用无界队列（例如，不具有预定义容量的 LinkedBlockingQueue）将导致在所有 corePoolSize 线程都忙时新任务在队列中等待。这样，创建的线程就不会超过 corePoolSize。（因此，maximumPoolSize 的值也就无效了。）当每个任务完全独立于其他任务，即任务执行互不影响时，适合于使用无界队列；例如，在 Web 页服务器中。这种排队可用于处理瞬态突发请求，当命令以超过队列所能处理的平均数连续到达时，此策略允许无界线程具有增长的可能性。

3. 有界队列，即队列容量小于余下任务量。当使用有限的 maximumPoolSizes 时，有界队列（如 ArrayBlockingQueue或具有预定义容量的 LinkedBlockingQueue）有助于防止资源耗尽，但是可能较难调整和控制。队列大小和最大池大小可能需要相互折衷：使用大型队列和小型池可以最大限度地降低 CPU 使用率、操作系统资源和上下文切换开销，但是可能导致人工降低吞吐量。如果任务频繁阻塞（例如，如果它们是 I/O 边界），则系统可能为超过您许可的更多线程安排时间。使用小型队列通常要求较大的池大小，CPU 使用率较高，但是可能遇到不可接受的调度开销，这样也会降低吞吐量。

工作队列(workQueue)

- 如果运行的线程少于 corePoolSize，则 Executor 始终首选添加新的线程，而不进行排队。（什么意思？如果当前运行的线程小于corePoolSize，则任务根本不会存放，添加到queue中，而是直接开始运行thread。
- 如果运行的线程等于或多于 corePoolSize，则 Executor 始终首选将请求加入队列，而不添加新的线程。
- 如果无法将请求加入队列，则创建新的线程，除非创建此线程超出 maximumPoolSize，在这种情况下，任务将被拒绝。

线程函数

- 函数pthread_create()创建一个线程。
- 函数pthread_join用来等待一个线程的结束。
- 初始化的函数为pthread_attr_init，这个函数必须在pthread_create函数之前调用。属性对象主要包括是否绑定、是否分离、堆栈地址、堆栈大小、优先级。默认的属性为非绑定、非分离、缺省1M的堆栈、与父进程同样级别的优先级。
- join是为了同步，将并行变成串行。
- notify_one/all wait 用于条件变量。

僵尸进程、孤儿进程、守护进程

当你运行一个程序时，它会产生一个父进程以及很多子进程。 所有这些子进程都会消耗内核分配给它们的内存和 CPU 资源。这些子进程完成执行后会发送一个 Exit 信号然后死掉。这个 Exit 信号需要被父进程所读取。父进程需要随后调用 wait 命令来读取子进程的退出状态，并将子进程从进程表中移除。

若父进程正确第读取了子进程的 Exit 信号，则子进程会从进程表中删掉。

但若父进程未能读取到子进程的 Exit 信号，则这个子进程虽然完成执行处于死亡的状态，但也不会从进程表中删掉。

僵尸进程对系统有害吗?

不会。由于僵尸进程并不做任何事情， 不会使用任何资源也不会影响其它进程， 因此存在僵尸进程也没什么坏处。 不过由于进程表中的退出状态以及其它一些进程信息也是存储在内存中的，因此存在太多僵尸进程有时也会是一些问题。

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

7. memcache

MemCache的工作流程如下：

先检查客户端的请求数据是否在memcached中，如有，直接把请求数据返回，不再对数据库进行任何操作；如果请求的数据不在memcached中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcached中（memcached客户端不负责，需要程序明确实现）；每次更新数据库的同时更新memcached中的数据，保证一致性；当分配给memcached内存空间用完之后，会使用LRU（Least Recently Used，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

Memcache是一个高性能的分布式的内存对象缓存系统，通过在内存里维护一个统一的巨大的hash表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。简单的说就是将数据调用到内存中，然后从内存中读取，从而大大提高读取速度。

8. 中断和异常

中断是CPU暂停当前工作，有计划地去处理其他的事情。中断的发生一般是可以预知的，处理的过程也是事先制定好的。处理中断时程序是正常运行的。

而异常是CPU遇到了无法响应的工作，而后进入一种非正常状态。异常的出现表明程序有缺陷。

9. 栈大小

Windows下程序栈空间的大小，VC++ 6.0 默认的栈空间是1M。 linux下非编译器决定栈大小，而是由操作系统环境决定；而在Windows平台下栈的大小是被记录在可执行文件中的（由编译器来设置)，即：windows下可以由编译器决定栈大小，而在Linux下是由系统环境变量来控制栈的大小的。

设计模式

[design partten blog]

类型	描述
创建型模式（Creational Patterns）	用于构建对象，以便它们可以从实现系统中分离出来。
结构型模式（Structural Patterns）	用于在许多不同的对象之间形成大型对象结构。
行为型模式（Behavioral Patterns）	用于管理对象之间的算法、关系和职责。

1. 创建型模式

- 单例模式（Singleton Pattern）

  保证一个类仅有一个实例，并提供一个访问它的全局访问点。(多线程安全时，要加互斥锁)

- 抽象工厂模式 （Abstract Factory Pattern）

  提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。（switch case）

- 建造者模式（Builder Pattern）

  将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。（用set函数来初始化）

- 工厂方法模式 （Factory Method Pattern）

  定义一个用于创建对象的接口，让子类决定将哪一个类实例化。Factory Method 使一个类的实例化延迟到其子类。(每个子类自己实例化)

- 原型模式（Prototype Pattern）

  用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象。（return new class(*this)）调用拷贝构造

2. 结构型模式

- 适配器模式（Adapter Pattern）

  将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。（转换头）

- 桥接模式（Bridge Pattern）

  将抽象部分与它的实现部分分离，使它们都可以独立地变化。（电风扇和开关的故事）

- 装饰者模式（Decorator Pattern）

动态地给一个对象添加一些额外的职责。就扩展功能而言，它比生成子类方式更为灵活。(星巴克，cost 虚函数)

- 组合模式（Composite Pattern）

  将对象组合成树形结构以表示“部分-整体”的层次结构。它使得客户对单个对象和复合对象的使用具有一致性。（has-A）

- 外观模式（Facade Pattern）

  为子系统中的一组接口提供一个一致的界面，Facade 模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。(京东物流 查状态)

- 享元模式（Flyweight Pattern）

  运用共享技术有效地支持大量细粒度的对象。(申请时检查一下有没有，有的话直接返回。)

- 代理模式（Proxy Pattern）

  为其他对象提供一个代理以控制对这个对象的访问。（代理点充值）

3. 行为型模式

- 模版方法模式 （Template Method Pattern）

  定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method 使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。（阿里、腾讯的故事）

- 命令模式（Command Pattern）

  将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可取消的操作。（计算器，一个负责解析，一个负责干活）

- 迭代器模式（Iterator Pattern）

  提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。(就是iterator)

- 观察者模式（Observer Pattern）

  定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新。（订阅-发布）

- 中介者模式（Mediator Pattern）

  用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。（房产中介）

- 备忘录模式 （Memento Pattern）

  在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到保存的状态。（快照）

- 解释器模式（Interpreter Pattern）

  给定一个语言，定义它的文法的一种表示，并定义一个解释器，该解释器使用该表示来解释语言中的句子。

- 状态模式（State Pattern）

  允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它所属的类。(红绿灯，类之间相互转换)

- 策略模式（Strategy Pattern）

  定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法的变化可独立于使用它的客户端。（出行方式）

- 职责链模式 （Chain of Responsibility Pattern）

  为解除请求的发送者和接收者之间耦合，而使多个对象都有机会处理这个请求。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。（请假，CEO最大权利）

- 访问者模式 （Visitor Pattern）

  表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。（清洁工和游客 去西安）

各种题目

多级哈希

多级哈希 多级hash可提供插入、查找和删除操作。 简单地，我们可以认为多级hash就是一个多维数组，行数一般在10到100之间，列数则不固定，元素大小固定。 每行可用的是一个质数，且不一样。

内存分配器

1. **基本功能**：首先将内存区(Memory Pool)以最小单位(chunk)定义出来，然后区分对象大小分别管理内存，小内存分成若干类(size class)，专门用来分配固定大小的内存块，并用一个表管理起来，降低内部碎片(internal fragmentation)。大内存则以页为单位管理, 配合小对象所在的页，降低碎片。
2. **回收及预测功能**: 当释放内存时，要能够合并小内存为大内存，根据一些条件，该保留的就保留起来，在下次使用时可以快速的响应。不需要保留时，则释放回系统，避免长期占用。

3. **优化多线程下性能问题**：针对多线程环境下，每个线程可以独立占有一段内存区间，被称为TLS(Thread Local Storage)，这样线程内操作时可以不加锁，提高性能。

hash冲突解决

1. 开放定址法。（后移一位，或者平方，或伪随机数）
2. 链地址法。
3. 再Hash法。
4. 公共溢出区

一致性hash

一个环，方便迁移。 虚拟节点。根据机器能力确定。

协程

zhihu链接

协程就是用户空间下的线程。

线程共享的环境包括：

1.进程代码段

2.进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)

3.进程打开的文件描述符、信号的处理器、进程的当前目录和进程用户ID与进程组ID。

线程独立的资源包括：

1.线程ID

每个线程都有自己的线程ID，这个ID在本进程中是唯一的。进程用此来标识线程。

2.寄存器组的值

由于线程间是并发运行的，每个线程有自己不同的运行线索，当从一个线程切换到另一个线程上 时，必须将原有的线程的寄存器集合的状态保存，以便将来该线程在被重新切换到时能得以恢复。

3.线程的堆栈

堆栈是保证线程独立运行所必须的。线程函数可以调用函数，而被调用函数中又是可以层层嵌套的，所以线程必须拥有自己的函数堆栈， 使得函数调用可以正常执行，不受其他线程的影响。

4.错误返回码

由于同一个进程中有很多个线程在同时运行，可能某个线程进行系统调用后设置了errno值，而在该线程还没有处理这个错误，另外一个线程就在此时被调度器投入运行，这样错误值就有可能被修改。所以，不同的线程应该拥有自己的错误返回码变量。

5.线程的信号屏蔽码

由于每个线程所感兴趣的信号不同，所以线程的信号屏蔽码应该由线程自己管理。但所有的线程都 共享同样的信号处理器。

6.线程的优先级

由于线程需要像进程那样能够被调度，那么就必须要有可供调度使用的参数，这个参数就是线程的优先级。

缓冲区溢出

起因：

计算机对接收的输入数据没有进行有效的检测（理想的情况是程序检查数据长度并不允许输入超过缓冲区长度的字符），向缓冲区内填充数据时超过了缓冲区本身的容量，而导致数据溢出到被分配空间之外的内存空间，使得溢出的数据覆盖了其他内存空间的数据。

危害：

而缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，比如得到shell，然后为所欲为。

内存映射

比如你的硬盘上有一个文件，你可以使用linux系统提供的mmap接口，讲这个文件映射到进程一块虚拟地址空间，这块空间会对应一块物理内存，当你读写这块物理空间的时候，就是在读取实际的磁盘文件，就是这么直接，这么高效。通常诸如共享库的加载都是通过内存映射的方式加载到物理内存的。

df命令底层实现

1. 读取/etc/mtab文件解析成一个mntent结构体。
2. 调用get*fs*usage。

百分比计算：已用和未用，不等于总共，应为有一部分元数据信息。

一个文件被删除，但空间没被释放。

当文件进程锁定，或有进程一直向这个文件写数据就会出现这个问题。（元数据没有被清掉）

弱类型、强类型、动态语言、静态语言

弱类型语言允许将一块内存看做任意类型。（c、c++）

强类型语言，没有强制类型转换不能相互操作。（java、c#、python）

动态语言类型检测是在运行时检查、静态语言是在编译时检查。

python修饰器

不修改原来的函数，添加功能。

腾讯题目

ping过程到底发生了什么。

ping

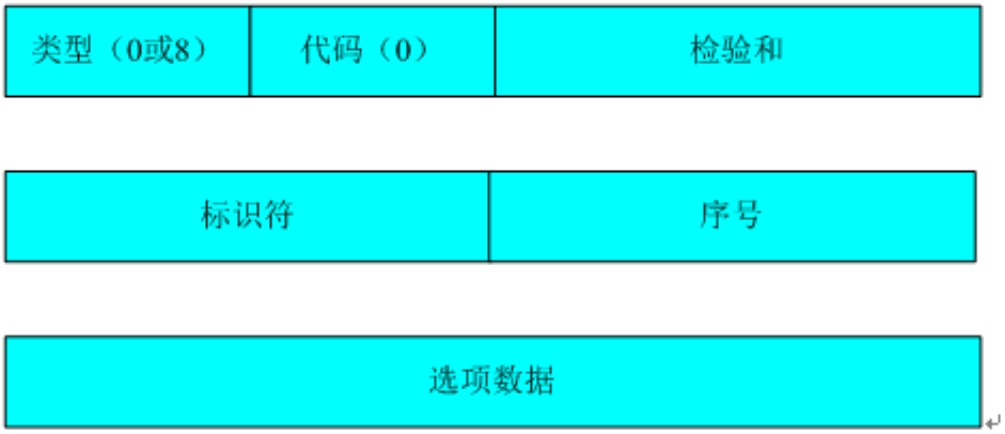
1. ICMP 创建一个回应请求数据包(数据域中只包含字母)。
2. ICMP将该有效负荷交给IP协议，IP用它创建一个分组。
3. IP协议判断目的IP是处于LAN还是某个远程网络。
4. 由于IP协议判定这是一个远程的跨网络请求，要将这一分组路由到远程网络，就必须将它发给默认路由。
5. 主机 Host A(192.168.1.2)要将分组数据发送到默认网关。(192.168.1.1), 就必须要知道Router A接口F0/0的MAC地址。
6. 检查主机ARP缓存，查看默认网关IP地址是否已经被解析为一个硬件地址。
7. 一旦分组和硬件地址交给链路层，局域网驱动程序负责选择合适的局域网类型(本例为以太网)的介质访问方式，创建数据帧。
8. 一旦帧创建完成，该帧被提交到物理层；物理层将 1bit/次的方式将数据发送到物理介质中。
9. 本冲突域中的每台主机(Host B)都会接收到这些比特，并将其重新组装成帧。
10. 将分组数据从帧中取出，并将其他部分丢弃；然后，分组被递交给以太网类型字段中列出的协议（本例为IP协议）。
11. IP接受该分组，并检查其目的IP。
12. 如果路由表中没有网络 192.168.20.0 的路由表项，则立即丢弃该分组；并向源主机Host A发送目标网络不可达的ICMP报文。
13. 如果在路由表中查找到了相关的路由表项，则分组将被交换到指定的输出接口 -- 本例为F0/1。
14. 路由器Router A将此分组交换到F0/1的缓冲区内。
15. F0/1需要获得目的方设备的MAC地址(本例为Router B的F0/0 MAC地址)
16. 帧创建完成后，将其交给物理层，并由物理层逐比特发送到物理介质中。
17. Router B执行与Router A相同的操作 (即step9 - step16)
18. Host C 接收到该帧，并立即运行CRC运算，若与FCS匹配，则检测帧中的目标MAC，如果同样相同，检查以太网类型值，判断网络层协议 -- 本例为IP。
19. 在网络层，IP接受到该分组，并对其头部进行CRC；若相同，则检查目标IP是否与本机相同；若相同，检查分组的协议字段，了解上层的交付对象 -- 本例为ICMP。

20. 负荷提交给ICMP；ICMP知道此为回应请求信号，ICMP将应答此请求。
21. 此时一个新的有效负荷产生，目标为Host A。
22. 将其提交给IP层，IP判断目的地址为本地LAN还是远程主机(本例为远程主机)，此分组首先发到默认网关。
23. 重复请求包所经历的步骤，直到将应答包发送到Host B的ICMP协议为止。
24. Host A的ICMP通过用户界面发送一个！表示已经收到这个回复。

TTL：生存时间. 如果同一服务器不同的ip,你ping这些 ip得到的ttl越高（经过转发的路由器少），延时越小，说明直连该ip会更快.

destination host unreachable 表示到达目标的路由都没有。 原因为： 1. 网线没插到网卡。2. 子网掩码设置错误。 3. DHCP故障。4.路由表返回了错误信息。

request timed out就是超时，表示IP是不通的。 原因为： 1. IP设置错误。 2. 对方关机，或网络上没这个地址。 3. 对方确实存在，但是防火墙设置了ICMP包过滤。 4. 对方与自己不在同一个网段，路由器无法找到。



ICMP包：Type和code可以组合成各种。

> 1. ping localhost: localhost的IP地址一般为127.0.0.1, 也称loopback(环回路由)；如果此时ping不通，则表示协议栈有问题；ping 该地址不经过网卡，仅仅是软件层面
> 2. ping 本机IP： ping 本机IP其实是从驱动到网卡，然后原路返回；所以如果此时ping不通，则表示网卡驱动有问题，或者NIC硬件有问题；
> 3. ping 网关： 所谓网关，就是连接到另外一个网络的“关卡”， 一般为离我们终端最近的路由器；可以使用ipconfig (windows)或ifconfig (Linux)查看；若此时ping不通，则为主机到路由器间的网络故障；
> 4. ping 目的IP: 若此步骤不成功，应该就是路由器到目的主机的网络有问题。

写磁盘的系统调用。

上面有。

tcp如何保证不丢。

- 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。
- 数据校验
- 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。

- 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。
- 拥塞控制：当网络拥塞时，减少数据的发送

如何在栈上面new。

预先分配多个对象的空间，比方说你要栈上就用数组这样咯。然后重载new为你希望的行为。跟内存池一个概念吧。

能用memset去操作new出来的对象吗?

在C++中，针对类对象除了用构造函数初始化对象外，也可以使用memset来进行初始化操作（确实有这种情况，不得已而为之，请大家不要把鸡蛋砸过来！）。但是一定要注意以下所说的这种情况：如果类包含虚拟函数，则不能用 memset 来初始化类对象。

如何在main函数之前运行代码

1. 指定程序入口。（物理（直接）的方法）
2. 定义全局变量，（构造函数、lambda表达式对象）

整个程序编译连接过程

上面有。

多线程各种。

c++初始化列表好处

1. C++ 类对像构造时，需要对类成员变量完成初始化赋值操作。使用初始化列表完成这步操作在性能上有益处。（直接用copy构造，否则要先用默认构造，再赋值。）

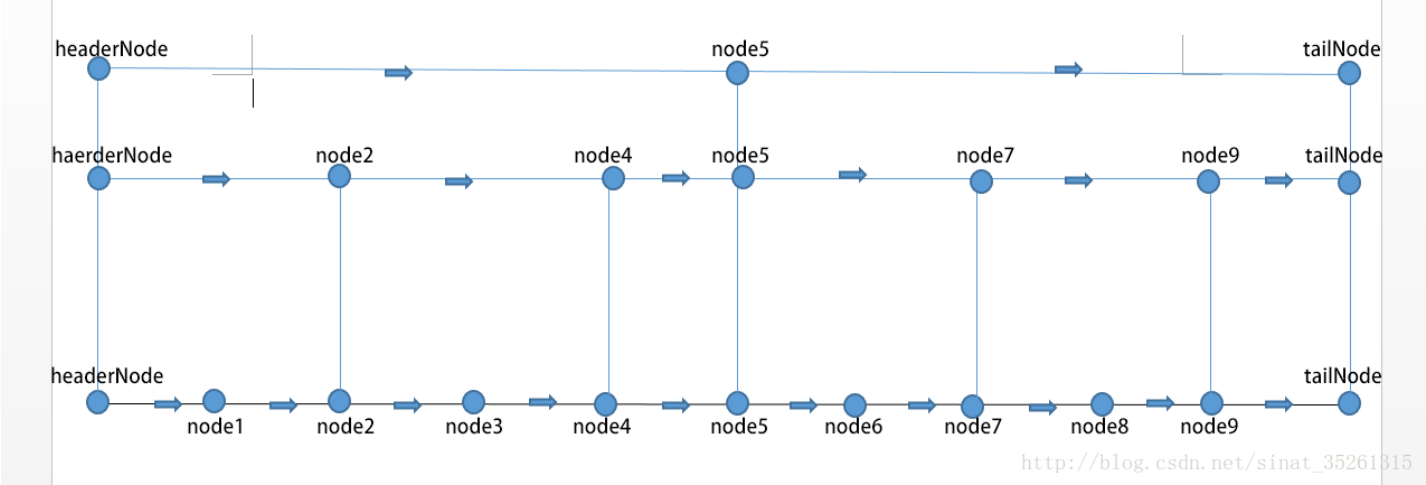
为什么会出现丢包。

（物理线路故障、设备故障、网络拥塞、路由错误）

美团

skip table

跳表：



索引

数据库连接池设计

连接池1 连接池2

无锁并发队列

无锁队列

1. 最普通的用互斥锁
2. 读写锁，设置优先级，在插入时设置，加写锁后，读锁不能被拿到。（链表级别读写锁）
3. 允许并发插入的链表（链表级别读写锁，节点上互斥锁）（插入时先加链表上读锁，防止被删除，然后在插入时给节点加互斥锁）
4. 无锁CAS

```
T Stack<T>::pop( )
{
    while (1) {
        Node* result = top;
        if (result == NULL)
           throw std::string("Cannot pop from empty stack");
        if (top && __sync_bool_compare_and_swap(&top, result, result->next)) { // CA
            return top->data;
        }
    }
}
```

__sync_bool_compare_and_swap

HBASE

HBASE

优点：

1. 海量存储
2. 列式存储（列族，一群列组成一个族）（列族最好小于或者等于3）
3. 极易扩展
4. 高并发
5. 稀疏

由于Hbase只支持3中查询方式：

- 基于Rowkey的单行查询
- 基于Rowkey的范围扫描
- 全表扫描

写文件逻辑：

1. client获取数据写入的Region所在的RegionServer。
2. 请求写Hlog
3. 请求写MemStore

只有当写Hlog和写MemStore都成功了才算请求写入完成。MemStore后续会逐渐刷到HDFS中。

备注：Hlog存储在HDFS，当RegionServer出现异常，需要使用Hlog来恢复数据。

刷HDFS情况

1. 当占用一定比例内存时，刷HDFS。
2. 达到一定上限。
3. Hlog到达一定数量。
4. 手动flush。
5. 关闭regionserver时。
6. Region使用HLOG恢复完数据后。

MongoDB

- MongoDB的提供了一个面向文档存储，操作起来比较简单和容易。
- 你可以在MongoDB记录中设置任何属性的索引 (如：FirstName="Sameer",Address="8 Gandhi Road") 来实现更快的排序。 你可以通过本地或者网络创建数据镜像，这使得MongoDB有更强的扩展性。
- 如果负载的增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上这就是所谓的分片。
- Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
- MongoDb 使用update()命令可以实现替换完成的文档（数据）或者一些指定的数据字段 。
- Mongodb中的Map/reduce主要是用来对数据进行批量处理和聚合操作。 Map和Reduce。Map函数调用emit(key,value)遍历集合中所有的记录，将key与value传给Reduce函数进行处理。
- Map函数和Reduce函数是使用Javascript编写的，并可以通过db.runCommand或mapreduce命令来执行MapReduce操作。
- GridFS是MongoDB中的一个内置功能，可以用于存放大量小文件。
- MongoDB允许在服务端执行脚本，可以用Javascript编写某个函数，直接在服务端执行，也可以把函数的定义存储在服务端，下次直接调用即可。

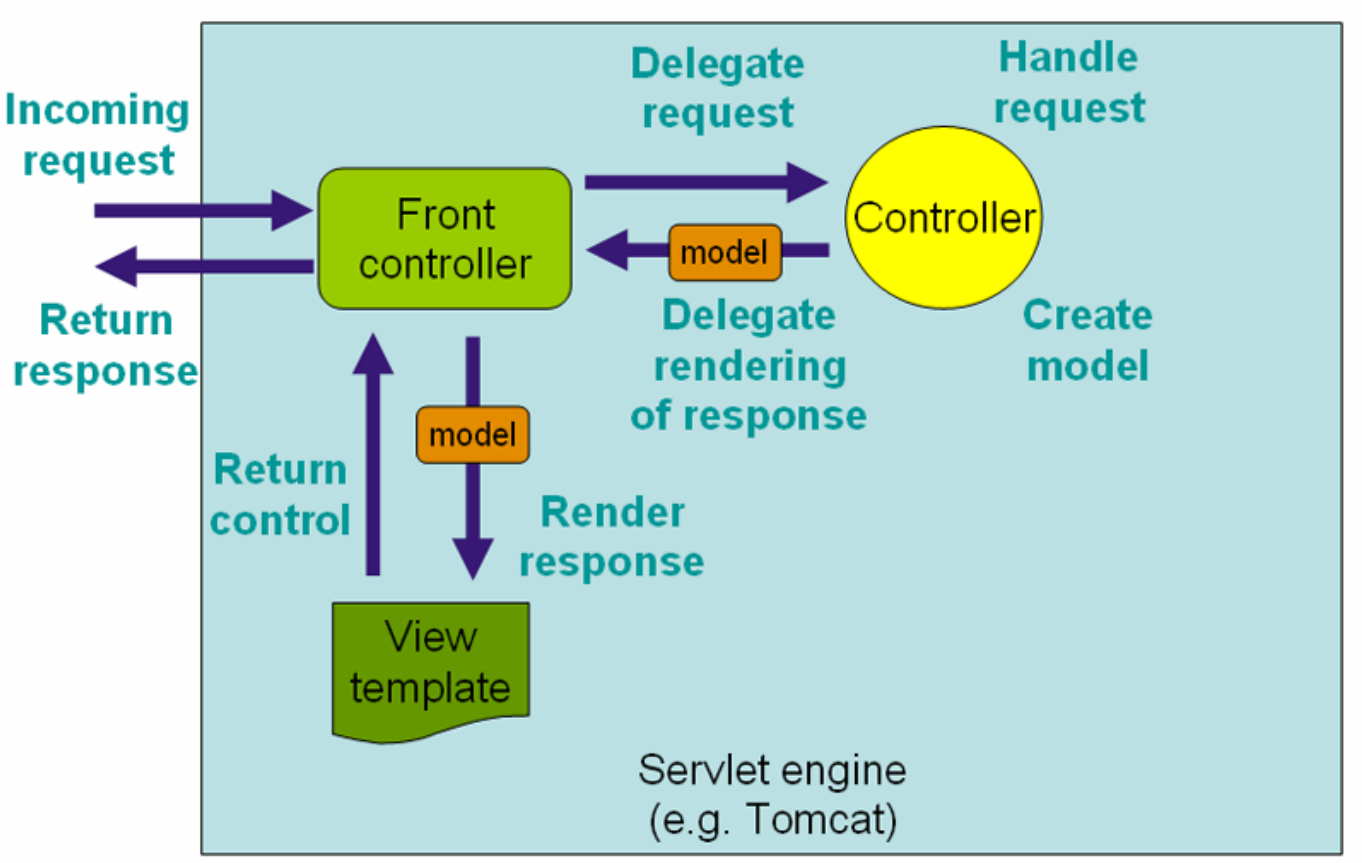
Redis

Redis是一个开源（BSD许可），内存存储的数据结构服务器，可用作数据库，高速缓存和消息队列代理。它支持字符串、哈希表、列表、集合、有序集合，位图，hyperloglogs等数据类型。内置复制、Lua脚本、LRU收回、事务以及不同级别磁盘持久化功能，同时通过Redis Sentinel提供高可用，通过Redis Cluster提供自动分区。

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
- 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
- 原子 – Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。
- 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存。在内存数据库方面的另一个优点是， 相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。 同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

SpringMVC



The requesting processing workflow in Spring Web MVC (high level)

mybatis和hibernate的区别

1. Hibernate功能强大，数据库无关性好，O/R映射能力强，如果你对Hibernate相当精通，而且对Hibernate进行了适当的封装，那么你的项目整个持久层代码会相当简单，需要写的代码很少，开发速度很快，非常爽。
2. Hibernate的缺点就是学习门槛不低，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用好Hibernate方面需要你的经验和能力都很强才行。
3. iBATIS入门简单，即学即用，提供了数据库查询的自动对象绑定功能，而且延续了很好的SQL使用经验，对于没有那么高的对象模型要求的项目来说，相当完美。
4. iBATIS的缺点就是框架还是比较简陋，功能尚有缺失，虽然简化了数据绑定代码，但是整个底层数据库查询实际还是要自己写的，工作量也比较大，而且不太容易适应快速数据库修改。

树的DFS和BFS

广度优先遍历(BFS):需要一个队列，先进先出。

深度优先遍历(DFS):需要一个栈，先进后出。

头条题目

1. 在数组中用空间复杂度为O(1)，时间复杂度为O(n)算法检查有无重复数字。（不能用加法）

https://blog.csdn.net/rabbitbug/article/details/1753147

5 2 1 3 4 4

1 2 3 4 5 -1 space : 4

-1 2 3 4 5 space : 1

2. epoll select poll详细区别

https://blog.csdn.net/chengzi_comm/article/details/51220163

3. 滑动窗口详细

https://www.zhihu.com/question/32255109

4. static在函数内部和外部的区别

作用域是不相同的!

5. 树的度问题

根据树的定义，在一颗树中，除树根结点外，每个结点有且仅有一个前驱结点，也就是说，每个结点与指向它的一个分支一一对应，所以除树根结点之外的结点树等于所有结点的分支数，即度数，从而可得树中的结点数等于所有结点的度数加1。总结点数为

$$1+n_1+2n_2+3n_3+\ldots+kn_k$$

而度为0的结点数就应为总结点数减去度不为0的结点数的总和，即

$$n_0=1+n_1+2n_2+3n_3+\ldots+kn_k-(n_1+n_2+n_3+\ldots+n_k)=1+\sum_{i=1}^{k}(i-1)n_i$$

微软题目

1. Google F1系统架构设计 （比较有意思，给一个条件，设计事务系统）
   https://www.cnblogs.com/foxmailed/p/4366692.html

   高扩展性（自动分片存储）；

   可用性和一致性（同步复制）；

   High commit latency: Can be hidden（分层架构、协议缓冲列类型、高效客户端代码）

还需要了解 spanner和megastore。

spanner

- 严格的CP系统以及远超5个9的可用性
- 基于2PC协议的内部顺序一致性
- 外部一致性，支持读写事务、只读事务、快照读
- 全球化的分布式存储系统，延迟是能够接受的
- 基于模式化的半关系表的数据模型

亮点是外部一致性。读事务的版本通过paxos实现。

F1也是纯内存的，在spanner上面一层。

https://segmentfault.com/a/1190000009707788（浅谈Spanner和F1）

1. 手写B+树。 写查询和插入。 （插入比较难）

https://www.cnblogs.com/wade-luffy/p/6292784.html

插入分三种情况。

删除操作要考虑两种情况。 一种是删除非索引值。 一种是删除了索引值，需要把另一个儿子

360题目

1. 链表逆序。
2. 判断一个数组是否为一个 排序二叉树的后续遍历。
3. 判断机器是大端还是小端
4. 手写memcpy