**C++语法**

# Slt：

## 容器分类

标准STL序列容器：vector、string、deque和list

标准STL关联容器：set、multiset、map和multimap

非标准序列容器slist和rope

Vector 数组 支持快速访问

List 双向链表 支持快速增删改查

（注：）经过实际计算，vector无论是push\_back还是insert10万个数据，都是list的三倍

Deque deque 的最大任务，便是在这些分段的定量连续空间上，维护其整体连续的假象，·并提供随机存取的接口。避开了“重新配置，复制，释放”的轮回

1、如果你需要高效的随即存取，而不在乎插入和删除的效率，使用 vector；  
2、如果你需要大量的插入和删除，而不关心随即存取，则应使用 list；

3、如果你需要随即存取，而且关心两端数据的插入和删除，则应使用deque。

Stack list/deque

Queue list/deque

## vector和list的区别

vector类似于数组，能高效的进行随机存取，时间复杂度为o(1);

list类似于链表，只能通过指针访问数据，所以list的随机存取非常没有效率，时间复杂度为o(n);

所以vector的迭代器可以+1，而list不可以.

总之，如果需要高效的随机存取，而不在乎插入和删除的效率，使用vector;  
 如果需要大量的插入和删除，而不关心随机存取，则应使用list。

Vector 的capacity保存的就是可以存放的元素个数

## vector的push\_back实现原理

vector是一段连续的内存空间。start，finish，end\_of\_storage三个指针描述了空间状态，这三个是普通的指针。start到finish是已经使用的内存，里面有元素。finish到end\_of\_storage是未使用的内存，里面没有元素。

vector有预存的内存（capacity），如果存入的元素大于了capacity，就重新分配一个比原来capacity大两倍的内存

## vector的reserve和resize;

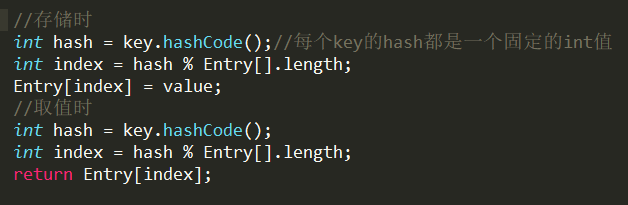
reserve是给vector预分配内存，但是不会给内存进行初始化

resize是改变vector内存，如果resize n个对象，如果当前size小大于n ,则只是保留前n个对象，如果size小于n，则容器大小改为n,并且构建n-size个对象

## hash\_map实现原理

底层用hash表存储。Hash\_map里面实现一个静态内部类Entry，其有key,value，next等属性。Hash\_map其实就是一个线性数组，Map的内容就保存在Entry[]里面。

## Hash\_map的存储实现



如果遇到冲突

如果冲突：比如，第一个键值A进来，通过计算key的hash得到index=0，记作Entry[0]=A,一会儿又进来一个键值对B，通过计算index也等于0。那么，B.next = A，Entry[0] = B，如果又进来类似情况C，则C.next = B,Entry[0]=C

所以，数组中存储的是最后插入的元素

## map和hash\_map的区别

STL的map底层是用红黑树实现的，查找时间复杂度是log(n)；

STL的hash\_map底层是用hash表存储的，查询时间复杂度是O(1)；

一般情况下，如果记录非常大，考虑hash\_map，查找效率会高很多，如果要考虑内存消耗，则要谨慎使用hash\_map。

## 哈希表时间复杂度如何?冲突了如何解?

构造哈希函数的原则是：①函数本身便于计算；②计算出来的地址分布均匀，即对任一关键字 k，f(k) 对应不同地址的概率相等，目的是尽可能减少冲突。

冲突解决方式

1.开放定址法

2.再哈希法

3.链地址法

4.建立公共溢出区

# 分析线性表、二叉平衡树和哈希表存储数据时各自的优劣。

链式：优点：插入和删除不需要移动，空间有效利用缺点：大量访问操作时不如顺序存储结构。

顺序：优点：可随机存取表中任一元素。缺点：插入或删除操作时，需大量移动元素。

二叉平衡数存储，查找O(lgn), 插入O(lgn);

优点在平衡树中按序查找方便， 缺点二叉平衡树的实现代价比较高。

哈希表存储，查找O(1),插入O(1)。    缺点：不能显示序的信息，不能找出最大最小值。

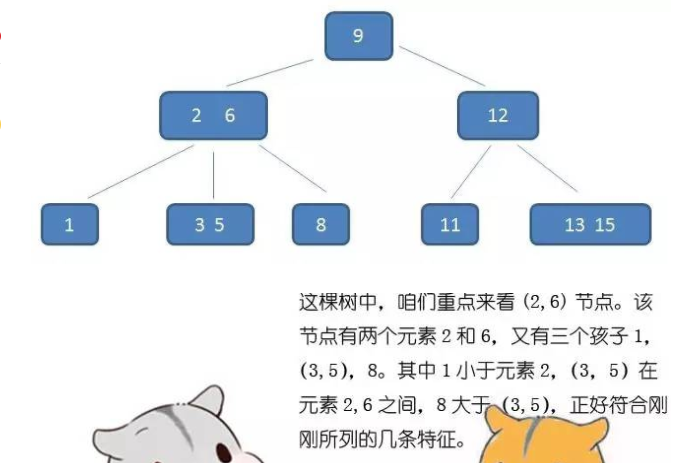
二叉平衡树O(lgn)比哈希表O(1)不会大太多，若储存的数有序的要求，用二叉平衡树比哈希要好

# B树、存储模型（B树、B+树与数据库引擎这块是热门问题）

B树是为了提高磁盘或外部存储设备查找效率而产生的一种多路平衡查找树。

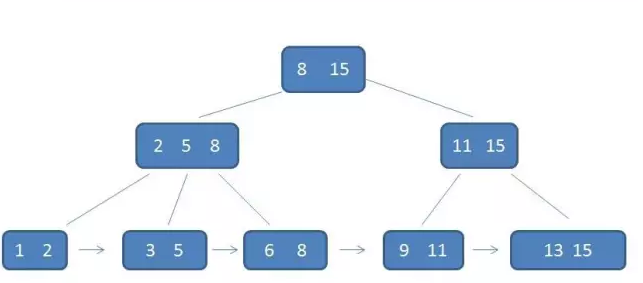
一个m阶的B树：

1. 根节点至少有两个子女
2. 每个中间节点都包含k-1个元素和k个孩子，其中m/2<=k<=m
3. 每个叶子节点都包含k-1个元素，其中m/2<=k<=m
4. 所有叶子节点都位于同一层
5. 每个节点中的元素从小到大排列，节点k-1个元素正好是k个孩子包含元素值域的划分



M阶B+树

1. 有k个子树的中间节点包含有k个元素（B树中是k-1个元素），每个元素不保存数据，只用来索引，所有数据都保存在叶子节点
2. 所有的叶子结点中包含了全部元素的信息，及指向含这些元素记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。
3. 所有的中间节点元素都同时存在于子节点，在子节点元素中是最大（或最小）元素



卫星数据：指索引元所指向的数据记录。

在数据库的聚集索引（Clustered Index）中，叶子节点直接包含卫星数据。在非聚集索引（NonClustered Index）中，叶子节点带有指向卫星数据的指针。

聚集索引：

表中行的物理顺序与键值（索引）顺序相同

非聚集索引：

表中行的物理顺序与键值（索引）顺序不同

# 指针数组，数组指针，指针函数，函数指针

指针数组：存放指针的数组int \*a[10]

数组指针：指向数组的指针 int (\*a)[10];

指针函数：返回值为指针的函数int\* a();

函数指针：指向函数的指针(int(\*f)(int a))

# 红黑树：

一种平衡检索树，并不是平衡二叉树，他放宽了平衡二叉树的某些要求，由于一定限度的不平衡，才得到了性能的提升

应用于stl的 set,mutiset,map,multimap等

平均高度log(N),最坏不会超过2log(N);

能以o(log2(N))的时间复杂度搜索，插入删除，任何不平衡都可以三次内解决

满足五个性质

1:节点是红色或者黑色

2:根节点是黑色  
 3 : 每个叶子节点（NULL或者空节点）是黑色

4：每个红色节点的两个子节点都是黑色的（不存在两个连续的红色节点）

5：从任意节点到其每个叶子结点的所有路径都包含相同数目的黑色节点

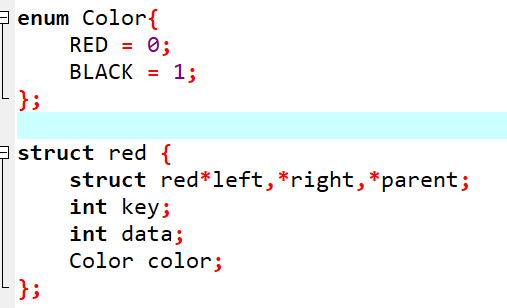
<http://blog.csdn.net/sun_tttt/article/details/65445754>

## 红黑树面试常考问题

（1）：stl的set底层用的什么数据结构

红黑树

（2）：红黑树的数据结构怎么定义的



（3）：红黑树有哪些性质

满足五个性质

（4）：红黑树的各种操作的时间复杂度

能保证在最坏情况下，基本的动态几何操作时间均为O(lgN)

（5）：红黑树相比BST和AVL树有什么有点

红黑树是牺牲了严格的高度平衡的优越条件为代价，它只是要求部分的达到平衡要求，降低了对旋转的要求，从而提高了性能，能以log2N的时间复杂度进行搜索，插入，删除操作，此外，由于他的设计，任何不平衡都可以在三次旋转内解决

相比于BST，因为红黑树可以确保树的最长路径不会超过两倍最短路径的长度，所以查找效果是有最低保证的，最坏情况也可以保证2logN。而BST最坏情况可达N

相比AVL，时间复杂度相同，但是统计性能比AVL树高，插入等操作节约时间

（6）：红黑树相比于哈希表，在选择时有什么依据

权衡三个因素，查找速度，数据量，内存使用，可扩展性

Hash查找速度更快，但是消耗内存，当数据特别多，构造速度也变慢

# C++有了malloc/free，为什么还要new/delete

1:malloc/free是标准库函数，而new/delete是运算符

2.对于非内部数据类型的对象而言，光是malloc/free无法满足动态对象的要求

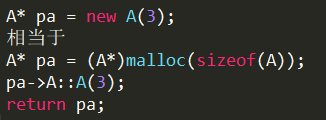
对象在创建的同时，自动执行构造函数，对象在消亡之前自动执行析构函数

由于malloc/free是库函数，而不是运算符，不在编译器控制权限之内，不能把执行构造函数的任务强加于malloc/free

3:new/delete是运算符，不是库函数

New的过程：获得一块内存空间，调用构造函数，返回一个正确的指针，如果创建的是内部数据类型变量，则没有第二步

而malloc没有调用构造函数这一步。所以，初始化得单独调用



解释：

内部数据对象，如int ,double

非内部数据对象：自定义的对象

库函数为什么不在编译器控制权限之内：库函数是已经编译的代码，编译器不会编译检查，由连接器直接同用户写的代码合成exe文件.

# Malloc后free报错，原因：

1：释放一个空指针。

2：重复释放。

3：释放一个非自己申请的内存

4：申请的内存块写过界了，此时内存块就被破坏，释放时为了避免释放了其他有用的数据，就会报错

# ****C++ public/private/protected继承****

类的一个特征就是封装--->public，private

用户代码（类外），可以访问public成员，不能访问private成员，private成员只能由类成员（类内）和友元访问

类的一个特征就是继承--->protected

Protected成员可以被派生类对象访问，不能被用户代码（类外）访问



# ****C++如果构造函数定义为private/protected。有什么方法可以调用？****

如果是protected ，只需在用它派生public一个派生类。派生类也可以实例化，因为，对于protected函数，子类的函数可以调用

如果是private，可以定义一个public static函数，这个函数内部可以实例化该类，这样，这个函数就的调用就不需要提前实例化一个对象，而可以直接被调用，而它又属于类的内部函数，也就可以调用构造函数了

# **C++友元函数：**

为该类提供一个外部访问的接口

这个友元函数不属于该类的成员函数，他是定义在类外的普通函数，只是这个函数可以访问类的成员变量

1）必须在类的说明中声明友元函数，说明时以关键字friend开头，后跟友元函数的函数原型，友元函数的声明可以出现在类的任何地方，包括在private和public部分；  
2）注意友元函数不是[类的成员函数](https://www.baidu.com/s?wd=%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9uAw-rHwbm1-WryRznhwb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1fLnHDvnHfd)，所以友元函数的实现和普通函数一样，在实现时不用"::"指示属于哪个类，只有成员函数才使用"::"作用域符号；  
3）友元函数可以访问对象的私有成员，但普通函数不行；4）调用友元函数时，在实际参数中需要指出要访问的对象，  
5）类与类之间的友元关系不能继承。  
6）一个[类的成员函数](https://www.baidu.com/s?wd=%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9uAw-rHwbm1-WryRznhwb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1fLnHDvnHfd)也可以作为另一个类的友元，但必须先定义这个类。

**Static成员变量：**

**1:存在于全局作用域的静态变量**

**2：存在于函数中的静态变量**

**3：存在于类的成员变量中的静态变量**

**4：存在于类的成员函数中的静态变量**

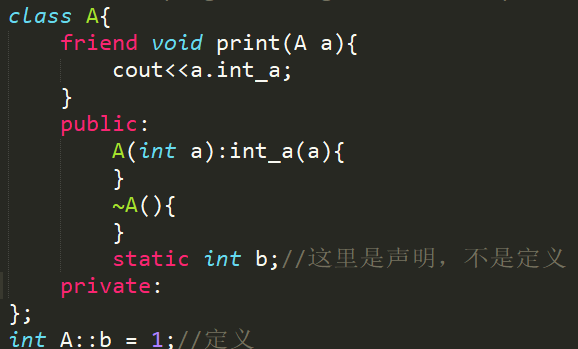
**5：存在于命名空间的静态变量**

**Static函数**

**1：存在于全局作用域的静态成员函数**

**2：存在于类中的静态函数**

**只能访问静态成员变量，对象或类都可以直接调用静态成员函数**



**Const成员变量初始化只能用初始化成员列表初始化**

**原因：在定义一个对象时，会先创建一个对象，再进入函数体，也就是说，会先定义const成员变量，然后进入构造函数，此时也就无法再对const赋值了**

# **内联函数和宏的区别**

内联函数在编译时展开，而宏在预编译时展开

在编译时，内联函数可以直接镶嵌在目标代码中，而宏只是简单的文本替换

内联函数可以完成类型检测，语句是否能正确编译等功能，宏不具备这种功能

内联函数是函数。而宏不是函数

内联函数传参不会出现二义性，而宏如果括号没写好，容易出现错误

# **指针和引用的区别**

初始化要求不同，引用要在创建的同时初始化，而指针可以暂时不用初始化

可修改性不同，引用一旦初始化指向一个对象，他就不能被改变成另一个对象的引用，而指针可以

不存在NULL引用，引用不能指向空值

测试需要的区别，引用不用检测是否为NULL,而指针需要检测----所以传引用比传指针安全

# **Main函数执行前会执行什么代码**

全局对象的构造函数（等）。。。

# 复制构造函数（拷贝构造函数）是什么。什么是深复制和浅复制

复制构造函数是基于同一类的其他对象的构建及初始化

C++下，一般三种情况对象需要复制构造函数

一个对象以传值的方式进入函数体

一个对象以传值的方式从函数体返回

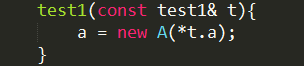
一个对象需要通过另一个对象进行初始化

浅复制：（被拷贝对象A，拷贝对象B）

只被复制函数一点不改变被复制，如果A的某个成员变量是指针，指向某个区域，浅拷贝后，如果B被释放，那么指向的这个区域也会被释放，对A来说，就会产生错误

深拷贝：

针对浅拷贝的问题，深拷贝对拷贝构造函数进行重写



# **复制构造函数和赋值函数的区别**

1. **复制构造函数是一个对象来初始化一块内存区域，这块内存就是新对象的内存区，而赋值函数是对一个已经被初始化的对象进行operator=操作**
2. **。。。**

**C++不支持哪些函数为虚函数**

**构造函数，内联函数，友元函数，静态函数（原因。。。）**

# Struct 和class 的区别。

**C++中两者的唯一区别在于：Class定义中默认情况下的成员都是私有的，而结构Struct定义中默认情况下的成员都是公有的。**

**在C中，结构中不能有成员函数，而在C++中可以有成员函数.**

# 函数调用过程

在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。

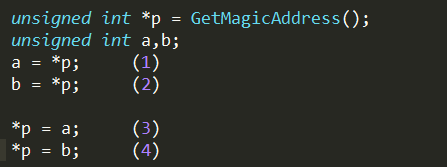
# malloc之后再进行free，free的内存空间一定被OS回收了吗？

os管理是对页进行分配和回收，如果free的内存只是某个页的某小块内存，而这个页的其他内存依旧被使用，这时os就不会回收内存空间。

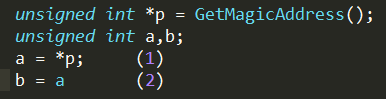
# Volatile起源

C++的volatile有三个特性：易变性，不可优化性，顺序性

最早出现于19世纪70年代，被用来处理memory-mapeed i/o带来的问题。在引入MMIO后，一块内存地址，既有可能是真正的内存，也可能是被映射到一个I/O端口，相对的。读写一个内存地址，既有可能是操作内存，也有可能是读写一个I/O设备。



如上代码，如果该指针指向的是一个I/O设备。那么（1），（2）中的a,b就会收到I/O设备的连续两个unsigned int类型的值。如果指向的是一个地址，那么，编译器可能做出如下优化

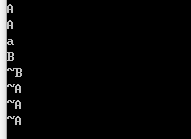
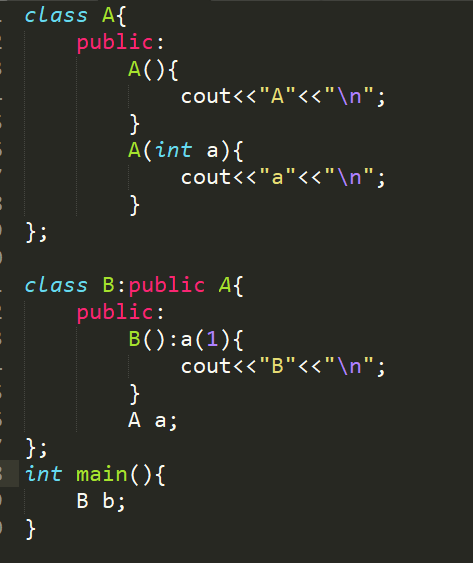


因此，对于I/O设备需要防止编译器做这个优化，不能假设 指针b指向的内容不变----（易变性）

注：

MMIO：MMIO就是通过将外围设备映射到内存空间，便于CPU的访问

类的成员变量，父类构造，



# 什么是面向对象技术：

面向对象技术是一种以对象为基础，以事件或者消息来驱动对象执行处理的程序设计技术。它具有抽象性，封装下，继承性以及多态性

面向对象与面向过程有什么不同

（1）：面向过程是采用函数或者过程来描述对数据的操作，但又将函数与其数据分离开。面向对象将数据与对数据的操作封装在一起，作为一个整体来处理，函数与数据分离。

（2）：面向过程程序设计方法以功能为中心来设计功能模块，难以维护，而面向对象以数据为中心来描述系统，数据相对于功能而言具有较强的稳定性，因此更加易于维护。

（3）：面向过程程序设计的控制流程由程序中预定顺序来决定，面向对象由运行时的各种

**操作系统**

# **进程至少具备的**

1. **task\_struct （身份）**
2. **可执行代码**
3. **独立的内存空间**
4. **独立的堆栈**

**copy on write是基于page的**

**fork（）**

**1：找到空闲的进程号**

**2：从父进程拷贝进程信息**

**Fork是系统调用，通过中断0x86,由eax记录系统调用号，进入内核，找到中断处理程序，再找到系统调用程序sys\_fork() --> do\_fork();**

**Fork,vfork,clone都是do\_fork()实现。**

**Do\_fork功能**

**1：定义pcb指针**

**2：为进程分配pid**

**3：调用p = copy\_process();(拷贝进程)**

**3.1为子进程分配pcb**

**一般分配8k或者4k,其中低地址1k保存task\_struct，其他作为堆栈**

# 线程

## 线程有几种状态

**新生状态，就绪状态，可运行状态，被阻塞状态，死亡状态**

## **线程怎样会阻塞**

**1：调用sleep（）**

**2等待I/O（等待阻塞）**

**3:试图得到一个锁，但这个锁被别的线程持有（同步阻塞）**

**4等待某个触发事件**

## **同步与互斥**

**进程互斥：**

**是进程之间的一种制约关系，当一个进程进入临界区使用临界资源时，另一个进程必须等待，只有当使用临界资源的进程退出临界区后，这个进程才解除阻塞状态。**

## **如何实现互斥**

**信号量机制**

**Wait(mutex);**

**Signal(mutex);**

**管程机制**

**把所有进程对某一种资源的同步操作都集中起来，构成一个所谓的秘书进程，凡是要访问该临界资源的进程，都需要先报告秘书。**

**秘书管道由四部分组成：**

1. **管道内部的共享变量**
2. **管道内部的条件变量**
3. **管道内部并行执行的进程**
4. **对于局部域管道内部的共享数据设置初始值的语句**

## **进程同步：**

**把异步环境下的一组并发进程，因直接制约而互相发送消息，进行互相合作，互相等待，使得进程按照一定的速度执行的过程**

## **进程同步需要遵循的规则**

**空闲让进**

**忙则等待**

**让权等待：当进程不能进入自己的临界区，应该立即释放处理机**

**有限等待**

## **进程间制约关系**

**资源共享关系（间接制约关系）**

**相互合作关系（直接制约关系）**

## **死锁定义：**

**如果一组进程每个进程都在等待仅由改组进程的其他进程才能引发的事件，那么这组进程是死锁的**

## **产生死锁的必要条件：**

**互斥条件**

**请求和保持条件**

**不可剥夺条件**

**环路等待条件**

## **处理死锁的方法**

**预防死锁：破坏四个必要条件**

**避免死锁：银行家算法—得出安全序列**

**检测死锁：利用资源分配图**

**解除死锁：终止进程。。。**

## **进程线程同步互斥的控制方法**

**1：临界区**

**适合一个进程内多个线程访问公共区域时使用**

**2：互斥量**

**适合不同进程访问公共区域使用**

**3：信号量**

**可以实现多个线程同时访问数据区域，原理与PV类似，先设置一个访问公共区域的最大线程连接数目，每有一个访问，数目减一**

**4：事件**

**适合线程间触发事件实现同步互斥**

## **信号量：**

**是一个记录型的数据结构，其中包含了一个信号量值和一个等待队列。其**

**中信号量值只能被p（wait）v（signal）操作**

## **进程间几种通信方式**

**1：管道：**

**无名管道局限于单向通信方式，且只能在创建它的进程和其子进程之间实现管道的共享**

**有名管道可以给任意关系的进程使用，但是由于长期存储在系统中，使用不当容易出错**

**2：消息：**

**本地套接字**

**网络通信**

3：共享内存（mmap内存映射）

# 互斥锁，自旋锁和读写锁

锁是一种为了解决某种临界资源互斥而使用的一种机制。

自旋锁和互斥锁功能差不多，每一时刻只能有一个执行单元占有锁，而占有锁的单元才能获得访问临界资源的所有权，从而达到互斥的目的

自旋锁和互斥锁的区别：

自旋锁在请求锁时，如果发现锁被占用，则不停的循环判断锁的状态，直到锁被释放，期间并不阻塞自己，它会一直占有cpu

互斥锁在等待锁被释放时，会把自己阻塞并放入队列种。当锁被释放，会唤醒它，阻塞会耗时，但是，不会消耗CPU

**读写锁实际是一种特殊的自旋锁。把访问者分为读者和写者，允许多个读者，或者一个写着**

# ****必须使用类的初始化成员列表初始化的三种情况****

如果成员变量也是一个类，且这个类没有无参数构造函数

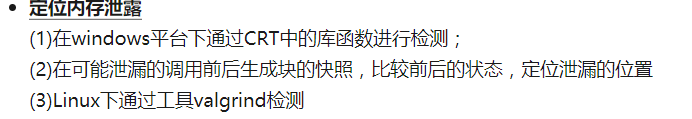
成员变量时const修饰或者是引用。这两个都需要在创建的同时初始化，后面无法赋值

子类初始化父类的静态成员变量

使用初始化成员列表效率高

不使用，成员变量先初始化，在赋值。效率低

# ****定位内存泄露****



# 多进程和多线程的选择

**使用多线程：**

**需要更可靠**

**使用多线程：（共用存储区，通信方便，节约储存，不安全）**

**需要频繁创建销毁**

**需要大量计算**

# 动态连接库的两种方式

**第一种是静态链接方式，在这种方式下，**[**动态链接库**](https://www.baidu.com/s?wd=动态链接库&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBuWNhnhD4mhD4nH6suWmd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1c4nWTLPHn4rjn3PHTzrjfd)**中的所有数据代码都将拷贝到调用程序的**[**代码空间**](https://www.baidu.com/s?wd=代码空间&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBuWNhnhD4mhD4nH6suWmd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1c4nWTLPHn4rjn3PHTzrjfd)**中去，此时它和调用程序本身的函数没有什么区别；**

**第二种是动态链接方式，在这种方式下，**[**动态链接库**](https://www.baidu.com/s?wd=动态链接库&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBuWNhnhD4mhD4nH6suWmd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1c4nWTLPHn4rjn3PHTzrjfd)**中的数据代码是在需要的时候才拷贝到内存中去的；**

# windows内存管理方式及其优缺点

**页式存储**

**优点：没有外碎片，每个碎片不超过业的大小**

**缺点：程序全部装入内存，要求有相应的硬件支持。**

**段式存储**

**优点：可以分别编写和编译，可以针对不同类型的段采用不同的保护，**

**可以按照段为单位进行共享。（信息共享，动态链接，方便编写，信息保护）**

**缺点：会产生碎片**

**段页式存储：**

**具有两种优点**

# **Linux伪文件**

**Linux系统的文件类型大致可分为三类：普通文件，目录文件，和伪文件**

**伪文件不是用来存储数据的，因为这些文件不占用磁盘空间，尽管这些文件确实存在于目录树中，是目录树的一部分。并且也按照目录的方式来组织。**

**伪文件的目的是提供一种服务，通过这种服务，系统可以像操作普通文件一样来访问这些文件代表的对象。常见的伪文件是proc文件，socket文件，管道**

# 缓冲区溢出

在计算机内部，输入数据通常被放在一个临时空间内，这个临时空间就是缓冲区，缓冲区的长度事先已经被程序或者操作系统确定好了。向缓冲区填充数据，如果数据长度很长，超过了缓冲区本身的容量，那么数据就会溢出存储空间。而这些溢出的数据还会覆盖在合法的数据上

# 守护进程：

守护进程是一种生存期很长的进程，它们运行在后台且独立于控制终端周期性的执行某种任务或者等待处理某些发生的事件。

一般的网络服务都是守护进程。

# 五种I/O模型

阻塞I/O

进程发送请求，等待内核读写成功，如果没有读写则阻塞

非阻塞I/O

进程发送请求，如果没有读写直接返回一个错误。继续执行其他事

多路复用I/O

进程可以同时检测多个文件描述符

异步I/O

进程发送请求，内核等待读写，内核缓冲区有数据并复制到用户态，则通知进程

信号机制

和异步I/O很相似，只是当内核缓冲区有数据，就通知进程来复制数据

到用户态

阻塞，非阻塞，同步，异步

一个IO操作一般分为两部分，发送IO请求和实际的IO操作

阻塞IO和非阻塞IO的区别在第一步，发起IO是否被阻塞，如果阻塞直到完成，那么就是传统的阻塞IO，如果不阻塞，那么就是非阻塞IO

同步异步的区别在第二步，实际IO操作是否阻塞，如果实际操作IO时，阻塞请求进程，那么就是同步，如果不阻塞，就是异步。

# 周转时间，平均周转时间，带权周转时间

周转时间 = 作业完成时间-作业提交时间

注：作业提交时间不是作业进入内存的时间，而是发出请求，提交就开始计时

平均周转时间 = （作业1周转时间+…+作业n周转时间）/n

带权周转时间 = 作业周转时间/作业实际运行时间

平均带权周转时间 = （。。。）/n

**Linux**

# kill用法，某个进程杀不掉的原因

kill -9 能杀死任何进程也杀不了

1：进程已经成为僵尸进程，当他的父进程将他回收，或者直接kill父进程

2：进程处于内核态中，当进程进入内核态，会屏蔽所有信号

# 处理僵尸进程的三种方法

1：父进程回收法

Wait函数将使其调用者阻塞，直到某个子进程终止

2：使用signal信号函数

当子进程死亡，发送信号给父进程，父进程调用Wait,回收

2：init进程回收法

1. 如果父进程先于子进程结束，那么子进程的父进程就会变成init进程
2. 如果init的子进程结束，则init进程会自动回收其子进程的资源，而不是让他变成僵尸

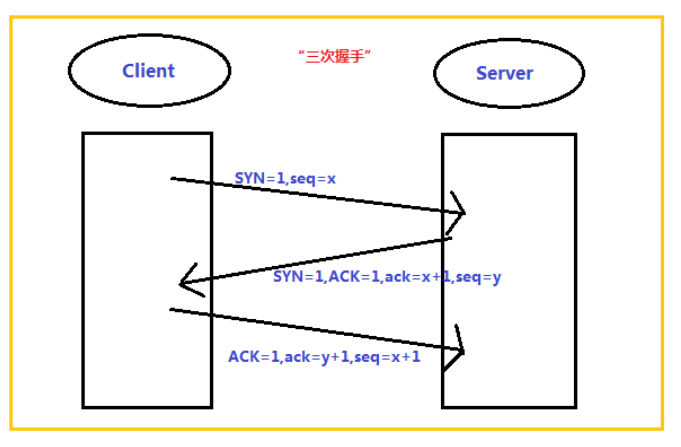
Df命令：

查看磁盘使用信息

磁盘总量 = 已使用的+未使用的+不可使用的

**计算机网络**

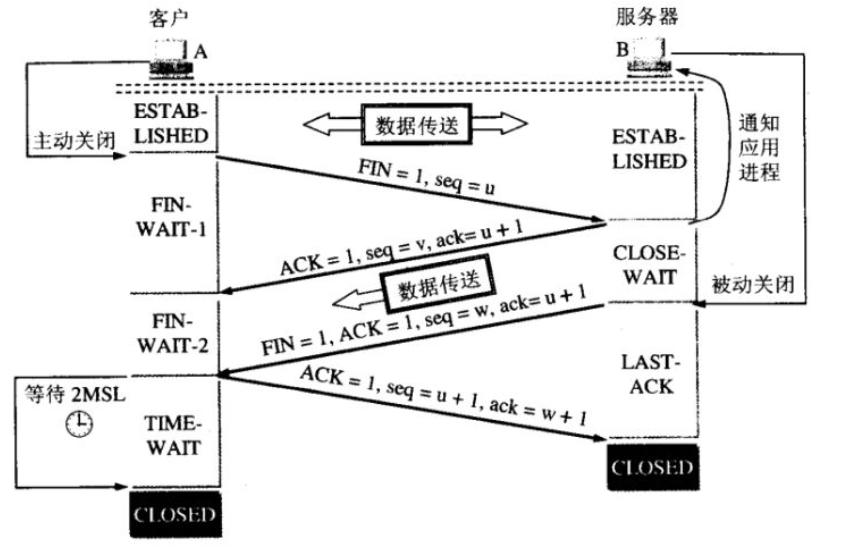
# **Tcp/ip三次握手**



# **TCP/IP为什么要三次握手，而不是两次或者四次.设A为客户端B为服务器**

**为了解决“网络中存在的延迟的重复分组”。A发出的第一个请求报文段并没有丢失，而是在某个网络站点长时间滞留了，A请求超时后又重写发出了请求，而先建立了连接，而此时之前滞留的分组也到达B，原本这个分组已经失效，但是，B却认为又是一次连接，。如果采用二次握手，这次连接又会被建立，资源会被白白浪费。采用三次握手，B又发出一个请求给A，此时A不会给B回复，自然无法建立第二个连接了**

# **Tcp/ip四次挥手**



# 为什么A要先进入TIME-WAIT状态，等待2ms才进入CLOSED状态

a.**为了保证B能收到A的确认应答**

**如果A发完确认应答后直接进入CLOSED状态，如果这个分组丢失，B等待超时后就会重新发送连接释放亲求，但此时A已经关闭了，不会做出任何响应因此，B永远无法正确关闭**

**b.**

**如果没有这个，分组丢失后,B会多次发送FIN，如果A再次发送建立连接的请求，则会接受到这个FIN包**

# TCP如何保证可靠传输 C:\Users\wangh\AppData\Roaming\Tencent\Users\1603635122\QQ\WinTemp\RichOle\%E[L1~(V7S9E8PJOZK9Q%NM.png

12:ICMP是什么协议,处于哪一层?

Internet控制报文协议，处于网络层（IP层）

15:IP组播有那些好处?

Internet上产生的许多新的应用，特别是高带宽的多媒体应用，带来了带宽的急剧  
  
消耗和网络拥挤问题。组播是一种允许一个或多个发送者（组播源）发送单一的数据包  
  
到多个接收者（一次的，同时的）的网络技术。组播可以大大的节省网络带宽，因为无  
  
论有多少个目标地址，在整个网络的任何一条链路上只传送单一的数据包。所以说组播  
  
技术的核心就是针对如何节约网络资源的前提下保证服务质量。

16:.列出3个常用网络协议使用的端口。  
 HTTP协议用80端口，FTP协议用20/21端口，POP3协议用110端口

17:**HTTP返回码中301与302的区别**

**18:**．请问交换机和路由器各自的实现原理是什么**?**分别在哪个层次上面实现的**?**

交换机属于ＯＳＩ第二层即数据链路层设备。它根据ＭＡＣ地址寻址，通过站表选择路由，站表的建立和维护由交换机自动进行。路由器属于ＯＳＩ第三层即网络层设备，它根据ＩＰ地址进行寻址，通过路由表路由协议产生。交换机最大的好处是快速，路由器最大的好处是控制能力强。

# Http1.0和http1.1区别

1：http1.1支持长连接和请求流水线处理，http1.0默认短链接，客户端发起请求后建，

立连接，服务器完成处理请求后立即断开TCP连接，服务器不跟踪每个客户

请求流水线处理：在一个TCP连接上可以传输多个HTTP请求和响应，减少建立

和关闭连接的消耗和延迟---例如，一个包含许多页面文件的多个请求可以在一个

连接中传输，但每个单独的页面文件的请求和应答仍然需要使用各自的连接

Http1.1还支持客户端不用等待上一次请求结果返回就发送下次请求，服务器按照客户端请求的先后次序返回应答结果

2：http1.1增加host字段，支持一个IP多台主机

3：增加状态码100（continue），一般用于客户端要发送大量数据，则先发一个请求，如果服务器返回100，则允许客户端发送数据

4：http1.1可以支持分块传输数据，避免消息过载

Http和https协议

Http协议不足

1：通信使用明文，内容可能被窃听

2：不验证通信方的身份，所以有可能遭遇伪装

3：无法证明报文的完整性，所以有可能已遭篡改

HTTP+加密+认证+完整性保护 = HTTPS

Session和cookie区别

1：存储位置不同

Session存储在服务器，cookie存储在客户端,所以session安全性更高

2: session保存的是对象,cookie保存的是字符串

3：生命周期不同：session默认直到关闭浏览器，cookie可以一直保存

4：应用：

Cookie判断用户是否登陆过网站。记录购物车信息

# MVC/MVP/MVVP

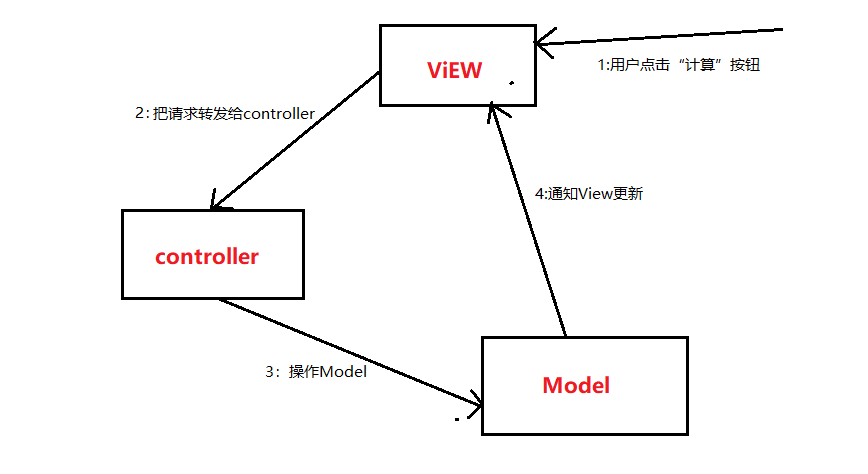
**MVC**

**M:Model,专门用来处理业务逻辑，不干别的事。比如在一个 薪水计算系统，用来**

**计算一个人的薪水等。。。**

**V：View,专注于页面布局和数据显示**

**Controller:翻译用户的输入 操作模型和视图**

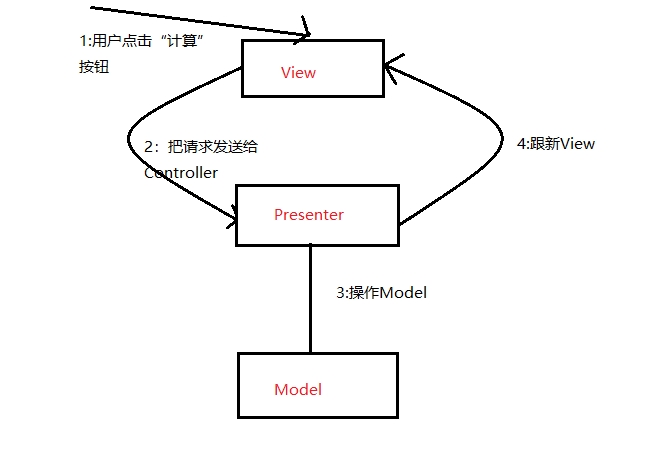
****

**可以看出View依赖Controller和Model**

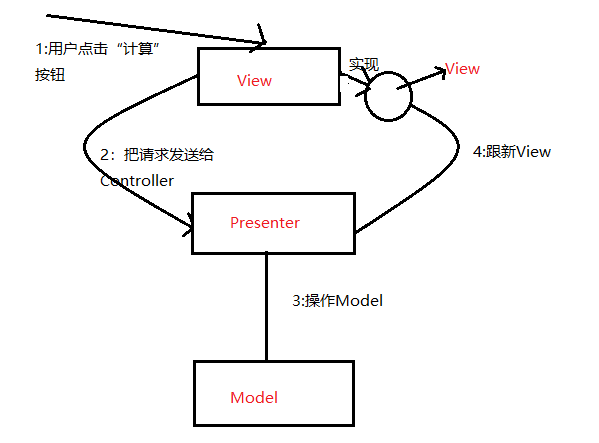
**Controller依赖View和Model**

**耦合还是很紧密，然后设计者改变Controller，把Model和View完全隔开**

**MVP：**

****

**MVVP**

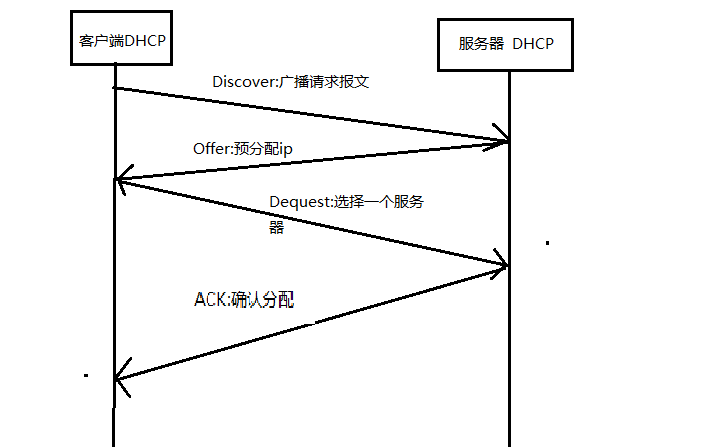
****

**对称加密和非对称加密**

**对称加密指加密和解密公用一个密钥**

**非对称加密指加密和解密不用一个密钥，一般有一个公开密钥和一个私有密钥**

DHCP如何实现分配ip的



常用协议以及对应端口

HTTP：80

HTTPS：443

Telent:23

FTP: 21/21

DNS: 53

Http状态码

200 OK 服务器已经成功处理了请求并提供了请求的页面

202 Accepted 已经接收请求，处理未完成

301 永久重定向

302 临时重定向

304 自从上次请求后，请求内容未修改

404 Not found

500 internet server error 服务器遇到一个错误，使其无法为请求提供服务

打开网页到页面显示之间的过程

输入网址

域名解析

与服务器TCP三次握手建立连接

浏览器向服务器发送HTTP请求

Web处理数据。并返回指定URL的数据

浏览器下载数据后，解析源文件，显示基础页面

分析超链接页面，重复上述过程，完成全部显示

DNS解析过程

**.。**

I/O多路复用系统调用（select,poll,epoll）

I/O多路复用的本质是通过一种机制（系统内核缓冲数据），让单个进程可以监视多个文件描述符，一旦某个描述符就绪（一般是读或者写就绪），就能通知程序进行相应的读写操作。

网络层，传输层，应用层协议

网络层：ip,icmp,arp,igmp

传输层：tcp,ip

应用层：ftp,http,https,telnet,dns

各层硬件

物理层：中继器，集线器

数据链路层：网桥，交换机

网络层：路由器，防火墙

传输层：

# select/poll/epoll

select

设置一个最大数目，比如有100个文件描述符，扫描一遍，查看是否有等待事件发生

每次调用select都要把fd集合从用户态拷贝到内核态，开销大

支持文件描述符数量小

Poll

不用设置最大数目，且文件描述符不同

Select/poll都需要自己不断查找所有fd集合，直到设备就绪，

Epoll（底层红黑树和链表）

只是在注册新事件才把fd拷贝到内核，不用像他们每次调用函数都拷贝

添加事件后，与设备驱动程序建立回掉关系，当事件发生，会调用回掉方法，会将发生的事件fd添加到双链表

只是访问就绪链表

epoll\_create

epoll\_ctl

epoll\_wait

MD5：

Message Digest Algorithm – 信息摘要算法，用来提供消息的完整性保护，确保信息传递的一致

特点：

1：压缩性，任意长度的数据，算出的MD5的值的长度固定

2：容易计算

3：抗修改性：对原数据的任何改动，哪怕是一个字节，所得到的MD5的值都有很

大的区别

4：强抗碰撞：已知原数据和其MD5，想找一个具有相同MD5值得数据（即伪造）

数据非常困难

作用：

让大容量得信息在用数字签名软件签署私人密钥前被“压缩”成一种保密得格式（就

是把任意长度的字节串换成一定的十六进制数字串）

数据帧，ip报文，tcp段

Mtu 最大报文

Mss 最大段

Mtu = mss+tcp头+ip头

# 算法

为什么快排快？？？

给一个场景，大量不重复的数据，设计一个通用排序算法

使用bitmap，将每个数据对应到bitmap的每一个位，然后遍历这个bitmap

# 数据库

## 什么是事务

指作为单个逻辑单元执行的一系列操作，这些操作要么都执行，要么都不执行

### 事务属性

原子性

事务必须是原子操作，要么全部执行，要么都不执行

一致性

事务从一个一致性状态转移到另一个一致性状态

隔离性

对于并发操作数据库的操作，相互之间不影响

持久性

对于已经执行的操作，数据修改是永久的

## 存储过程

指经过预编译的存储在数据库的可多次被调用的一系列sql代码集合

优缺点

优点：

1）存储过程经过预编译，执行效率高。而普通sq语句执行一次就编译一次。

2）代码直接放在了数据库，通过存储过程名直接调用，减少网络通信

2）安全性高，可以指定特定用户才能使用这个存储过程

4）可以重复使用，可减少开发者工作量

## 触发器

是一种特殊的存储过程，主要靠事件来触发。当对数据表进行，插入删除更新等操作时，

被触发而执行，主要来保证数据的完整性约束条件

主要作用

能够实现由主键，外键等所不能保证的复杂的参照完整性和数据一致性

其他作用

强化约束：

跟踪变化

级联运行

存储过程的调用

索引

数据库中对一列或者多列进行排序的一种结构

视图：

从一个或者多个基本表或者视图中导出的表，是一个虚表

事务四种隔离级别出现原因

1：更新丢失：

两个事务都同时更新一行数据，一个事务对数据的更新把另一个事务对数据的更新覆盖了，这是因为系统没有执行任何的锁操作，因此并发事务并没有被隔离开来

2：脏读

一个事务读取到另一个事务未提交的数据操作结果

3：不可重复读

一个事务对同一行数据重复读取两次但是却得到了不同的结果

（1）：虚读---事务T1读取某一数据后，事务T2对其做了修改，当事务T1再次

读取数据时与前一次读取值不同

（2）：幻读---事务在操作过程中进行了两次查询，第二次查询的结果包含了第一

次查询中未出现的数据或者缺少了第一次查询中出现的数据，---这是因为两次查询过程中有另一个事务插入数据造成的

事务的四种隔离级别

由高到低：Read uncommitted,Read committed,Repeatable read,Serializable

Read uncommitted：读未提交，就是一个事务可以读取另一个未提交事务的数据

事例：老板要给程序员发工资，程序员的工资是3.6万/月。但是发工资时老板不小心按错了数字，按成3.9万/月，该钱已经打到程序员的户口，但是事务还没有提交，就在这时，程序员去查看自己这个月的工资，发现比往常多了3千元，以为涨工资了非常高兴。但是老板及时发现了不对，马上回滚差点就提交了的事务，将数字改成3.6万再提交。

Read committed：一个事务要等待另一个事务提交后才能读取数据

设计模式：

1：策略模式

2：观察者模式