你们需要的C++面试题来了

程序猿编码 今天

说一下C++和C的区别

参考回答：

设计思想上：

C++是面向对象的语言，而C是面向过程的结构化编程语言

语法上：

C++具有封装、继承和多态三种特性

C++相比C，增加多许多类型安全的功能，比如强制类型转换、

C++支持范式编程，比如模板类、函数模板等

局部变量和全局变量的区别？

参考回答：

局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内部，离开该函数后再使用这种变量是非法的。

局部变量从存储方式上可分为动态(auto)存储类型和静态(static)存储类型。

动态存储类型的局部变量都是动态的分配存储空间，数据存储在动态存储区（栈）中。函数调用结束后自动释放，生存期是在声明该变量的函数执行过程。

静态存储类型的局部变量则是静态的分配存储空间，数据存储在静态存储区中。在程序整个运行期间都不释放，生存期贯穿于程序运行的整个过程。

函数中的局部变量，如不专门声明为static存储类别，默认都是动态地分配存储空间的，我们在平时的声明变量的过程中auto都是默认省略的。

全局变量

全局变量也称为外部变量，是在函数的外部定义的，它的作用域为从变量定义处开始，到本程序文件的末尾。全局变量全部存放在静态存储区，在程序开始执行时给全局变量分配存储区，程序行完毕就释放。在程序执行过程中它们占据固定的存储单元，而不动态地进行分配和释放；

如果外部变量不在文件的开头定义，其有效作用域只限于定义处到文件终。

如果在定义点之前的函数想引用该外部变量，则应该在引用之前用关键字extern对该变量作“外部变量声明”。表示该变量是一个已经定义的外部变量。有了此声明，就可以从“声明”处起，合法地使用该外部变量。其有效作用域就被拓展到从这个文件extern声明处到文件结束。

如果在全局变量声明的时候，前面加上关键字static，那么其他文件就不能再访问和使用该变量，其有效作用域只限于定义处到文件终。

C++的三大特性？

封装、继承、多态。

封装

也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。简单的说，一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。

继承

是指可以让某个类型的对象获得另一个类型的对象的属性的方法。它支持按级分类的概念。继承是指

这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。通过继承创建的新类称为“子类”或“派生类”，被继承的类称为“基类”、“父类”或“超类”。

多态

分为静态多态和动态多态：

> 静态多态

编译器在编译期间完成的，编译器会根据实参类型来推断该调用哪个函数，如果有对应的函数，就调用，没有则在编译时报错。

> 动态多态

是指在程序运行时才能确定函数和实现的链接，此时才能确定调用哪个函数，父类指针或者引用能够指向子类对象，调用子类的函数，所以在编译时是无法确定调用哪个函数，在使用的时候基类中必须有虚函数，在派生类中必须重写虚函数。

使用struct关键字和class关键字定义类有啥区别？

参考回答：

strcut和class主要的区别是访问权限不同。默认情况下，struct 成员的访问级别为public，而class 成员的访问级别是private 。

虚函数的作用？

参考回答：

虚函数作用是：C++中用于实现多态的机制。核心理念就是通过基类访问派生类定义的函数。对于虚函数来说，父类和子类都有各自的版本。由多态方式调用的时候动态绑定。

请你说说虚函数表具体是怎样实现运行时多态的？

参考回答：

子类若重写父类虚函数，虚函数表中，该函数的地址会被替换，对于存在虚函数的类的对象，在VS中，对象的对象模型的头部存放指向虚函数表的指针，通过该机制实现多态。

虚函数与纯虚函数的区别？

参考回答：

虚函数: 在基类用virtual声明成员函数为虚函数。
纯虚函数: 除了有virtual 关键字外，还令它等于0。例如; virtual void MyString()=0; 纯虚函数一定没有定义，纯虚函数用来规范派生类的行为，即接口。包含纯虚函数的类是抽象类，抽象类不能定义实例，但可以声明指向实现该抽象类的具体类的指针或引用。

请你来说一下静态函数和虚函数的区别

参考回答：

静态函数在编译的时候就已经确定运行时机，虚函数在运行的时候动态绑定。虚函数因为用了虚函数表机制，调用的时候会增加一次内存开销。

C++中不能被基类继承的函数有哪些？

构造函数，析构函数，友元函数，拷贝构造函数。

C++中如何阻止一个类被实例化？

参考回答：

（1）将类定义为抽象基类或者将构造函数声明为private；
（2）不允许类外部创建类对象，只能在类内部创建对象。

C++中一个空类，包含哪些成员函数？

参考回答：

（1）默认构造函数 Class()《参考C++ 构造函数初步认识》
（2） 复制构造函数 Class(const Class& other)
（3）析构函数 ~Class()
（4） 赋值函数 Class & operator=(const Class & other)

什么是重载与覆盖？

参考回答：

成员函数被重载的特征：
（1）相同的范围（在同一个类中）；
（2）函数名字相同；
（3）参数不同；
（4）virtual关键字可有可无。

可以参考这篇《什么是函数重载？》

覆盖是指派生类函数覆盖基类函数，特征是：
（1）不同的范围（分别位于派生类与基类）；
（2）函数名字相同；
（3）参数相同；
（4）基类函数必须有virtual关键字。

请你回答一下野指针是什么？

参考回答：

野指针就是指向一个已删除的对象或者未申请访问受限内存区域的指针。

new和malloc的区别？

参考回答：

（1）c/c++中，new和malloc都是申请内存。new分配内存按照数据类型进行分配，malloc分配内存按照大小分配；

（2）new不仅分配一段内存，而且会调用构造函数，但是malloc则不会。

（3）new返回的是指定对象的指针，而malloc返回值的是void*，因此malloc的返回值一般都需要进行类型转化；

（4）new是一个操作符可以重载，malloc是一个库函数；

（5）new分配的内存要用delete销毁，malloc要用free来销毁；delete销毁的时候会调用对象的析构函数，而free则不会；

（6）new和new[]的区别，new[]一次分配所有内存，多次调用构造函数，分别搭配使用delete和delete[]，同理，delete[]多次调用析构函数，销毁数组中的每个对象。而malloc则只能sizeof(int) * n；

请你回答一下new/delete与malloc/free的区别是什么

参考回答：

首先，new/delete是C++的关键字，而malloc/free是C语言的库函数，后者使用必须指明申请内存空间的大小，对于类类型的对象，后者不会调用构造函数和析构函数。

深拷贝和浅拷贝的区别？

参考回答：

浅拷贝

只是对指针的拷贝，拷贝之后，两个指针同时指向同一个内存。 char p1[]=“hello”; char *p2 = p1,只是拷贝指针，没有分配内存。

深拷贝

深拷贝就是拷贝他的值，重新生成的对像。 char p1[]=“hello”;
char *p2=new char[];p2=p1;重新new申请内存空间。

简单的说，深浅拷贝就是看是否重新申请内存空间。

内联函数与宏定义的区别？

（1）内联函数是用来消除函数调用时的时间开销。频繁被调用的短小函数非常受益。内联函数会检查参数类型。
（2）宏定义不检查函数参数，返回值什么的，只是编译过程中替换展开。

请你说一说strcpy和strlen

参考回答：

strcpy是字符串拷贝函数，原型：
char strcpy(char dest, const char *src);

从src逐字节拷贝到dest，直到遇到'\0'结束，因为没有指定长度，可能会导致拷贝越界，造成缓冲区溢出漏洞，安全版本是strncpy函数。
strlen函数是计算字符串长度的函数，返回从开始到'\0'之间的字符个数。

请你来说一说隐式类型转换

参考回答：

首先，对于内置类型，低精度的变量给高精度变量赋值会发生隐式类型转换，其次，对于只存在单个参数的构造函数的对象构造来说，函数调用可以直接使用该参数传入，编译器会自动调用其构造函数生成临时对象。

C++中内存管理分为哪些？

参考回答：

（1）栈，在执行局部函数时（在函数内部，或复合语句内部定义的变量），函数内局部变量的存储单元都可以在栈上创建，函数执行结束由系统自动被释放。栈内存分配运算内置于处理器的指令集中，效率高，分配的内存容量有限。

（2）堆，就是那些由malloc等分配的内存块，用free来释放内存。

（3）自由存储区，那些由new分配的内存块，由应用程序去控制，一般一个new就要对应一个delete。如果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。

（4）全局/静态存储区，全局变量和静态变量（定义的时候，前面加 static 修饰）被分配到同一块内存中，在以前的C语言中，全局变量又分为初始化的和未初始化的，在C++里面没有这个区分了，他们共同占用同一块内存区。

（5）常量存储区，这是一块比较特殊的存储区，他们里面存放的是常量，不允许修改。

怎么判断一个数是二的倍数，怎么求一个数中有几个1，说一下你的

思路并手写代码

参考回答：

1、判断一个数是不是二的倍数，即判断该数二进制末位是不是0：

a % 2 == 0 或者a & 0x0001 == 0。

2、求一个数中1的位数，可以直接逐位除十取余判断：

```
int fun(long x)
{
int _count = 0;
while(x)
{
    if(x % 10 == 1)
        ++_count;
    x /= 10;
}
return _count;
}
int main()
{
    cout << fun(123321) << endl;
    return 0;
}
```

编写类String的构造函数、析构函数和赋值函数？

已知类String的原型为：

```
class String
{
  public:
    String(const char *str = NULL); // 普通构造
    String(const String &other);     // 拷贝
    ~ String(void);                  // 析构
    String & operate =(const String &other);
  private:
    char    *m_data;                 // 用于保存
};
```

请编写String的上述4个函数。

```
// String的析构函数
String::~String(void)
{
    delete [] m_data;

}

// String的普通构造函数
String::String(const char *str)
{
    if(str==NULL)
    {
        m_data = new char[1];    // 若能加 NULL 判
```

```
        *m_data = '\0';
    }
    else
    {
        int length = strlen(str);
        m_data = new char[length+1];  // 若能加 NU
        strcpy(m_data, str);
    }
}

// 拷贝构造函数
String::String(const String &other)
{
    int length = strlen(other.m_data);
    m_data = new char[length+1];       // 若能加 NU
    strcpy(m_data, other.m_data);
}
// 赋值函数
String & String::operate =(const String &other)
{
    if(this == &other)  // (1) 检查自赋值
    return *this;
    delete [] m_data;// (2) 释放原有的内存资源
    int length = strlen(other.m_data);  // （3）分
    m_data = new char[length+1];          // 若能加
    strcpy(m_data, other.m_data);
    return *this;   // （4）返回本对象的引用
}
```

输入一个链表,输出该链表中倒数第k个节点。

```
/*
struct ListNode
{
int val;
struct ListNode *next;
ListNode(int x) : val(x), next(NULL) { }
};*/

class Solution
{
//实现该函数
    public: ListNode* FindKthToTail(ListNode* pLi
    {
      if(pListHead == NULL)
        return NULL;
      ListNode* p1 = pListHead, *p2 = pListHead;
      for(int i = 0;i < k;i++)
       {
        if(p2 != NULL)
            p2 = p2->next;
        else
            return NULL;
        }
        while(p2 != NULL)
        {
            p1 = p1->next;
```

```
            p2 = p2->next;
        }
        return p1;
    }
};
```

说一说c++中四种cast转换

参考回答：

C++中四种类型转换是：static_cast, dynamic_cast, const_cast, reinterpret_cast

C++中四种类型转换可以参考这篇《C++四种强制类型转换运算符》

const_cast

用于将const变量转为非const

static_cast

用于各种隐式转换，比如非const转const，void*转指针等, static_cast能用于多态向上转化，如果向下转能成功但是不安全，结果未知；

dynamic_cast

用于动态类型转换。只能用于含有虚函数的类，用于类层次间的向上和向下转化。只能转指针或引用。向下转化时，如果是非法的对于指针返回NULL，对于引用抛异常。要深入了解内部转换的原理。

向上转换：指的是子类向基类的转换

向下转换：指的是基类向子类的转换

它通过判断在执行到该语句的时候变量的运行时类型和要转换的类型是否相同来判断是否能够进行向下转换。

reinterpret_cast

几乎什么都可以转，比如将int转指针，可能会出问题，尽量少用；

为什么不使用C的强制转换？

C的强制转换表面上看起来功能强大什么都能转，但是转化不够明确，不能进行错误检查，容易出错。

C++中算法所要求迭代器操作可以分为哪5个迭代器类别？

（1）输入迭代器：只读，不写；单遍扫描，只能递增；
（2）输出迭代器：只写，不读；单遍扫描，只能递增；
（3）前向迭代器：可读可写，多遍扫描，只能递增；
（4）双向迭代器：可读可写，多遍扫描，可递增递减；
（5）随机访问迭代器：可读可写，多遍扫描，支持全部迭代器运算；

STL中各种容器的底层实现

（1）vector 底层数据结构为数组 ，支持快速随机访问

（2）list 底层数据结构为双向链表，支持快速增删

（3）deque：双端队列 底层数据结构为一个中央控制器和多个缓冲区，

（4）stack 底层一般用list或deque实现，封闭头部即可，不用vector的原因应该是容量大小有限制，扩容耗时

（5）queue 底层一般用list或deque实现，封闭头部即可，不用vector的原因应该是容量大小有限制，扩容耗时

（6）set 底层数据结构为红黑树，有序，不重复

（7）multiset 底层数据结构为红黑树，有序，可重复

（8）map 底层数据结构为红黑树，有序，不重复

（9）multimap 底层数据结构为红黑树，有序，可重复

（10）hash_set 底层数据结构为hash表，无序，不重复

（11）hash_multiset 底层数据结构为hash表，无序，可重复

（12）hash_map 底层数据结构为hash表，无序，不重复

（13）hash_multimap 底层数据结构为hash表，无序，可重复

什么是二叉树？什么是红黑树？

二叉树

二叉树是每个结点最多有两个子树的树结构。通常子树被称作“左子树”和“右子树”。二叉树常被用于实现二叉查找树和二叉堆。

（1）若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值；

（2）若它的右子树上所有结点的值均大于它的根节点的值；

《参考数据结构——二叉搜索树》

红黑树

红黑树是一种“平衡的”二叉查找树，它是一种经典高效的算法，能够保证在最坏的情况下动态集合操作的时间为O（lgn）。

（1）节点为红色或者黑色；

（2）根节点为黑色；

（3）从根节点到每个叶子节点经过的黑色节点个数的和相同；

（4）如果父节点为红色，那么其子节点就不能为红色。

请你回答一下如何判断内存泄漏？

内存泄漏通常是由于调用了malloc/new等内存申请的操作，但是缺少了对应的free/delete。为了判断内存是否泄露，我们一方面可以使用linux环境下的内存泄漏检查工具Valgrind,另一方面我们在写代码时可以添加内存申请和释放的统计功能，统计当前申请和释放的内存是否一致，以此来判断内存是否泄露。

请你来说一下什么时候会发生段错误？

段错误通常发生在访问非法内存地址的时候，具体来说分为以下几种情况：

1、使用野指针。
2、试图修改字符串常量的内容。

C/C++内存分配？

内存分配的类型

在C/C++中内存分为5个区，分别为栈区、堆区、全局/静态存储区、常量存储区、代码区。

静态内存分配:编译时分配。包括:全局、静态全局、静态局部三种变量。

动态内存分配:运行时分配。包括:栈(stack): 局部变量。堆(heap): c语言中用到的变量被动态的分配在内存中。(malloc或calloc、realloc、free函数)

变量的内存分配

内存分配可以先参考这篇《C语言——内存分配，你真的知道吗？》

栈区（stack）：指那些由编译器在需要的时候分配，不需要时自动清除的变量所在的储存区，如函数执行时，函数的形参以及函数内的局部变量分配在栈区，函数运行结束后，形参和局部变量去栈（自动释放）。栈内存分配运算内置与处理器的指令集中，效率高但是分配的内存空间有限。

堆区（heap）：指哪些由程序员手动分配释放的储存区，如果程序员不释放这块内存，内存将一直被占用，直到程序运行结束由系统自动收回，c语言中使用malloc，free申请和释放空间。

静态储存区（static）：全局变量和静态变量的储存是放在一块的，其中初始化的全局变量和静态变量在一个区

域，这块空间当程序运行结束后由系统释放。

常量储存区（ const ）：常量字符串就是储存在这里的，如“ABC”字符串就储存在常量区，储存在常量区的只读不可写。const修饰的全局变量也储存在常量区，const修饰的局部变量依然在栈上。

程序代码区：存放源程序的二进制代码。

请自己设计一下如何采用单线程的方式处理高并发？

在单线程模型中，可以采用I/O复用来提高单线程处理多个请求的能力，然后再采用事件驱动模型，基于异步回调来处理事件来。

请你说一说C++ STL 的内存优化？

1）二级配置器结构
STL内存管理使用二级内存配置器。

1、第一级配置器
第一级配置器以malloc()，free()，realloc()等C函数执行实际的内存配置、释放、重新配置等操作，并且能在内存需求不被满足的时候，调用一个指定的函数。
一级空间配置器分配的是大于128字节的空间
如果分配不成功，调用句柄释放一部分内存
如果还不能分配成功，抛出异常

2、第二级配置器
在STL的第二级配置器中多了一些机制，避免太多小区块造成的内存碎片，小额区块带来的不仅是内存碎片，配置时还有额外的负担。区块越小，额外负担所占比例就越大。

3、分配原则
如果要分配的区块大于128bytes，则移交给第一级配置器处理。
如果要分配的区块小于128bytes，则以内存池管理（memory pool），又称之次层配置（sub-allocation）：每次配置一大块内存，并维护对应的16个空闲链表（free-list）。下次若有相同大小的内存需求，

则直接从free-list中取。如果有小额区块被释放，则由配置器回收到free-list中。
当用户申请的空间小于128字节时，将字节数扩展到8的倍数，然后在自由链表中查找对应大小的子链表
如果在自由链表查找不到或者块数不够，则向内存池进行申请，一般一次申请20块
如果内存池空间足够，则取出内存
如果不够分配20块，则分配最多的块数给自由链表，并且更新每次申请的块数
如果一块都无法提供，则把剩余的内存挂到自由链表，然后向系统heap申请空间，如果申请失败，则看看自由链表还有没有可用的块，如果也没有，则最后调用一级空间配置器

2）二级内存池
二级内存池采用了16个空闲链表，这里的16个空闲链表分别管理大小为8、16、24……120、128的数据块。这里空闲链表节点的设计十分巧妙，这里用了一个联合体既可以表示下一个空闲数据块（存在于空闲链表中）的地址，也可以表示已经被用户使用的数据块（不存在空闲链表中）的地址。

1、空间配置函数allocate
首先先要检查申请空间的大小，如果大于128字节就调

用第一级配置器，小于128字节就检查对应的空闲链表，如果该空闲链表中有可用数据块，则直接拿来用（拿取空闲链表中的第一个可用数据块，然后把该空闲链表的地址设置为该数据块指向的下一个地址），如果没有可用数据块，则调用refill重新填充空间。

2、空间释放函数deallocate
首先先要检查释放数据块的大小，如果大于128字节就调用第一级配置器，小于128字节则根据数据块的大小来判断回收后的空间会被插入到哪个空闲链表。

3、重新填充空闲链表refill
在用allocate配置空间时，如果空闲链表中没有可用数据块，就会调用refill来重新填充空间，新的空间取自内存池。缺省取20个数据块，如果内存池空间不足，那么能取多少个节点就取多少个。
从内存池取空间给空闲链表用是chunk_alloc的工作，首先根据end_free-start_free来判断内存池中的剩余空间是否足以调出nobjs个大小为size的数据块出去，如果内存连一个数据块的空间都无法供应，需要用malloc取堆中申请内存。
假如山穷水尽，整个系统的堆空间都不够用了，malloc失败，那么chunk_alloc会从空闲链表中找是否有大的数据块，然后将该数据块的空间分给内存池（这个数据块

会从链表中去除）。

4、总结：

使用allocate向内存池请求size大小的内存空间，如果需要请求的内存大小大于128bytes，直接使用malloc。

如果需要的内存大小小于128bytes，allocate根据size找到最适合的自由链表。

a. 如果链表不为空，返回第一个node，链表头改为第二个node。

b. 如果链表为空，使用blockAlloc请求分配node。

x. 如果内存池中有大于一个node的空间，分配竟可能多的node(但是最多20个)，将一个node返回，其他的node添加到链表中。

y. 如果内存池只有一个node的空间，直接返回给用户。

z. 若果如果连一个node都没有，再次向操作系统请求分配内存。

①分配成功，再次进行b过程。

②分配失败，循环各个自由链表，寻找空间。

I. 找到空间，再次进行过程b。

II. 找不到空间，抛出异常。

用户调用deallocate释放内存空间，如果要求释放的内存空间大于128bytes，直接调用free。

否则按照其大小找到合适的自由链表，并将其插入。

什么是右值引用，跟左值又有什么区别？

右值引用是C++11中引入的新特性，它实现了转移语义和精确传递。它的主要目的有两个方面：

> 1、消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率。
> 2、能够更简洁明确地定义泛型函数。

左值和右值的概念

左值：能对表达式取地址、或具名对象/变量。一般指表达式结束后依然存在的持久对象。
右值：不能对表达式取地址，或匿名对象。一般指表达式结束就不再存在的临时对象。

右值引用和左值引用的区别

1、左值可以寻址，而右值不可以。
2、左值可以被赋值，右值不可以被赋值，可以用来给左值赋值。
3、左值可变,右值不可变（仅对基础类型适用，用户自定义类型右值引用可以通过成员函数改变）。

include头文件的顺序以及双引号""和尖括号<>的区别？

Include头文件的顺序：对于include的头文件来说，如果在文件a.h中声明一个在文件b.h中定义的变量，而不引用b.h。那么要在a.c文件中引用b.h文件，并且要先引用b.h，后引用a.h,否则汇报变量类型未声明错误。

双引号和尖括号的区别：编译器预处理阶段查找头文件的路径不一样。

对于使用双引号包含的头文件，查找头文件路径的顺序为：

当前头文件目录
编译器设置的头文件路径（编译器可使用-I显式指定搜索路径）
系统变量CPLUS_INCLUDE_PATH/C_INCLUDE_PATH指定的头文件路径

对于使用尖括号包含的头文件，查找头文件的路径顺序为：
编译器设置的头文件路径（编译器可使用-I显式指定搜索路径）
系统变量CPLUS_INCLUDE_PATH/C_INCLUDE_PATH指定的头文件路径

请问C++11有哪些新特性？

C++11 最常用的新特性如下：

auto关键字：编译器可以根据初始值自动推导出类型。但是不能用于函数传参以及数组类型的推导 《参考C++11特性：auto关键字》

nullptr关键字：nullptr是一种特殊类型的字面值，它可以被转换成任意其它的指针类型；而NULL一般被宏定义为0，在遇到重载时可能会出现问题。

智能指针：C++11新增了std::shared_ptr、std::weak_ptr等类型的智能指针，用于解决内存管理的问题。

初始化列表：使用初始化列表来对类进行初始化

右值引用：基于右值引用可以实现移动语义和完美转发，消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率

atomic原子操作用于多线程资源互斥操作

新增STL容器array以及tuple

请你说说fork,wait,exec函数

参考回答：

父进程产生子进程使用fork拷贝出来一个父进程的副本，此时只拷贝了父进程的页表，两个进程都读同一块内存，当有进程写的时候使用写实拷贝机制分配内存，exec函数可以加载一个elf文件去替换父进程，从此父进程和子进程就可以运行不同的程序了。fork从父进程返回子进程的pid，从子进程返回0.调用了wait的父进程将会发生阻塞，直到有子进程状态改变,执行成功返回0，错误返回-1。exec执行成功则子进程从新的程序开始运行，无返回值，执行失败返回-1

请你讲讲STL有什么基本组成

参考回答：

STL主要由：以下几部分组成：
容器、迭代器、仿函数、算法、分配器、配接器
他们之间的关系：分配器给容器分配存储空间，算法通过迭代器获取容器中的内容，仿函数可以协助算法完成各种操作，配接器用来套接适配仿函数

请你说一说STL中MAP数据存放形式

参考回答：

红黑树。unordered map底层结构是哈希表。

抽象类、接口类、聚合类

抽象类：含有纯虚函数的类
接口类：仅含有纯虚函数的抽象类
聚合类：用户可以直接访问其成员，并且具有特殊的初始化语法形式。满足如下特点：
所有成员都是 public
没有定义任何构造函数
没有类内初始化
没有基类，也没有 virtual 函数

请你说一说vector和list的区别

Vector

连续存储的容器，动态数组，在堆上分配空间
底层实现：数组

List

动态链表，在堆上分配空间，每插入一个元数都会分配空间，每删除一个元素都会释放空间。
底层：双向链表

迭代器和指针的区别

参考回答：

迭代器不是指针，是类模板，表现的像指针。他只是模拟了指针的一些功能，通过重载了指针的一些操作符，->、*、++、–等。迭代器封装了指针，是一个“可遍历STL（ Standard Template Library）容器内全部或部分元素”的对象， 本质是封装了原生指针，是指针概念的

一种提升（lift），提供了比指针更高级的行为，相当于一种智能指针，他可以根据不同类型的数据结构来实现不同的++，–等操作。

迭代器返回的是对象引用而不是对象的值，所以cout只能输出迭代器使用*取值后的值而不能直接输出其自身。

请你来说一下一个C++源文件从文本到可执行文件经历的过程？

对于C++源文件，从文本到可执行文件一般需要四个过程：

预处理阶段：对源代码文件中文件包含关系（头文件）、预编译语句（宏定义）进行分析和替换，生成预编译文件。

编译阶段：将经过预处理后的预编译文件转换成特定汇编代码，生成汇编文件

汇编阶段：将编译阶段生成的汇编文件转化成机器码，生成可重定位目标文件

链接阶段：将多个目标文件及所需要的库连接成最终的可执行目标文件

请你说一说C++的内存管理是怎样的？

参考回答：

在C++中，虚拟内存分为代码段、数据段、BSS段、堆区、文件映射区以及栈区六部分。

代码段:包括只读存储区和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码。

数据段：存储程序中已初始化的全局变量和静态变量

bss 段：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量。

堆区：调用new/malloc函数时在堆区动态分配内存，同时需要调用delete/free来手动释放申请的内存。

映射区:存储动态链接库以及调用mmap函数进行的文件映射

栈：使用栈空间存储函数的返回地址、参数、局部变量、返回值

红黑树的特征是什么？

参考回答：

1、节点是红色或黑色。
2、根是黑色。
3、所有叶子都是黑色（叶子是 NIL 节点）。
4、 每个红色节点必须有两个黑色的子节点。（从每个叶子到根的所有路径上不能有两个连续的红色节点。）（新增节点的父节点必须相同）
5、 从任一节点到其每个叶子的所有简单路径都包含相同数目的黑色节点。（新增节点必须为红）

应用

关联数组：如 STL 中的 map、set

进程与线程

参考回答：

对于有线程系统：

进程是资源分配的独立单位

线程是资源调度的独立单位

对于无线程系统：

进程是资源调度、分配的独立单位

进程之间的通信方式以及优缺点

参考回答：

管道（PIPE）

有名管道：一种半双工的通信方式，它允许无亲缘关系进程间的通信

优点：可以实现任意关系的进程间的通信
缺点：
长期存于系统中，使用不当容易出错
缓冲区有限

无名管道：一种半双工的通信方式，只能在具有亲缘关系的进程间使用（父子进程）
优点：简单方便
缺点：
局限于单向通信
只能创建在它的进程以及其有亲缘关系的进程之间
缓冲区有限

信号量（Semaphore）：一个计数器，可以用来控制多个线程对共享资源的访问
优点：可以同步进程
缺点：信号量有限
信号（Signal）：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

消息队列（Message Queue）：是消息的链表，存放在内核中并由消息队列标识符标识
优点：可以实现任意进程间的通信，并通过系统调用函数来实现消息发送和接收之间的同步，无需考虑同步问

题，方便
缺点：信息的复制需要额外消耗 CPU 的时间，不适宜于信息量大或操作频繁的场合

共享内存（Shared Memory）：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
优点：无须复制，快捷，信息量大
缺点：
通信是通过将共享空间缓冲区直接附加到进程的虚拟地址空间中来实现的，因此进程间的读写操作的同步问题
利用内存缓冲区直接交换信息，内存的实体存在于计算机中，只能同一个计算机系统中的诸多进程共享，不方便网络通信

套接字（Socket）：可用于不同计算机间的进程通信
优点：
传输数据为字节级，传输数据可自定义，数据量小效率高
传输数据时间短，性能高
适合于客户端和服务器端之间信息实时交互
可以加密,数据安全性强
缺点：需对传输的数据进行解析，转化成应用级的数据。

线程之间的通信方式

参考回答：

锁机制：包括互斥锁/量（mutex）、读写锁（reader-writer lock）、自旋锁（spin lock）、条件变量（condition）

互斥锁/量（mutex）：提供了以排他方式防止数据结构被并发修改的方法。
读写锁（reader-writer lock）：允许多个线程同时读共享数据，而对写操作是互斥的。
自旋锁（spin lock）与互斥锁类似，都是为了保护共享资源。互斥锁是当资源被占用，申请者进入睡眠状态；而自旋锁则循环检测保持者是否已经释放锁。
条件变量（condition）：可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件的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。

信号量机制(Semaphore)
无名线程信号量

命名线程信号量

信号机制(Signal)：类似进程间的信号处理

屏障（ barrier ）：屏障允许每个线程等待，直到所有的合作线程都达到某一点，然后从该点继续执行。

什么是死锁

参考回答：

原因：

系统资源不足

资源分配不当

进程运行推进顺序不合适

产生条件：

互斥

请求和保持

不剥夺

环路

计算机网络体系结构：

TCP 与 UDP 的区别

1、TCP 面向连接，UDP 是无连接的；
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 的逻辑通信信道是全双工的可靠信道；UDP 则是不可靠信道

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 面向字节流（可能出现黏包问题），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（不会出现黏包问题）

6、 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）

7、 TCP 首部开销20字节；UDP 的首部开销小，只有 8 个字节

TCP 黏包问题

参考回答：

原因：

TCP 是一个基于字节流的传输服务（ UDP 基于报文

的），“流” 意味着 TCP 所传输的数据是没有边界的。所以可能会出现两个数据包黏在一起的情况。

解决：

> 1、发送定长包。如果每个消息的大小都是一样的，那么在接收对等方只要累计接收数据，直到数据等于一个定长的数值就将它作为一个消息。
> 2、 包头加上包体长度。包头是定长的 4 个字节，说明了包体的长度。接收对等方先接收包头长度，依据包头长度来接收包体。
> 3、在数据包之间设置边界，如添加特殊符号 \r\n 标记。FTP 协议正是这么做的。但问题在于如果数据正文中也含有 \r\n，则会误判为消息的边界。
> 4、使用更加复杂的应用层协议。

TCP 为什么要进行三次握手？

参考回答：

可以先参考这篇《网络编程-从TCP三次握手说起》

众所周知tcp传输层协议在建立连接的时候需要三次才能建立起一个真正的可靠连接，可是为什么是三次呢，不可以是两次，四次等等呢？

一个TCP连接是全双工的，即数据在两个方向上能同时传输。因此，建立连接的过程也就必须确认双方的收发能力都是正常的。

如果采用两次握手，那么服务端就会认为这个报文是新的连接请求，于是建立连接，等待客户端发送数据，但是实际上客户端根本没有发出建立请求，也不会理睬服务端，因此导致服务端空等而浪费资源。通信双方协商好一个初始 seq，至少需要一次 SYN 和 一次 ACK。

由于 TCP 是全双工的，所以 TCP 要协商两个初始 seq，所以双方各需要一次 SYN 和一次 ACK。

四次握手是否可以呢？

完全可以！但是没有必要！在服务端收到SYN之后，它可以先回ACK，再发送SYN，但是这两个信息可以一起发送出去，因此没有必要。简单优化，可以将中间的 ACK + SYN 合并。所以就变成了 TCP 建立连接的三次握手。

TIME_WAIT的目的？

参考回答：

TIME_WAIT发生在TCP挥手的第四次挥手之后，这个状态的主要目的在于，客户端要确认最后一个ACK能够顺利的发送到服务端，当服务端没有收到ACK确认报文，那么一定是会重传这个FIN包。

TIME_WAIT为什么是2倍的MSL？

MSL（Maximum Segment Lifetime）是报文段的最大生存时间。

当第四次挥手发送完一个ACK报文的时候，它到达服务端的最大报文段传输时间为MSL，在极端情况下，刚好一个MSL的时候ACK报文段丢失，那么服务端就会重传一个FIN，那么它发送到客户端的时间就又是一个MSL，那么这种情况是极端的情况，也就是最大会有2倍的MSL时间间隔，当丢失后重传的FIN到达客户端的时候，那客户端就会重新设置为2倍的MSL,并且重传ACK。

请你说说select，epoll的区别，原理，性能，限制都说一说

参考回答：

1）IO多路复用

IO复用模型在阻塞IO模型上多了一个select函数，select函数有一个参数是文件描述符集合，意思就是对这些的

文件描述符进行循环监听，当某个文件描述符就绪的时候，就对这个文件描述符进行处理。

这种IO模型是属于阻塞的IO。但是由于它可以对多个文件描述符进行阻塞监听，所以它的效率比阻塞IO模型高效。

IO多路复用就是我们说的select，poll，epoll。select/epoll的好处就在于单个process就可以同时处理多个网络连接的IO。它的基本原理就是select，poll，epoll这个function会不断的轮询所负责的所有socket，当某个socket有数据到达了，就通知用户进程。

当用户进程调用了select，那么整个进程会被block，而同时，kernel会“监视”所有select负责的socket，当任何一个socket中的数据准备好了，select就会返回。这个时候用户进程再调用read操作，将数据从kernel拷贝到用户进程。

所以，I/O 多路复用的特点是通过一种机制一个进程能同时等待多个文件描述符，而这些文件描述符（套接字描述符）其中的任意一个进入读就绪状态，select()函数就可以返回。

I/O多路复用和阻塞I/O其实并没有太大的不同，事实上，还更差一些。因为这里需要使用两个system call (select 和 recvfrom)，而blocking IO只调用了一个system call (recvfrom)。但是，用select的优势在于它可以同时处理多个connection。

所以，如果处理的连接数不是很高的话，使用select/epoll的web server不一定比使用multi-threading + blocking IO的web server性能更好，可能延迟还更大。select/epoll的优势并不是对于单个连接能处理得更快，而是在于能处理更多的连接。）

在IO multiplexing Model中，实际中，对于每一个socket，一般都设置成为non-blocking，但是，如上图所示，整个用户的process其实是一直被block的。只不过process是被select这个函数block，而不是被socket IO给block。

select

select：是最初解决IO阻塞问题的方法。用结构体fd_set来告诉内核监听多个文件描述符，该结构体被称

为描述符集。由数组来维持哪些描述符被置位了。对结构体的操作封装在三个宏定义中。通过轮寻来查找是否有描述符要被处理。

存在的问题：

1）内置数组的形式使得select的最大文件数受限与FD_SIZE；

2）每次调用select前都要重新初始化描述符集，将fd从用户态拷贝到内核态，每次调用select后，都需要将fd从内核态拷贝到用户态；

3）轮寻排查当文件描述符个数很多时，效率很低；

poll

poll：通过一个可变长度的数组解决了select文件描述符受限的问题。数组中元素是结构体，该结构体保存描述符的信息，每增加一个文件描述符就向数组中加入一个结构体，结构体只需要拷贝一次到内核态。poll解决了

select重复初始化的问题。轮寻排查的问题未解决。

epoll

epoll：轮寻排查所有文件描述符的效率不高，使服务器并发能力受限。因此，epoll采用只返回状态发生变化的文件描述符，便解决了轮寻的瓶颈。

epoll对文件描述符的操作有两种模式：LT（level trigger）和ET（edge trigger）。LT模式是默认模式

LT模式：
LT(level triggered)是缺省的工作方式，并且同时支持block和no-block socket.在这种做法中，内核告诉你一个文件描述符是否就绪了，然后你可以对这个就绪的fd进行IO操作。如果你不作任何操作，内核还是会继续通知你的。

ET模式：
ET(edge-triggered)是高速工作方式，只支持no-block socket。在这种模式下，当描述符从未就绪变为就绪时，内核通过epoll告诉你。然后它会假设你知道文件描

述符已经就绪，并且不会再为那个文件描述符发送更多的就绪通知，直到你做了某些操作导致那个文件描述符不再为就绪状态了(比如，你在发送，接收或者接收请求，或者发送接收的数据少于一定量时导致了一个EWOULDBLOCK 错误）。但是请注意，如果一直不对这个fd作IO操作(从而导致它再次变成未就绪)，内核不会发送更多的通知(only once)

ET模式在很大程度上减少了epoll事件被重复触发的次数，因此效率要比LT模式高。epoll工作在ET模式的时候，必须使用非阻塞套接口，以避免由于一个文件句柄的阻塞读/阻塞写操作把处理多个文件描述符的任务饿死。

LT模式与ET模式的区别如下：
LT模式：当epoll_wait检测到描述符事件发生并将此事件通知应用程序，应用程序可以不立即处理该事件。下次调用epoll_wait时，会再次响应应用程序并通知此事件。
ET模式：当epoll_wait检测到描述符事件发生并将此事件通知应用程序，应用程序必须立即处理该事件。如果不处理，下次调用epoll_wait时，不会再次响应应用程序并通知此事件。

ARP是什么？ARP是怎么找到MAC地址的？

参考回答：

ARP协议是地址解析协议（Address Resolution Protocol）是通过解析IP地址得到MAC地址的，是一个在网络协议包中极其重要的网络传输协议，它与网卡有着极其密切的关系。

在TCP/IP分层结构中，把ARP划分为网络层，为什么呢，因为在网络层看来，源主机与目标主机是通过IP地址进行识别的，而所有的数据传输又依赖网卡底层硬件，即链路层，那么就需要将这些IP地址转换为链路层可以识别的东西，在所有的链路中都有着自己的一套寻址机制，如在以太网中使用MAC地址进行寻址，以标识不同的主机，那么就需要有一个协议将IP地址转换为MAC地址。

由此就出现了ARP协议，所有ARP协议在网络层被应用，它是网络层与链路层连接的重要枢纽，每当有一个数据要发送的时候都需要在通过ARP协议将IP地址转换

成MAC地址

推荐好文

GDB调试入门，看这篇就够了!

Linux 系统调用和库函数的区别

数据结构——二叉搜索树

web性能压力测试工具：Webbench 源码分析

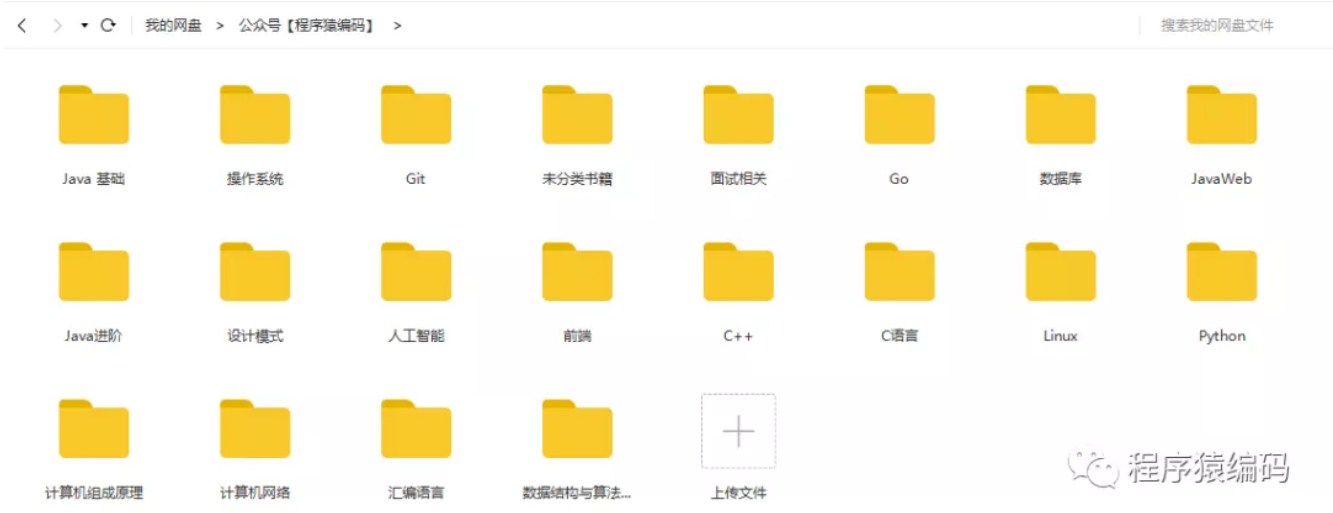
GCC编译过程（预处理->编译->汇编->链接）

计算机网络知识总结

网络编程-一个简单的客户端与服务器程序，（看这篇就够了）

Linux 性能分析命令

欢迎关注公众号**【程序猿编码】**，添加本人微信号(**17865354792**)，回复：领取学习资料。进入**技术交流群**。网盘资料有如下：



写留言

阅读原文