C++知识点

C++

1.const

C语言中的const：用const修饰的变量仍然是一个变量

1. 指针常量:即指针本身的值是不可改变的，而指针指向的变量的值是可以改变的;
2. 常量指针:即指针指向的变量的值是不可改变的，而指针本身的值是可以改变的;
3. 用const修饰的变量必须在声明时进行初始化;

C++中的const：用const修饰过后，就变成常量了

1. 指针常量:即指针本身的值是不可改变的，而指针指向的变量的值是可以改变的;
2. 常量指针:即指针指向的变量的值是不可改变的，而指针本身的值是可以改变的;
3. 用const修饰的变量必须在声明时进行初始化;
4. const 修饰类成员函数时，成员函数内的成员变量是不可以改变的

考点1 const修饰的数据并非一定安全

mutable修饰成员变量时，可以在const修饰的函数里被改变

```c++
void Kf()const{
    ++_cm; // 错误
    ++_ct; // 正确
}
private:
int _cm;
mutable int _ct;
```

const仅仅是不能通过变量名去操作这块内存，但是可以通过其它方法，如通过指针是可以修改被const修饰的那块内存的。

考点2 const 成员函数mutable 修饰

函数不改变类的成员变量，除非变量被

```
const Type Func(Type Val);
```

考点3 const 与 #define相比不同之处

const常量有数据类型，而宏常量没有数据类型
编译器可以对前者进行类型安全检查，而对后者只进行字符替换，没有类型安全检查

2.多态

多态类型:

静态多态:函数重载和运算符重载属于静态多态

动态多态：通过继承重写基类的虚函数实现的多态，**在程序运行时**根据基类的引用（指针）指向的对象来确定自己具体该调用哪一个类的虚函数。 运行时在虚函数表中寻找调用函数的地址。

STL

> C++的面向对象和泛型编程思想目的就是复用性的提升，STL是一套数据结构和算法的标准，用来减少重复性工作。

STL基本概念

STL(Standard Template Library,标准模板库)

STL从广义上分为：容器，算法，迭代器

容器和算法之间通过迭代器进行无缝连接。

STL几乎所有的代码都采用了模板类或者模板函数

STL六大组件

STL大致分为六大组件：容器，算法，迭代器，仿函数，适配器(配接器)，空间配置器

1. 容器：各种数据结构，如vector，list，deque，set，map等，用来存放数据。
2. 算法：各种常用的算法，如sort，find，copy，for，each等
3. 迭代器：扮演了容器与算法之间的胶合剂
4. 仿函数：行为类似函数，可作为算法的某种策略
5. 适配器：一种用来修饰容器或者仿函数或者迭代器接口的东西。
6. 空间适配器：负责空间的配置与管理

STL中的容器，算法，迭代器

容器

将常用的数据结构实现出来:数组，链表，树，栈，队列，集合，映射表等

容器分类：

1. 序列式容器：强调值的排序，序列式容器中的每个元素均有固定的位置。
2. 关联式容器：二叉树结构，各元素间没有严格物理上的顺序关系。

算法（algorithms）

有限的步骤，解决逻辑上或数学上的问题

算法分类:

1. 质变算法:指运算过程中会更改区间内的元素内容，例如拷贝，替换，删除等等。
2. 非质变算法:是指运算过程中不会更改区间内的元素内容，例如查找，计数，遍历，查找极值等等。

迭代器

提供一种方法，使之能够依序寻访某个容器所含的各个元素，而又无需暴露该容器的内部表示方式。

每个容器都有专属的迭代器。

迭代器种类

输入迭代器	**对数据的只读访问**	**只读，支持++，==，！=**
输出迭代器	**对数据的只写访问**	**只写，支持++**
前向迭代器	读写操作，并能向前推进迭代器	读写，支持++，==，！=
双向迭代器	读写操作，并能向前和向后操作	读写，支持++，--
随机访问迭代器	读写操作，可以以跳跃方式访问任意数据，功能最强大迭代器	读写，支持++，--，[n],-n,<,<=,>,>=

常用的迭代器为双向迭代器和随机访问迭代器

```
void myPrint(int val) {
    cout << val << endl;
}
// 遍历容器
for_each(v.begin(), v.end(), myPrint);//利用回调的原理进行处理
```

容器 string

1. string 是C++的字符串，本质上是一个类
2. string和char的区别：
   *char*是一个指针

   string是一个类，类内部封装了char*，是一个char*类型的容器.

3. string封装了一些成员方法:find 查找，copy 拷贝，delete 删除， replace 替换， insert 插入
4. string构造函数

   string()//无参构造

   string(const char* s)//有参构造 const char * str = “helloworld”;

   string(const string& str)//拷贝构造

   string(int n,char c)//使用n个字符串初始化
5. string查找和替换

   int find(const string& str,int pos = 0)const;//查找第一次出现的位置，从pos开始查找。。。

   string& replace(int pos,int n,const string& str);//替换从pos开始n个字符为字符串str

string字符串比较：按照字符串ASCII码进行比较，= 返回1，> 返回1, <返回-1

int compare(const string &s);与字符串s进行比较

例:string str1;string str2;

if(str1.compare(str2)==0){}

6. string字符获取

char& operator //通过[]方式获取字符（str.size()返回字符串长度）,例str[i]

char&at(int n);//通过st方法获取字符,例:str.at(i)

修改:

str[i] = 'x';

str.at(i) = 'x';

7. string插入和删除

插入：str.insert(1,str2)//插入位置，插入字符串

删除：str.erase(1,3)//从位置1起删除3个字符

8. string截取子串

substr = str.substr(1,3)//从位置1起截取3个字符

容器 vector

功能:vector 数据结构和数组非常相似，也称为单端数组

vector和数组区别：数组是静态空间，而vector可以动态扩展

(动态扩展不是在原空间之后续接新空间，而是找更大的空间，然后将原数据拷贝到新空间，释放原空间)

vector赋值操作:

vector& operator = (const vector &vec);//重载等号操作符 v1 = v2

assign(beg,end);//将[beg, end]区间中的数据拷贝赋值给本身 v3 = assign(v1.begin(),v1.end())

assign(n,elem);//将n个elem拷贝赋值给本身v4 = assign(4,100) -> [100,100,100,100]

vector容量和大小相关操作:

empty() //容器是否为空

capacity() //容器的容量

size() //返回容器中元素的个数

resize(int Num) //指定容器长度，若容器变长，则以默认值填充新位置，反之，则超出元素被删除

resize(int Num,elem)//同上，只不过默认值为elem

vector插入和删除操作:

push_back(ele) //尾部插入元素ele

pop_back() //删除最后一个元素

insert(const_iterator pos,ele) //迭代器指向pos插入ele

insert(const_iterator pos,count,ele) //迭代器指向pos插入count个ele

erase(const_iterator pos) //删除迭代器指向的元素

erase(const_iterator start，const_iterator end) //删除迭代器指向的元素区间

clear() //删除容器中的所有元素

vector数据存储

v.at(idx) //返回索引idx所指的数据

v[idx] //同上

v.front() //返回容器中的第一个元素

v.end() //返回容器中的最后一个元素

vector互换容器

//可以利用容器元素互换收缩容器内存

v.swap(vec) //将vec与本身的元素互换

vector预留空间

//减少vector在动态扩展容量时的扩展次数

reserve(int len) //容器预留len个元素长度，预留位置不初始化，元素不可访问

deque 容器

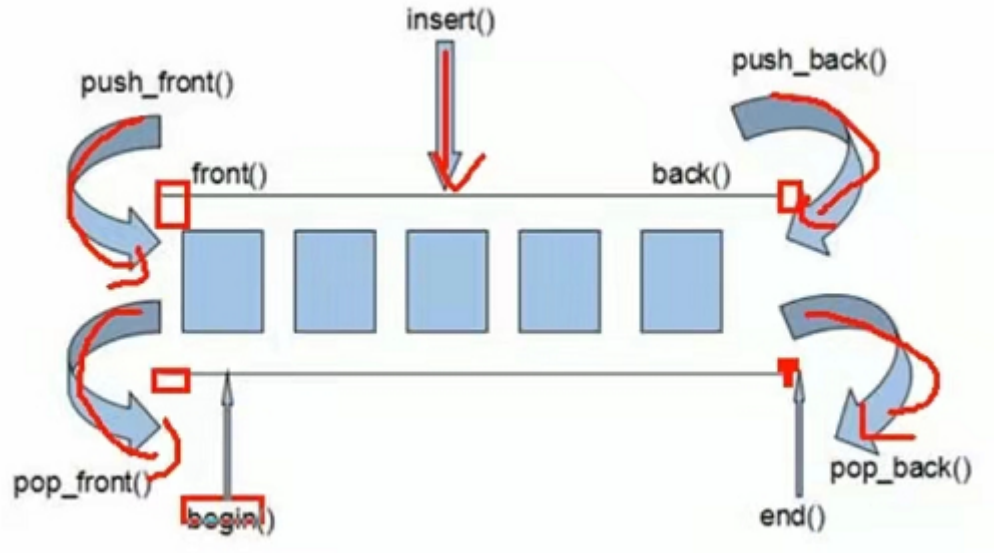
功能:双端数组，可以对头部和尾部插入删除

deque和vector区别

vector对于头部插入删除效率低，数据量越大，效率越低

deque相对而言，头部的插入删除速度更快

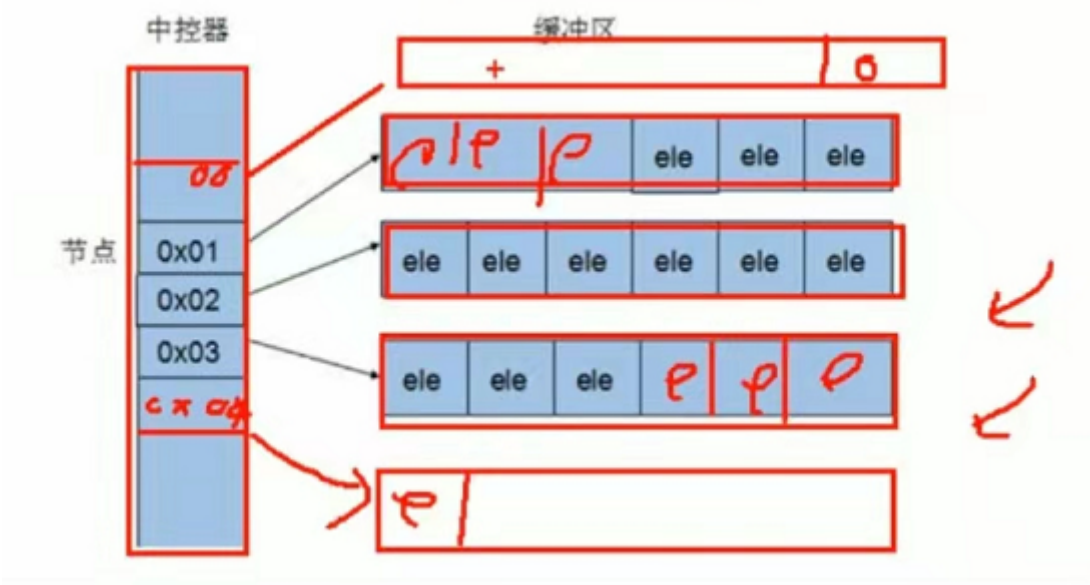
vector访问员是时速度会比deque快



deque内部工作原理:

deque内部有一个中控器，维护每段缓冲区中的内容，缓冲区中存放真实数据

中控器维护的是每个缓冲区的地址，使得使用deque时像一片连续的内存空间



deque构造函数和vector的构造函数几乎一致

deque构造函数	vector构造函数

deque d;
deque(d.beg,d.end);
deque(n,elem);
deque(const deque &deq);
	vector v
	vector(v.beg,v.end);
	vector(n,elem);
	vector(const vector &vec);

deque大小操作	vector大小操作
d.empty() d.size() d.resize(num) d.resize(elem,num)	v.empty() v.capacity() v.size() v.resize(num) v.resize(elem,num)

deque没有容量的概念，所以不需要capacity()

deque插入和删除	vector插入和删除
push_back(elem) push_front(elem) pop_back() pop_front() insert(pos,elem) insert(pos,n,elem) insert(pos,d2.beg,d2.end) clear() erase(d.beg,d.end) erase(pos)	push_back(elem) pop_back() insert(pos,elem) insert(pos,n,elem) clear() erase(v.beg,v.end) erase(pos)

问题

1. static关键字的作用

	静态成员变量（面向对象）	**静态成员函数（面向对象）**	**静态全局变量（面向过程）**	**静态局部变量（面向过程）**	**静态函数（面向过程）**
内存分配	编译时在静态(全局)区分配，程序结束时释放；sizeof计算时不会包含静态成员变量		全局区	全局区	
作用域		通过对象名或者类名访问		静态局部变量始终驻留在全局数据区，直到程序运行结束。但其作用域为局部作用域，当定义它的函数或语句块结束时，其作用域随之结束	只能在声明它的文件当中可见，不能被其它文件使用
访问限制		静态成员函数不能访问非静态成员函数和非静态成员变量；非静态成员函数可以任意地访问静态成员函数和静态数据成员			
其他	静态成员变量没有进入程序的全局命名空间，因此不存在与程序中其它全局命名冲突的可能	它不具有this指针；			

2.析构函数为虚函数的作用，解释原理；构造函数里面调用虚函数执行的是父类还是子类，为什么。

3.assert的作用。

作用:原型定义在<assert.h>中，其作用是如果它的条件返回错误，先向stderr打印一条出错信息，然后通过调用 abort 来终止程序运行。

缺点:已放弃使用assert()的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销。在调试结束后，可以通过在包含#include <assert.h>的语句之前插入 #define NDEBUG 来禁用assert调用。

4.内存对齐，对齐的意义。

Sizeof(B)为8字节的整数倍。

用空间效率来换时间效率。

5.类之间的关系有哪些。举例现实中的事物说明。

继承（父子），组合(不可分割整体)，聚合(可分割整体)，

泛化 = 实现 > 组合 > 聚合 > 关联 > 依赖

6.面向对象的设计原则有哪些。

单一职责原则	一个类，最好只做一件事
开放封闭原则	软件实体应该是可扩展的，而不可修改的
Liskov替换原则	子类可以替换基类，但是基类不一定能替换子类。拓展性,继承复用
依赖倒置原则	具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。
接口隔离原则	使用多个小的专门的接口，而不要使用一个大的总接口
合成复用原则	尽量使用合成/聚合的方式，而不是使用继承

7.用过哪些设计模式，详细讲一下观察者模式。

工厂模式

观察者模式

作用:定义对象间的一种一对多的依赖关系，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被自动更新。

使用方式:在抽象类里有一个 ArrayList 存放观察者们。

实例： 1、拍卖的时候，拍卖师观察最高标价，然后通知给其他竞价者竞价。

优点：1、观察者和被观察者是抽象耦合的。 2、建立一套触发机制。

缺点：1、如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。 2、如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。 3、观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

MVC 模式

适配器模式

作用:是作为两个不兼容的接口之间的桥梁

案例:读卡器是作为内存卡和笔记本之间的适配器

装饰器模式

功能:允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

案例:不论一幅画有没有画框都可以挂在墙上，但是通常都是有画框的，并且实际上是画框被挂在墙上。在挂在墙上之前，画可以被蒙上玻璃，装到框子里；这时画、玻璃和画框形成了一个物体。

8.进程的几种状态，对内核对象的了解，用过哪些，有什么注意事项

就绪:等CPU

运行:

阻塞:等IO资源

内核，是一个操作系统的核心。是基于硬件的第一层软件扩充，提供操作系统的最基本的功能，是操作系统工作的基础， 它负责管理系统的进程、内存、设备驱动程序、文件和网络系统，决定着系统的性能和稳定性。

现代操作系统设计中，为减少系统本身的开销，往往将一些与硬件紧密相关的（如中断处理程序、设备驱动程序等）、基本的、公共的、运行频率较高的模块（如时钟管理、进程调度等）以及关键性数据结构独立开来， 使之常驻内存，并对他们进行保护。

也就是说：内核是操作系统进行管理的一块区域（内存），对于应用程序是不可见的。

内核对象 就是在操作系统内核中进行资源分配和管理的一种数据结构。

9.malloc和new的区别

特征	new/delete	malloc/free
分配内存的位置	自由存储区	堆
内存分配成功的返回值	完整类型指针	void*
内存分配失败的返回值	默认抛出异常	返回NULL
分配内存的大小	由编译器根据类型计算得出	必须显式指定字节数
处理数组	有处理数组的new版本new[]	需要用户计算数组的大小后进行内存分配
已分配内存的扩充	无法直观地处理	使用realloc简单完成
是否相互调用	可以，看具体的operator new/delete实现	不可调用new
分配内存时内存不足	客户能够指定处理函数或重新制定分配器	无法通过用户代码进行处理
函数重载	允许	不允许
构造函数与析构函数	调用	不调用

new分配内存到**自由存储区**，不仅可以是堆，还可以是静态存储区，这都看operator new在哪里为对象分配内存

9.面向对象和面向过程的区别

面向过程是一种以**过程**为中心的编程思想，它首先分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，在使用时依次调用，是一种基础的顺序的思维方式。

面向对象是按人们认识客观世界的系统思维方式，采用基于对象（实体）的概念建立模型，模拟客观世界分析、设计、实现软件的编程思想，通过面向对象的理念使计算机软件系统能与现实世界中的系统一一对应。

10.sizeof 与 strlen的区别

sizeof 是运算符，strlen 是函数。

sizeof 可以用类型做参数，**strlen** 只能用 **char*** 做参数，且必须是以 **\0** 结尾的

数组做 **sizeof** 的参数不退化，传递给 **strlen** 就退化为指针了

大部分编译程序在编译的时候就把 **sizeof** 计算过了，是类型或是变量的长度，这就是 **sizeof(x)** 可以用来定义数组维数的原因；strlen 的结果要在运行的时候才能计算出来，是用来计算字符串的长度，不是类型占内存的大小。

```
char str[20]="0123456789";
int a=strlen(str); // a=10;
int b=sizeof(str); // 而 b=20;
```