1. **进程和线程的区别**
2. **拷贝构造函数为什么传引用**

不传引用而进行传值传参会造成无穷递归的情况。

函数传参，实际上是需要拷贝一份传给函数作为形参。

C++中规定，函数传参传递自定义类型的时候，自定义类型的拷贝会自动调用拷贝构造来完成。如果实现的拷贝构造是传值传参，传参的时候自定义类型需要拷贝会自动调用拷贝构造。

为什么C++要规定自定义类型拷贝的时候要自动调用拷贝构造？

实际上C++规定是这样：

* + - 内置类型会直接拷贝
    - 对于自定义类型会调用他的拷贝构造。

**2、C++中new和malloc的区别**

new操作符从自由存储区为对象动态分配内存空间，malloc函数从堆上动态分配内存。

自由存储区是C++基于new操作符的一个抽象概念，凡是通过new操作符进行内存申请，该内存即为自由存储区。而堆是操作系统中的术语，是操作系统所维护的一块特殊内存，用于程序的内存动态分配，C语言使用malloc从堆上分配内存，使用free释放已分配的对应内存。

new操作符内存分配成功时，返回的是对象类型的指针，无需进行类型转换，符合类型安全性的操作符。

malloc内存分配成功返回void\*，需要通过强制类型转换把void\*转换成需要的类型。

new内存分配失败时，会抛出bac\_alloc异常，不会返回NULL。

malloc分配内存失败时返回NULL。

new由编译器根据类型计算得出分配内存的大小。

malloc必须显示指定字节数。

new可以函数重载，调用构造函数和析构函数。

**3、new的内存不够了，操作系统会做什么操作**

C++提供两种报告方式：抛出bac\_alloc异常 或者 返回空指针而不抛出异常

**4、引用和指针的区别**

**5、多态**

编译时多态：函数重载

运行时多态，通过虚函数表实现

纯虚函数，抽象接口类，不能实例化

**6、父类指针指向子类对象，如何对子类虚函数进行寻址**

**7、构造函数可以是虚函数吗**

不可以

C++对象模型构造对象第一个就指向虚表的指针。

虚函数的调用是通过虚函数表来查找的，而虚函数表由类的实例化对象的vptr指针指向，该指针放在对象的内部空间中，需要调用构造函数完成初始化。如果构造函数是虚函数，那么调用构造函数就要去找vptr，但是此时vptr还没有初始化

**8、C++怎么调用C**

extern C

**9、脏读、幻读、不可重复读的概念**

10、**B+树、红黑树、二叉树**

11、**虚拟内存**

MMU

12、**左值和右值，什么时候使用右值，右值引用，移动语义，深拷贝，浅拷贝**

左值：一般是可以修改的值，可以取地址的，通常是变量。

右值：一般是常量(除const修饰的)，表达式或者函数的传值返回(生成临时变量)。

传引用返回是右值。

**引用**：只能引用左值，不能引用右值，const引用既可以引用左值，又可以引用右值。

int& a = 10; 错误

const int& a = 10; 正确

**右值引用**：类型+&&，只能引用右值，不能引用左值。但是右值引用可以引用move之后的左值。

int&& a = 10; 正确

int&& b = a; 错误

int&& b = std::move(a); 正确

引用做参数和返回值可以减少拷贝次数，特别是对于深拷贝，可以提高效率，但是返回值是函数的局部对象时不能引用返回，需要传值返回。

String Fun(String& s) {

String ret(s); //调用拷贝构造

return ret;

}

String s1(“左值”);

String s2 = Fun(s1);

ret在按照值返回时，必须拷贝构造一个临时对象，需要进行深拷贝。将Fun函数返回值赋值给s2时，也就是将临时对象拷贝构造s2，也需要进行深拷贝。其实s2和临时对象空间里内容相同。创建了三个完全相同的对象，空间浪费，效率也降低了。

**移动语义：**将一个对象中的资源移动到另一个对象中。

临时对象是一个将亡值，也就是右值，可以重载一个参数为右值引用的拷贝构造函数。这个拷贝构造函数成为**移动构造函数**

String(String&& s) {

\_str = s.str;

s.\_str = nullptr;

}

String s1(“左值”);

String s2(Fun(s1)); //调用移动构造

ret是左值，在拷贝构造临时对象时，调用了拷贝构造函数，而临时对象是右值(将亡值)。构造s2时，调用移动构造，将临时对象的资源换给了s2，减少了一次深拷贝过程。相当于s2指向了临时对象指向的一块内存地址，临时对象也不再指向这片内存，所以临时对象生命周期结束销毁时，不会释放这片内存。

C++11中，编译器会为类默认生成一个移动构造，但是浅拷贝。因此涉及资源管理时，必须显示自己的移动构造。

如果没有右值引用，用对象接收，会调用拷贝构造，对应string/vector等容器，需要进行深拷贝，效率低。

**右值引用主要应用就是重载了移动构造函数**，利用将亡值，将将亡值里的空间交换到拷贝的对象中，**减少深拷贝**。

std::move主要用于将左值强行转换为右值，被转化的左值生命周期并没有因为这种转换和改变。在使用std::move时需要注意：一旦该左值被转换为右值，如果和移动语义结合使用，那么该左值的生命周期就将结束，如果此后还继续使用该左值，会出现严重错误。

{

Class A;

Class B(std::move(A));

cout << \*A.pointer <<endl; //A由于移动语义，其对内存实际被释放，后续调用会报错

}

确保std::move用于移动语义的变量是一个临时量。

std::move与移动语义结合时，

**完美转发：**在函数模板中，完全依照模板的参数的类型，将参数传递给函数模板中调用的另一个函数。由于拷贝问题存在，所以完美转发一般不包括值传递。

std::move：移除左值属性

std::forward：完美转发

13、**智能指针，shared\_ptr和unique\_ptr区别，性能对比，weak\_ptr及其作用**

智能指针线程安全

14、**lambda表达式，有哪些捕获类型**

匿名函数

[ ](int a, int b) {

return a + b;

}

捕获列表：

[ ] 不捕获任何变量

[var] 捕获变量var

[=] 以值传递的方式捕获所有父作用域的变量，包括this

[&var] 引用传递方式捕获变量var

[&] 引用传递方式捕获所有父作用域的变量，包括this

[this] 值传递方式捕捉当前的this指针

[=, &] 拷贝与引用混合

[=, &a, &b] 引用传递方式捕获变量a和b，以值传递方式捕获其他所有变量

**捕获列表不允许变量重复传递**

参数列表：

类似于函数参数列表，(int var)可选，( )可不选。

可变规格mutable：

Mutable修饰符，默认情况下lambda函数总是一个const函数，mutable可以取消其常量性，在使用该修饰符时，参数列表不可以省略。使用mutable表示可以修改按值捕获的副本，并不修改原来的值

异常说明：

可以使用throw()异常规范来指示lambda表达式不会引发任何异常。

[ ]() throw() { throw 5; }();

返回类型：

Lambda表达式的返回类型会自动推导。

15．**std::function**

std::function是一个通用的多台函数包装器，std::function的实例可以存储、复制和调用任何可调用的目标:包括函数、lambda表达式、绑定表达式或其他函数对象、以及指向成员函数和指向数据成员的指针。当std::function对象未包裹任何实际的可调用元素，调用std::function对象将抛出std::bad\_function\_call异常。

std::function<void()> func

例：std::function<void()> func = 《lambda表达式》

通过std::bind函数调用类成员函数

std::function<void()> func = std::bind(&Class::func, this);

**std::bind:**可以将可调用对象和参数一起绑定，绑定后的结果使用std::function进行保存，并延迟调用到任何我们需要的时候。bind绑定成员函数需要传入对象指针或this指针，C++对象模型限制成员函数的地址是一个局部地址。

std::placeholders::\_1表示一个占位符。

**16．explicit、default、delete关键字**

explicit关键字的作用防止类构造函数的隐式自动转换，告诉编译器按照实际的类型来构造函数，不允许做隐式转换

显示构造函数，声明单参数构造函数。

**17．std::unique\_lock**

**18．std::thread、多线程**

**19．try…catch(…) throw()异常捕获**

**20. C++11新特性**

1、自动类型推导(auto)：允许编译器推导变量的类型。

2、范围-based for循环：简化对容器元素的遍历。

3、智能指针：引入std::shared\_ptr和std::unique\_ptr等智能指针，用于管理动态分配的内存，帮助防止内存泄漏。

4、lambda表达式：允许在函数内部定义匿名函数，提高代码可读性和灵活性。

5、nullptr：引入空指针常量nullptr，用于替代传统的空指针NULL。

6、强制类型转换(Type Casting)：引入了static\_cast、dynamic\_cast、const\_cast、reinterpret\_cast等更安全和灵活的类型转换操作符。

7、右值引用和移动语义：支持通过右值引用实现移动语义，提高了对临时对象的处理效率。

8、新的容器和算法：引用新的容器，如std::unordered\_map、std::unordered\_set，以及一些新的算法。

9、线程支持：std::thread，提供了原生的多线程支持，使得并发编程更加方便。

**21. 范围-based for循环**

也被称为foreach循环，C++11引入的一种简化遍历容器元素的语法结构。它提供了一种更简洁、易读的方式来遍历容器中的元素，特别适用于需要遍历整个容器的情况。

语法格式：

for(const auto& element : container) {

//循环体，使用element访问容器中的元素

}

语法结构详解：

* + const auto& element：const表示循环变量是只读的，auto表示编译器会根据初始化表达式的类型自动推导循环变量的类型。&表示引用。使得在循环体中可以通过element访问容器中的元素，而且element是对元素的常量引用，因此不能在循环中修改元素的值。
  + **:** ：冒号表示这是范围-based for循环的语法结构，用于遍历容器中的元素。
  + Container：要遍历的容器，可以是数组、std::vector、std::list等。

优点：

* 简洁性：语法相对简单，减少了书写错误的可能性，提高了代码的可读性。
* 安全性：使用const auto&可以避免在循环中对元素进行意外修改。
* 遍历全容器：范围-based for循环天生适合遍历整个容器，无需手动处理索引和迭代器。

**22. C++异常类**

**23. STL vector push\_back过程**

push\_back(a)分两种情况：

* 当a定义有拷贝构造函数时：调用a的拷贝构造函数，如果还有成员变量没有被初始化，那么会继续调用成员变量的无参构造器，例如int类型会为0，指针为空
* 没有定义拷贝构造函数时：执行浅拷贝。但如果成员变量声明过拷贝构造函数，那么会自动调用成员变量的拷贝构造函数。指针类型的拷贝就是多出一个指针指向同一个区域。这意味着如果push进了元素是指针或者该元素含有指针变量且析构函数会释放所指空间，那么pop出来，前者会导致a成为野指针；后者如果在析构函数之后没有将释放空间后的指针指向NULL，也会导致野指针的出现。