**C++11新特性**

1. 原始字面量

描述：如果想让字符串中的一些特殊字符不转义，或者让字符串在多行中书写，就用原始字面量。

形式：R”(hello\world)”; 或者 R”备注(hello\world)备注”; 备注前后必须一样，不然会报错

1. constexpr关键字

const有两种语义，一是让一个变量只读，而是修饰常量。常量就是编译的时候就已经知道结果了的。而constexpr就是专门用来修饰常量的

1. auto自动类型推导

比如：auto a = 3;这里auto推导出来就为int

注意点：使用auto必须初始化。auto是交给编译器去推导，不是运行时推导。当变量不是指针或者引用类型时，推导出的结果不会保留const和volatile关键字。当变量是指针或者引用类型时，推导出的结果会保留const和volatile关键字。

1. decltype

使用：decltype(表达式)

1. 返回值类型后置

返回值类型后置需要auto和decltype组合起来使用。比如说我们调用者不知道函数模板的返回值类型的时候就可以使用，如下例子

template <typename T, typename U>

auto add(T t, U u) -> decltype(t+u)

{

return t+u;

}

这里时用decltype推导出来的结果给atuo初始化

1. final关键字

用来修饰类和函数，表示类不能被继承和函数不能被重写。只能修饰虚函数，因为只有虚函数才能被重写。示例：class Test final{ } void fun() final{ }

1. override关键字

在重写父类的虚函数的时候加上override关键字，这样假如我们函数名拼写错误编译器就能报错了。示例：void fun() override{ }

1. using关键字

using关键字有三个作用：一、声明要使用的命名空间。比如：using namespace std;

二、类型取别名，作用跟typedef完全一样。

比如：using myint = int; using func = void(\*)(int, double);

三、在子类中使用父类被隐藏（覆盖）的函数。比如：using Father::fun;这样使用fun方法的时候就是父类的fun了，而不是子类的fun

1. 委托构造函数

就是构造函数能调用构造函数，防止代码的冗余。形式test(int a, int b, intc) :test(a,b){ }调用另一个构造函数的时候尽量写在初始化列表里，写在函数体里有时候会报错。

1. 继承构造函数

如果想要在子类创建对象的时候使用父类的构造函数，那就需要using Father::Father;将父类的构造函数显示出来。

1. 初始化列表{ }

C++的一种统一的初始化方式，对所有的类型都适用。加不加等号都一样

比如：int a = {3}; int a{3}; int\* p = new int{30};

1. 无论是左值还是右值都是别名

左值是能取地址的，右值是不能取地址的。右值有两种，一种是纯右值，比如说数字，常量字符串，另一种是将亡值。

1. 对象资源拷贝的方式有三种

移动构造还是不能全部转移，只能转移堆内存资源

一：用深拷贝构造Test(const Test& ts)

二：用移动构造Test(Test&& ts) Test b = func( ) func返回一个对象

三：用右值引用管理将亡的资源 Test&& b = func( ) func返回一个对象

1. 不是所有情况下&&都表示右值引用

T&&和auto&&表示未定义的引用类型，如果用右值去推导，最终是推导出来是右值引用，用其他的推导都是左值引用。const T&&和const auto&&表示右值引用。右值引用传递就会被编译器看成是左值，比如int&& a = 5; int&& b = a;这样就会报错，因为int&& b = a;这条语句的a被编译器看成了左值。

**智能指针#include <memory>**

**共享的智能指针**

初始化方式：

shared\_ptr<int> ptr(new int); 在构造函数里直接new一块内存

shared\_ptr<int> ptr(ptr); 用另一个共享的智能指针去初始化

shared\_ptr<int> ptr = ptr1; 用拷贝构造初始化，引用计数为二

shared\_ptr<int> ptr = move(ptr1); 用移动构造初始化，转移资源，引用计数为一

shared\_ptr<int> ptr1(move(ptr))；

make\_shared<类型>(初始化的值)；用make\_shared初始化

reset();引用计数减一，指针置nullptr，如果减到了零就释放空间

reset(new int);重新管理另外一块空间，把原来的空间的引用计数减一，如果到零就释放

get();获取原始指针

use\_count();查看引用计数

**共享智能指针的删除器**

如果智能指针管理的是一块数组内存，那么它智能释放一个元素，不能全部释放，因此我们要指定删除器来释放全部的内存。

shared\_ptr<int> ptr(new int(3), 删除器函数的地址); 第二个参数可以是普通函数也可以是lambda表达式。例如：

shared\_ptr<int> ptr(new int[5], [](int\* t) { delete[ ] t; });管理的指针会传给删除器，所以lambda表达式的参数是管理的指针类型，t指向管理的那块空间

默认删除器：（不需要自己写了，通用）

shared\_ptr<int> ptr(new int[5], default\_delete<int[5]>( ) );模板参数列表里是管理的空间类型。

**智能指针怎么使用**

1. 取出原始地址来使用
2. 直接使用智能指针对象，用法跟指针一样

**共享的智能指针是一块内存可以由多个智能指针对象管理，独占的智能指针是一块内存只能由一个智能指针对象管理**

**独占的智能指针**

初始化方式：

unique\_ptr<int> ptr(new int); 在构造函数里直接new一块内存

unique \_ptr<int> ptr = move(ptr1); 用移动构造初始化，转移资源

unique\_ptr<int> ptr1(move(ptr));

make\_shared<类型>(初始化的值)；用make\_shared初始化

reset();引用计数减一，指针置nullptr，释放空间就

reset(new int);重新管理另外一块空间，将原来的空间释放

get();获取原始指针

**独占的智能指针的删除器**

unique\_ptr<int, 函数的类型> ptr(new int(3), 删除器函数的地址); 第二个参数可以是普通函数也可以是lambda表达式。例如：

using funptr = void(\*)(int\* t);

unique\_ptr<int, funptr> ptr(new int[5], [](int\* t) { delete[ ] t; });管理的指针会传给删除器，所以lambda表达式的参数是管理的指针类型，t指向管理的那块空间

lambda表达式不捕获变量的时候可以看成是函数，捕获变量的时候是仿函数，所以捕获变量的时候要这样写：

unique\_ptr<int, function<void(int\*)>> ptr(new int[5], [=](int\* t) { delete[ ] t; });

C++11以后共享的智能指针和独占的智能指针都能删除数组了，只要把类型指定成数组，例如：

shared\_ptr<int[5]> ptr(new int[5]);

unique\_ptr<int[5]> ptr(new int[5]);

**weak\_ptr**

weak\_ptr是辅助共享智能指针的，因为共享智能指针可能会出现下面这些问题。

共享智能指针使用注意事项：

1. 不能使用同一个原始地址初始化多个共享智能指针，因为用原始地址初始化的智能指针引用计数都是一。
2. 共享智能指针不能循环引用，因为智能指针要先释放管理的内存再释放自己。