20220409-C++

1.过程描述

作用域–栈和堆 C++ 复制代码

```
这种写法是错误的，因为这是在栈上分配的内存，一旦函数结束，内存就被释放了
int* CreateArray()
{
    int Array[50];
    return Array;
}

int  main() {
    int* a = CreateArray();
}
-----------------------------
在heap上分配是OK的
int* CreateArray()
{
    int* Array=new int[50];
    return Array;
}
```

智能指针示例 C++ 复制代码

```cpp
class ScopePtr
{
private:
    Entity* m_Ptr;
public:
    ScopePtr(Entity* ptr)
        :m_Ptr(ptr)
    {

    }
    ~ScopePtr()
    {
        delete m_Ptr;
    }
};

//不用delete也能释放内存
int  main() {
    {
        ScopePtr e = new Entity();
    }
    cin.get();
}
-------------------------------------------------
#include <memory>
int  main() {
    {
        //unique智能指针，当这个scope结束时，也就自动delete了
        unique_ptr<Entity> entity;
        unique_ptr<Entity> entity(new Entity());
        unique_ptr<Entity> entity = make_unique<Entity>();//safer
    }

    {
        shared_ptr<Entity> e0;
        {
            shared_ptr<Entity> sharedEntity = make_shared<Entity>();
            e0 = sharedEntity;//unique_ptr不能这么搞.would increase ref count
            weak_ptr<Entity> weakEntity = sharedEntity;//would not increase ref count
        }
    }
    cin.get();
}
```

复制构造函数 C++ 复制代码

```cpp
class String
{
private:
    char* m_buffer;
    unsigned int m_Size;
public:
    String(const char* string)
    {
        m_Size = strlen(string);
        m_buffer = new char[m_Size+1];
        memcpy(m_buffer, string, m_Size);
        m_buffer[m_Size] = 0;
    }

    String(const String& other)
        :m_Size(other.m_Size)
    {
        cout << "copied" << endl;
        m_buffer = new char[m_Size + 1];
        memcpy(m_buffer, other.m_buffer, m_Size + 1);
    }

    ~String()
    {
        delete[] m_buffer;
    }

    char& operator[](unsigned int index)
    {
        return m_buffer[index];
    }

    friend ostream& operator<<(ostream& stream, const String& string);
};

ostream& operator<<(ostream& stream, const String& string)
{
    stream << string.m_buffer;
    return stream;
}

void Printstring(String string)
{
    cout << string << endl;
}
```

```
46
47  void Printstring2(const String& string)//always pass your object by const
    reference
48  {
49      cout << string << endl;
50  }
51
52  int  main() {
53      String string = "Cherno";
54      String second = string;//如果没有定义复制构造函数，指针指向与string相同的内
    存，一旦delete就会出错（不能delete两次
55      second[2] = 'a';//如果没有定义复制构造函数，都会变成charno
56      Printstring(string);
57      Printstring(second);
58      /*结果
59      copied
60      copied
61        Cherno
62      copied
63        Charno
64      */
65      Printstring2(string);
66      Printstring2(second);
67      /*结果
68      copied
69        Cherno
70        Charno
71      */
72
73      cin.get();
74  }
```

动态数组vector C++ 复制代码

```cpp
struct Vertex
{
    float x, y, z;
};

ostream& operator<<(ostream& stream, const Vertex& vertex)
{
    stream << vertex.x << "," << vertex.y << "," << vertex.z;
    return stream;
}

void Func(const vector<Vertex>& vertices) //注意引用符号的位置
{
}

int main()
{
    // Vertex* vertices = new Vertex[5];static array
    vector<Vertex> vertices;
    vertices.push_back({ 1,2,3 });
    vertices.push_back({ 4,5,6 });

    for (int i = 0; i < vertices.size(); i++)
        cout << vertices[i] << endl;
    for (Vertex& v : vertices)
        cout << v << endl;

    vertices.erase(vertices.begin() + 1);
    for (Vertex& v : vertices)
        cout << v << endl;
}
-------------------------------------------------
vector性能优化
//vertices.push_back(Vertex(1, 2, 3));
//vertices.push_back(Vertex(4, 5, 6));
//vertices.push_back(Vertex(7, 8, 9));会有六个copied

vertices.reserve(3);
vertices.emplace_back(1, 2, 3);
vertices.emplace_back(4, 5, 6);
vertices.emplace_back(7, 8, 9);//0个copy
```

local static C++ 复制代码

```C++
class Singleton
{
public:
    static Singleton& Get() //static extend the lifetime
    {
        static Singleton instance;
        return instance;
    }
    void Hello(){}
};

int main()
{
    Singleton::Get().Hello();
}
```

2.结果输出

今天依旧只看了The cherno的C++视频，效率不高，感觉封在寝室里整个人都比较废。后面加油吧。