77. move semantics(转移语义)(C++11)

通过用移动语义，可以用移动语义来替代复制，代码表现会好很多。

1. 一般的拷贝

```
class String
{
public:
   String() = default;
   String(const char* string)
   {
      printf("Created!\n");
      m_Size = strlen(string);
      m_Data = new char[m_Size];
      memcpy(m_Data, string, m_Size);
   }
   //copy constructor
   String(const String& other)
   {
      printf("Copy!\n");
      m_Size = other.m_Size;
      m_Data = new char[m_Size];
      memcpy(m_Data, other.m_Data, m_Size);
   }
   ~String()
   {
      printf("Destroyed!\n");
      delete m_Data;
   }
   void Print()
   {
      for (uint32_t i = 0; i < m_Size; i++)
      {
         printf("%c", m_Data[i]);
      }
      printf("\n");
   }
private:
   char* m_Data;
   uint32_t m_Size;
};

class Entity
{
public:
   Entity(const String& name)
      :m_Name(name)
   {
   }
   void PrintName()
   {
      m_Name.Print();
   }
   ~Entity()
```

```
    {
    }

private:
    String m_Name;
};

int main()
{
    Entity entity("EDED");
   /*
   Created!
   Copied!
   */
    entity.PrintName();
}

```

78. std::move and Move Assignment Operator(移动赋值运算符) in C++

- `std::move`:

```
   String string = "Hello";
   //String dest((String&&)string);
   String dest(std::move(string));
   String dest = std::move(string);
```

- Move Assignment Operator :

  将一个变量移动到另一个已经存在的变量，称之为移动赋值。

  需要重载等号运算符:

```
String& operator=(String&& other) noexcept
{
    printf("Moved!\n");
        //添加判断，防止出现dest = std::move(dest);
     //这样的语句出现错误
        if (this != &other)
        {
        //要给当前的对象赋新值的话，需要释放掉原来的内存
            delete[] m_Data;
            m_Size = other.m_Size;
            m_Data = other.m_Data;
            other.m_Size = 0;
            other.m_Data = nullptr;
        }
        return *this;
}
```

79. ARRAY--Making data structures

内存放在栈上的数组数据结构

复习`std::array`

```
  std::array<int, 10> collection;
  for (int i = 0; i < collection.size(); i++)
        collection[i] = collection.size() - i;
  //这样的写法只能是在有迭代器的时候写
  for (int i : collection)
  {
     std::cout << i << std::endl;
  }
```

自己写一个`Array`:

```
template<typename T, size_t S>
class Array
{
public:

  //不占用额外的空间来存储数组的长度，非常amazing
   constexpr int Size() const { return S; }

   T& operator[](size_t index)
   {
      if (!(index< S))
      {
         __debugbreak();
      }
      return m_Data[index];
   }
   const T& operator[](size_t index) const
   {
      if (!(index < S))
      {
         __debugbreak();
      }
      return m_Data[index];
   }

   T* Data(){  return m_Data; }
   const T* Data() const { return m_Data;}
private:
   T m_Data[S];
};
```

80. Vector: dynamic array--Making data structures

81. Iterarators

- 用来遍历容器中的元素

```
std::vector<int> values = {1,2,3,4,5};

//1.
for(int i =0; i < values.size(); i++)
  std::cout<<values[i] << std::endl;

//2.
for(int value: values)
  std::cout<<value<<std::endl;

//3.
for(std::vector<int>::iterator it = values.begin();
    it != values.end(); it++)
  std::cout<<*it<<std::endl;
```

一些场景必须要用到iterator:

```
std::unorder_map<std::string, int> map;

using ScoreMap =
  std::unorder_map<std::string, int>;
using ScoreMapConstIter = ScoreMap::const_iterator;
map["dede"] = 2;
map["2dd"] = 23;

for(ScoreMapConstIter it = map.begin();
    it!=map.end();it++)
{
   auto& key = it->first;
  auto& value = it->second;
  std::cout<<key<<"="<<value<<std::endl;
}

for(auto kv : map)
{
      auto& key = kv.first;
  auto& value = kv.second;
  std::cout<<key<<"="<<value<<std::endl;
}

//c++17
for(auto [key, value] : map)
  std::cout<<key<<"="<<value<<std::endl;
```

82. Writing an Iterators in c++