90 std--move and the Move Assignment Operator in C++

经过上一节89 Move Semantics in C++，已经基本了解移动语义的要点和所有基本知识了。移动语义能够将一个对象移动到另一个对象上，但是还没有涉及到其中的两个关键部分：`std::move`和 *move assignment oprator*（移动赋值运算符），这是一个赋值操作符，当我们想把一个对象移动到一个已有的对象中时（不是构造一个新对象）。

1. std::move

上节的代码中，这里用`std::move`可以让我们把这个临时变量name，转移到m_Name这个可以永久居住的地方（类成员），

```C++
    Entity(String&& name)
    //  :m_Name(name) {}
        :m_Name(std::move(name)) {}
```

新建一个String：

```C++
String string = "Hello";
String dest = string; // 复制字符串到新变量中，并非移动
```

想要移动的话，显然需要Entity的移动构造函数，为了使用它我们需要确保传入的字符串变为**临时的**

```C++
String string = "Hello";
//  String dest = (String&&)string;
//  String dest((String&&)string);
// 这并非最优雅且对每个类型都使用的方法。因此可以使用一个灵活一些的函数，可以在编译时用auto找出输入的类型
String dest(std::move (string));
```

```
template <class _Ty>  <T> Provide sample template arguments for IntelliSense
_NODISCARD constexpr remove_reference_t<_Ty>&& move(_Ty&& _Arg) noexcept { //
  forward _Arg as movable
    return static_cast<remove_reference_t<_Ty>&&>(_Arg);
}
```

> 可以看到它返回一个右值引用类型，是以一种很好的模板化的方式来实现的，可以正确处理所有类型，包括常量等。

这里哪种方法都是创建一个新对象，因此会用移动构造函数，这就引出了**移动赋值运算符。**

2. 移动赋值运算符

赋值操作符，只有当我们把一个变量赋值给一个已有变量时才会被调用，比如：

```C++
String dest(std::move (string));
dest = std::move(string);
```

> 运算符实际上就像一个函数，所以这里调用=运算符时，就像是你有一个assign函数dest.assgin(std::move(string))、

移动赋值运算符长得很像移动构造函数：

```C++
// 移动赋值操作符
String& operator=(String&& other) noexcept
{
    printf("Moved!\n");

    // 检查self-assignment（自我赋值），确保不是将对象赋值给自身
    if (this != &other)
    {
        // 删除当前对象持有的资源
        delete[] m_Data; // **重点** 因为要覆盖原对象，删除内存防止内存泄漏

        // 将"other"对象的资源"窃取"到当前对象
        m_Size = other.m_Size;
        m_Data = other.m_Data;

        // 使"other"对象进入一个有效但未定义的状态
        other.m_Size = 0;
        other.m_Data = nullptr;
    }

    // 返回*this以支持链式赋值
    return *this;
}
```

调用测试：

```cpp
int main()
{
    String apple = "Apple";
    String dest;

    std::cout << "Apple: ";
    apple.Print();
    std::cout << "Dest: ";
    dest.Print();

    dest = std::move(apple);

    std::cout << "Apple: ";
    apple.Print();
    std::cout << "Dest: ";
    dest.Print();

    std::cin.get();
}
```

> 可以看到apple的资源被dest“偷走”了，我们基本上转移了整个字符数组的所有权，没有做任何复制或解除分配之类的事情。

总而言之，移动赋值操作符是你想要包含在类中的东西，当你包含一个移动构造函数时，因为可能会想要将一个对象移动到一个现有变量中。它基本上是**五法则**的一部分，五法则包含了新移动语义

> C++三法则：如果需要析构函数，则一定需要拷贝构造函数和拷贝赋值操作符；
> C++五法则：为了支持移动语义，又增加了移动构造函数和移动赋值运算符。

而`std::move`是你想要将一个对象转换为临时对象时要做的，换句话说如果你需要把一个已经存在的变量变为临时变量，你可以标记它，表示你可以从这个特定的变量中窃取资源，这使我们能够在现有的变量上执行移动通过操作。

要注意的是，`std::move`本身并不执行任何移动操作；它仅仅重新解释对象，使其可以被当作右值使用。实际的移动行为是由特定的移动构造函数或移动赋值操作符执行的，这些函数在接收到标记为右值的对象时被调用。"