

Конструкторы move и операторы присваивания move (C++)

05.03.2018 • Время чтения: 4 мин • 🌐 🔄 📄

В этой статье

[Пример](#)

[Пример](#)

[Отказоустойчивость](#)

[См. также](#)

В этом разделе описывается создание *конструктор перемещения* и оператор присваивания перемещения для класса C++. Конструктор перемещения позволяет ресурсы, принадлежащие объекте rvalue должна быть перемещена в другую lvalue без копирования. Дополнительные сведения о семантике перемещения см. в разделе [декларатор ссылки Rvalue: & &](#).

Этот раздел построен на основе приведенного ниже класса C++ MemoryBlock, который управляет буфером памяти.

C++

Копировать

```
// MemoryBlock.h
#pragma once
#include <iostream>
#include <algorithm>

class MemoryBlock
{
public:

    // Simple constructor that initializes the resource.
    explicit MemoryBlock(size_t length)
        : _length(length)
        , _data(new int[length])
    {
        std::cout << "In MemoryBlock(size_t). length = "
                    << _length << "." << std::endl;
    }

    // Destructor.
    ~MemoryBlock()
    {
        std::cout << "In ~MemoryBlock(). length = "
                    << _length << ".";

        if (_data != nullptr)
        {
```

```
        std::cout << " Deleting resource.";
        // Delete the resource.
        delete[] _data;
    }

    std::cout << std::endl;
}

// Copy constructor.
MemoryBlock(const MemoryBlock& other)
    : _length(other._length)
    , _data(new int[other._length])
{
    std::cout << "In MemoryBlock(const MemoryBlock&). length = "
                << other._length << ". Copying resource." << std::endl;

    std::copy(other._data, other._data + _length, _data);
}

// Copy assignment operator.
MemoryBlock& operator=(const MemoryBlock& other)
{
    std::cout << "In operator=(const MemoryBlock&). length = "
                << other._length << ". Copying resource." << std::endl;

    if (this != &other)
    {
        // Free the existing resource.
        delete[] _data;

        _length = other._length;
        _data = new int[_length];
        std::copy(other._data, other._data + _length, _data);
    }
    return *this;
}

// Retrieves the length of the data resource.
size_t Length() const
{
    return _length;
}

private:
    size_t _length; // The length of the resource.
    int* _data; // The resource.
};
```

В следующих процедурах описывается создание конструктора перемещения и оператора присваивания перемещения для этого примера класса C++.

Создание конструктора перемещения для класса C++

1. Определите пустой метод конструктора, принимающий в качестве параметра ссылку rvalue на тип класса, как показано в следующем примере:

C++	Копировать
<pre>MemoryBlock(MemoryBlock&& other) : _data(nullptr) , _length(0) { }</pre>	

2. В конструкторе перемещения присвойте создаваемому объекту данные-члены класса из исходного объекта:

C++	Копировать
<pre>_data = other._data; _length = other._length;</pre>	

3. Присвойте данным-членам исходного объекта значения по умолчанию. Это не позволяет деструктору многократно освобождать ресурсы (например, память):

C++	Копировать
<pre>other._data = nullptr; other._length = 0;</pre>	

Создание оператора присваивания перемещения для класса C++

1. Определите пустой оператор присваивания, принимающий в качестве параметра ссылку rvalue на тип класса и возвращающий ссылку на тип класса, как показано в следующем примере:

C++	Копировать
<pre>MemoryBlock& operator=(MemoryBlock&& other) { }</pre>	

2. В операторе присваивания перемещения добавьте условный оператор, который не выполняет никакой операции при попытке присвоить объект самому себе.

C++	Копировать
<pre>if (this != &other) { }</pre>	

```
}

```

3. В условном операторе освободите все ресурсы (такие как память) из объекта, которому производится присваивание.

В следующем примере освобождается член `_data` из объекта, которому производится присваивание:

C++	Копировать
<pre>// Free the existing resource. delete[] _data; </pre>	

Выполните шаги 2 и 3 из первой процедуры, чтобы переместить данные-члены из исходного объекта в создаваемый объект:

C++	Копировать
<pre>// Copy the data pointer and its length from the // source object. _data = other._data; _length = other._length; // Release the data pointer from the source object so that // the destructor does not free the memory multiple times. other._data = nullptr; other._length = 0; </pre>	

4. Верните ссылку на текущий объект, как показано в следующем примере:

C++	Копировать
<pre>return *this; </pre>	

Пример

В следующем примере показаны полные конструктор перемещения и оператор назначения перемещения для класса `MemoryBlock`:

C++	Копировать
<pre>// Move constructor. MemoryBlock(MemoryBlock&& other) : _data(nullptr) , _length(0) { std::cout << "In MemoryBlock(MemoryBlock&&). length = " </pre>	

```
        << other._length << ". Moving resource." << std::endl;

// Copy the data pointer and its length from the
// source object.
_data = other._data;
_length = other._length;

// Release the data pointer from the source object so that
// the destructor does not free the memory multiple times.
other._data = nullptr;
other._length = 0;
}

// Move assignment operator.
MemoryBlock& operator=(MemoryBlock&& other)
{
    std::cout << "In operator=(MemoryBlock&&). length = "
               << other._length << "." << std::endl;


    if (this != &other)
    {
        // Free the existing resource.
        delete[] _data;

        // Copy the data pointer and its length from the
        // source object.
        _data = other._data;
        _length = other._length;

        // Release the data pointer from the source object so that
        // the destructor does not free the memory multiple times.
        other._data = nullptr;
        other._length = 0;
    }
    return *this;
}
```

Пример

В следующем примере показано, как семантика перемещения может повысить производительность приложений. В примере добавляются два элемента в объект-вектор, а затем вставляется новый элемент между двумя существующими элементами. `vector` Класс использует семантику перемещения для эффективного выполнения операции вставки путем перемещения элементов вектора вместо их копирования.

C++	 Копировать
<pre>// rvalue-references-move-semantics.cpp // compile with: /EHsc</pre>	

```
#include "MemoryBlock.h"
#include <vector>

using namespace std;

int main()
{
    // Create a vector object and add a few elements to it.
    vector<MemoryBlock> v;
    v.push_back(MemoryBlock(25));
    v.push_back(MemoryBlock(75));

    // Insert a new element into the second position of the vector.
    v.insert(v.begin() + 1, MemoryBlock(50));
}
```

В этом примере выводятся следующие данные:

Output

Копировать

In MemoryBlock(size_t). length = 25.
In MemoryBlock(MemoryBlock&&). length = 25. Moving resource.
In ~MemoryBlock(). length = 0.
In MemoryBlock(size_t). length = 75.
In MemoryBlock(MemoryBlock&&). length = 25. Moving resource.
In ~MemoryBlock(). length = 0.
In MemoryBlock(MemoryBlock&&). length = 75. Moving resource.
In ~MemoryBlock(). length = 0.
In MemoryBlock(size_t). length = 50.
In MemoryBlock(MemoryBlock&&). length = 50. Moving resource.
In MemoryBlock(MemoryBlock&&). length = 50. Moving resource.
In operator=(MemoryBlock&&). length = 75.
In operator=(MemoryBlock&&). length = 50.
In ~MemoryBlock(). length = 0.
In ~MemoryBlock(). length = 0.
In ~MemoryBlock(). length = 25. Deleting resource.
In ~MemoryBlock(). length = 50. Deleting resource.
In ~MemoryBlock(). length = 75. Deleting resource.

До выхода Visual Studio 2010 этот пример формирует следующие выходные данные:

Output

Копировать

In MemoryBlock(size_t). length = 25.
In MemoryBlock(const MemoryBlock&). length = 25. Copying resource.
In ~MemoryBlock(). length = 25. Deleting resource.
In MemoryBlock(size_t). length = 75.
In MemoryBlock(const MemoryBlock&). length = 25. Copying resource.

```
In ~MemoryBlock(). length = 25. Deleting resource.
In MemoryBlock(const MemoryBlock&). length = 75. Copying resource.
In ~MemoryBlock(). length = 75. Deleting resource.
In MemoryBlock(size_t). length = 50.
In MemoryBlock(const MemoryBlock&). length = 50. Copying resource.
In MemoryBlock(const MemoryBlock&). length = 50. Copying resource.
In operator=(const MemoryBlock&). length = 75. Copying resource.
In operator=(const MemoryBlock&). length = 50. Copying resource.
In ~MemoryBlock(). length = 50. Deleting resource.
In ~MemoryBlock(). length = 50. Deleting resource.
In ~MemoryBlock(). length = 25. Deleting resource.
In ~MemoryBlock(). length = 50. Deleting resource.
In ~MemoryBlock(). length = 75. Deleting resource.
```

Версия этого примера, в которой используется семантика перемещения, более эффективна, чем версия, в которой эта семантика не используется, поскольку в ней выполняется меньше операций копирования, выделения памяти и освобождения памяти.

Отказоустойчивость

Во избежание утечки ресурсов (таких как память, дескрипторы файлов и сокеты) обязательно освобождайте их в операторе присваивания перемещения.

Чтобы предотвратить невозстановимое уничтожение ресурсов, в операторе присваивания перемещения необходимо правильно обрабатывать присваивания самому себе.

Если для класса определены как конструктор перемещения, так и оператор присваивания перемещения, можно исключить избыточный код, написав конструктор перемещения так, чтобы он вызывал оператор присваивания перемещения. В следующем примере показана измененная версия конструктора перемещения, вызывающая оператор присваивания перемещения:

C++

Копировать

```
// Move constructor.
MemoryBlock(MemoryBlock&& other)
    : _data(nullptr)
    , _length(0)
{
    *this = std::move(other);
}
```

[Std::move](#) функция сохраняет свойство rvalue параметра *других* параметра.

См. также

[Декларатор ссылки Rvalue: &&std::move](#)

Были ли сведения на этой странице полезными?

 Да  Нет
