Move семантики. Rule of Five.

титуляр на курса: д-р Тодор Цонков (ttsonkov@gmail.com)

От какво ще се състои настоящата лекция?

Move семантики (C++ 11) - ползи, lvalue, rvalue, move конструктор/оператор=, std::move.
Примери за клас стринг и клас Student - Rule Of Five.

Какво са move семантиките?

Move позволява прехвърляне на ресурс (pointer) от един обект в друг – почти O(1).

Копирането на големи буфери е скъпо (O(n)).

Подобрява производителността при контейнери (например std::vector) и фабрични функции.

Намалява алокациите и освобождавания на памет.

Пример

```
SimpleString a("Very very long string ...");

SimpleString b = a; // copy ctor → заделя нов буфер и копира всички символи

SimpleString c = std::move(a); // move ctor → прехвърля само указателя
```

При копиране: нов new[] + memcpy.

При move: само присвояване на указател и зануляване на стария.

Какво са lvalues и rvalues?

lvalue: стойност с име/адрес – може да има длъжност да бъде променяна. Пример: x, obj.member.

rvalue: временна стойност, няма дългосрочен адрес. Пример: 42, x + y, std::string("tmp").

съществуват още xvalues, prvalues, glvalues

Пример

```
int x = 10; // x е lvalue
int y = x; // x се използва като lvalue

int z = x + y; // (x + y) е rvalue


SimpleString s1("Hello"); // s1 е lvalue
SimpleString s2 = s1; // копиране
(lvalue → copy ctor)


SimpleString s3 = SimpleString("Temp");
// временен обект → rvalue
```

rvalue reference (T&&)

Какво е rvalue reference?

T&& може да се свърже само с rvalue (временни обекти).

Основният механизъм зад move семантиката.

Позволява "източване" на ресурси от временни обекти.

Допълнение

rvalue reference не означава временен живот — s има име и е lvalue вътре във функцията.

Затова често се използва std::move(s) вътре в тялото.

```
void takeByRvalue(SimpleString&& s) {
  // s е rvalue reference → можем да му "вземем" ресурсите
  SimpleString local = std::move(s);
}

int main() {
  SimpleString a("Hello");
  // takeByRvalue(a); // ❌ грешка: a е lvalue
  takeByRvalue(std::move(a)); // ✅ превръщаме a в rvalue
  // ✅ временен обект (rvalue)
  takeByRvalue(SimpleString("Temp"));
  return 0;
}
```

std::move - примери

```
Какво прави std::move -

std::move не мести нищо сам по себе си.
Той прави cast към rvalue reference: static_cast<T&&>(obj).
След std::move обектът е валиден, но със неопределено
съдържание

SimpleString a("Hello");
SimpleString b = std::move(a); // извиква move ctor
// a е валиден, но съдържанието му не трябва да се използва

SimpleString a("Hello");
SimpleString b;
b = std::move(a); // извиква move operator=

void foo(SimpleString s);

SimpleString x("Test");
foo(x); // copy ctor (x е lvalue)
foo(std::move(x)); // move ctor (x е cast-нат до rvalue)
```

Грешки при ползване на std::move

```
1)std::string createString() {
    std::string result = "expensive
data";
    return std::move(result);// ГРЕШКА
}
```

RVO е задължително от C++ 17!

2) Опит за move от константен обект

```
    const std::vector<int> data =
getData();
    consume(std::move(data));//copy,
not move
```

Обобщение

```
3)std::string name = "Alice";
std::string movedName =
std::move(name);
std::cout << name << std::endl;  //
```

Undefined state!

Сравнение по скорост на 10000 операции:

Операция	Време
Deep Copy	7.82 ms
Move (Correct)	1.08 ms
Move from const	7.50 ms

Move Assignment operator

1)Използва се при присвояване от rvalue.

2)Освобождава текущите ресурси и ги взема от other.

3)Оставя other в валидно, но празно състояние.

4)noexcept е важно да се слага заради STL контейнери!

Пример

```
SimpleString a("Hello");
SimpleString b("Hello");
b = std::move(a); // move оператор държи "Hello"
//a е във валидно, но специфицирано състояние
SimpleString a("A");
SimpleString b("B");
b=a;//copy operator=
b=std::move(a);//move operator=
```

Пример

Rule of Five

Ако класът управлява ресурс (heap, файл, mutex и т.н.) и дефинирате един от следните – трябва да дефинирате всички:

1) Деструктор
2) Copy конструктор
3) Copy оператор=
4) Move конструктор
5) Move оператор=

```
class Resource {
public:
  Resource();
  ~Resource(); // освобождава ресурса

  Resource(const Resource&); // копира ресурса
  Resource& operator=(const Resource&);

  Resource(Resource&&) noexcept; // прехвърля
  Resource& operator=(Resource&&) noexcept;
};
```

Пример за голямата петорка

```
class SimpleString {
public:
  SimpleString();
  SimpleString(const char* s); // конструктор от C-string
  SimpleString(const SimpleString& other); // копиращ
  SimpleString(SimpleString&& other) noexcept; // move конструктор
  SimpleString& operator=(const SimpleString& other); // копиращ =
  SimpleString& operator=(SimpleString&& other) noexcept; // move
= ~SimpleString();

private:
  char* data_ = nullptr; // менажира буфера
  size_t size_ = 0;
};
```

Имплементация на операторите за копиране

```
SimpleString::SimpleString(const SimpleString& other) {
    size_ = other.size_;
    if (size_) {
        data_ = new char[size_ + 1];
        std::memcpy(data_, other.data_, size_ + 1);
    }
}


SimpleString& SimpleString::operator=(const SimpleString& other)
{   if (&other == this) return *this;
    delete[] data_;
    size_ = other.size_;
    data_ = (size_) ? new char[size_ + 1] : nullptr;
    if (data_) std::memcpy(data_, other.data_, size_ + 1);
    return *this;
}
```

Имплементация на move операторите

```
SimpleString::SimpleString(SimpleString&& other) noexcept
: data_(other.data_), size_(other.size_) {
    other.data_ = nullptr;
    other.size_ = 0;
}


SimpleString& SimpleString::operator=(SimpleString&& other) noexcept
{   if (this == &other) return *this;
    delete[] data_; // освободим наши ресурси
    data_ = other.data_;
    size_ = other.size_;
    other.data_ = nullptr;
    other.size_ = 0;
    return *this;
}
```

Rule of Five с класа Student

```
include <cstddef>

class Student {
    char* name_;           // heap-allocated C-string
    double* grades_;       // heap-allocated array of grades
    std::size_t n_grades_;
public:
    Student(const char* name, const double* grades, std::size_t
n); ~Student();

    Student(const Student& other);
    Student& operator=(const Student& other);

    Student(Student&& other) noexcept;
    Student& operator=(Student&& other) noexcept;

    // helpers
    void print() const;
    std::size_t size() const { return n_grades_; }
};

//Student.h
```

Rule of Five operator=

```cpp
#include <utility> // std::swap

// swap helper
friend void swap(Student& a, Student& b) noexcept {
    using std::swap;
    swap(a.name_, b.name_);
    swap(a.grades_, b.grades_);
    swap(a.n_grades_, b.n_grades_);
}

// copy-assignment via copy-and-swap (strong exception
Student& Student::operator=(const Student& other) {
    if (this != &other) {
        Student tmp(other); // may throw
        swap(*this, tmp);
    }
    return *this;
}

//Student.cpp
```

Rule of Five - move assignment operator and move constructor

```
    // Move constructor
    Student(Student&& other) noexcept
        : name(other.name),
          grades(other.grades),
          n_grades(other.n_grades)
    {
        other.name = nullptr;
        other.grades = nullptr;
        other.n_grades = 0;
    }

    // Move assignment operator
    Student& operator=(Student&& other) noexcept
{       if (this != &other) {
            // release current resources
            delete[] name;
            delete[] grades;

            // steal resources
            name = other.name;
            grades = other.grades;
            n_grades = other.n_grades;

            // leave other in valid state
            other.name = nullptr;
            other.grades = nullptr;
            other.n_grades = 0;
        }
        return *this;
    }
```

Въпроси?

Благодаря за вниманието!

Допълнителни материали: learncpp.com - глава 14, глава 21