Программирование на языке C++
Лекция 8

Семантика перемещения

Александр Смаль

Излишнее копирование

```
struct String {
    String() = default;
    String(String const & s);
    String & operator=(String const & s);
    //...
private:
    char * data_ = nullptr;
    size_t size_ = 0;
};
```

```
String getCurrentDateString() {
    String date;
    // date заполняется "21 октября 2015 года"
    return date;
}
```

```
String date = getCurrentDateString();
```

Перемещающий конструктор и перемещающий оператор присваивания

```
struct String
{
    String (String && s) // && - rvalue reference
        : data_(s.data_)
        , size_(s.size_) {
        s.data_ = nullptr;
        s.size_ = 0;
    }
    String & operator = (String && s) {
        delete [] data_;
        data_ = s.data_;
        size_ = s.size_;
        s.data_ = nullptr;
        s.size_ = 0;
        return *this;
    }
};
```

Перемещающие методы при помощи swap

```
#include<utility>

struct String
{
    void swap(String & s) {
        std::swap(data_, s.data_);
        std::swap(size_, s.size_);
    }

    String (String && s) {
        swap(s);
    }

    String & operator = (String && s) {
        swap(s);
        return *this;
    }
};
```

Использование перемещения

```
struct String {
    String() = default;
    String(String const & s);  // lvalue-reference
    String & operator=(String const & s);
    String(String const && s); // rvalue-reference
    String & operator=(String const && s);
private:
    char * data_ = nullptr;
    size_t size_ = 0;
};
```

```
String getCurrentDateString() {
    String date;
    // date заполняется "21 октября 2015 года"
    return std::move(date);
}
```

```
String date = getCurrentDateString();
```

Перегрузка с lvalue/rvalue ссылками

При перегрузке перемещающий метод вызывается для временных объектов и для явно перемещённых с помощью `std::move`.

```
String a(String("Hello"));  // перемещение

String b(a);                // копирование

String c(std::move(b));     // перемещение

a = b;                      // копирование

b = std::move(c);           // перемещение

c = String("world");        // перемещение
```

Это касается и обычных методов и функций,
которые принимают lvalue/rvalue-ссылки.

Перемещающие особые методы

- Особые методы класса:
  - конструктор по умолчанию,
  - конструктор копирования,
  - оператор присваивания,
  - деструктор,
  - перемещающий конструктор,
  - перемещающий оператор присваивания.

Перемещающие особые методы

- Особые методы класса:
  - конструктор по умолчанию,
  - конструктор копирования,
  - оператор присваивания,
  - деструктор,
  - перемещающий конструктор,
  - перемещающий оператор присваивания.
- Перемещающие методы генерируются только, если в классе отсутствуют пользовательские копирующие операции, перемещающие операций и деструктор.

Перемещающие особые методы

- Особые методы класса:
  - конструктор по умолчанию,
  - конструктор копирования,
  - оператор присваивания,
  - деструктор,
  - перемещающий конструктор,
  - перемещающий оператор присваивания.
- Перемещающие методы генерируются только, если в классе отсутствуют пользовательские копирующие операции, перемещающие операций и деструктор.
- Генерация копирующих методов для классов с пользовательским конструктором признана устаревшей.

Пример: unique_ptr

```
#include <memory>
#include "units.hpp"

void foo(std::unique_ptr<Unit> p);

std::unique_ptr<Unit> bar();

int main() {
    // p1 владеет указателем
    std::unique_ptr<Unit> p1(new Elf());

    // теперь p2 владеет указателем
    std::unique_ptr<Foo> p2(std::move(p1));

    p1 = std::move(p2);  // владение передаётся p1

    foo(std::move(p1));  // p1 передаётся в bar

    p2 = bar();  // std::move не нужен
}
```