Лекция 2. Семантика перемещения

ИУ8

September 26, 2017

Содержание

На текущей лекции

- rvalue ссылки
- семантика перемещения
- прямая передача и универсальные ссылки

Мнение

Perhaps the most significant new feature in C++11 is rvalue references (*Scott Meyers*)

Rvalue ссылки, пожалуй, самая главная возможность C++11

Мотивация

Функция swap

```
1 // old style swap
2 template<class T>
3 void swap(T& a, T& b)
4 {
5   T tmp(a); // now we have two copies of a
6   a = b;  // now we have two copies of b
7   b = tmp; // now we have two copies of tmp (aka a)
8 }
```

Задача

Требуется поменять обложки двух книг местами.

Интуитивное определение

lvalue значение это выражение, которое может появляться **слева** или **справа** от знака присваивания

rvalue значение-это выражение, которое может появлятьсся **ТОЛЬКО справа** от знака присваивания

Альтернативное определение

lvalue значение-это выражение, которое ссылается на область памяти и к которому применима операция получения адреса выражения

rvalue значение-это НЕ **lvalue** выражение

Критерий rvalue

If it has a name, then it is an lvalue. Otherwise, it is an rvalue.

Пример

```
int a = 42;
int b = 43;

// a and b are both l-values:
a = b; // ok
b = a; // ok
a = a * b; // ok
```

```
// a * b is an rvalue:
int c = a * b; // ok
a * b = 42; // error

&++a; // ok (++a - lvalue)
&a++; // error (a++ - rvalue)
```

```
// lvalues:

int i = 42;
i = 43; // ok
int* p = &i; // ok
int& foo();
foo() = 42; // ok
int* p1 = &foo(); // ok
```

```
// rvalues:
int foobar();
int j = 0;
j = foobar(); // ok

//can't take address of rvalue
int* p2 = &foobar(); //error
j = 42; // ok
```

Семантика перемещений

Пример

```
1  class String {
2      char * data = nullptr;
3  public:
4      String& operator=(const String& rhs) {
5          if (this == &rhs)
6              return *this;
7
8          delete[] data; // 1.
9          data = new char[strlen(rhs.data) + 1];
10         strcpy(data, rhs.data); // 2.
11     }
12
13     ~String() {
14         delete[] data; // 3.
15     }
16 };
```

Семантика перемещений

Рассмотрим пример

```
1 String get_War_and_Peace();
2 String x;
3 // use x in various ways
4 x = get_War_and_Peace();
```

Что происходит в последней строке

Копируется ресурс из временного объекта, возвращенного из *get_War_and_Peace()*:

1. вызывается деструктор ресурса в объекте **x**
2. ресурс объекта **x** заменяется на копию ресурса из временного объекта
3. вызывается деструктор временного объекта, тем самым освобождается его ресурс

Семантика перемещений

Оператор присваивания класса X

```
1 X& X::operator=(const X& rhs) {
2   // Make a clone of what rhs.pResource refers to.
3   // Destruct the resource that pResource refers to.
4   // Attach the clone to pResource.
5 }
```

Очевидно, что в нашем примере эфективнее использовать другую схему "оператора присваивания". А именно в случае, когда аргумент оператора является rvalue, нам необходимо просто поменять местами указатели на ресурсы.

Семантика перемещений

Оператор перемещения

```
1 String& operator=(String&& rhs) {
2   if (this == &rhs)
3       return *this;
4     char * tmp = data;
5     data = rhs.data;
6     rhs.data = tmp;
7 }
```

New style swap

C++98 and std::swap

```
1 // old style swap
2 template<class T>
3 void swap(T& a, T& b) {
4   T tmp(a);
5   a = b;
6   b = tmp;
7 }
```

```
1 X a, b;
2 swap(a, b); // a, b is lvalue!
```

Итак, тут нет нигде rvalue ссылок, но интуиция должна подсказывать, что это отличное место для оптимизации и семантики перемещения.

New style swap

C++11 and std::move

```
1 // new style swap
2 template<class T>
3 void swap(T& a, T& b) {
4   T tmp(std::move(a));
5   a = std::move(b);
6   b = std::move(tmp);
7 }
```

Теперь все три строки используют семантику перемещения.

Отметим, что для типов, которые не реализуют оператор и конструктор перемещения, новый std::swap будет работать аналогично старому std::swap!

std::move

Исходный код C++14

```
1 template<typename T>
2 decltype(auto) move(T&& arg) {
3   using ReturnType = remove_reference_t<T>&&;
4   return static_cast<ReturnType>(arg);
5 }
```

- std::move выполняет безусловное примедение агрумента к rvalue ссылке
- std::move ничего не перемещает!
- std::move во время выполнения ничего не делает!
- std::move не генерирует выполняющийся код!

- Не объявляйте объекты константными, если хотите иметь возможность их перемещать
- std::move не гарантирует, что приведенный объект будет иметь право быть перемещенным

Семантика перемещения

Выводы

Семантика перемещения позволяет компиляторам заменять дорогостоящие операции копирования менее ресурсозатратными перемещениями. Перемещающие конструкторы и перемещающие операторы присваивания предоставляют контроль над семантикой перемещения.

Семантика перемещения позволяет также создавать типы, которые могут реализовывать только операцию перемещения, такие как std::unique_ptr, std::future или std::thread.

Никогда не используйте объект после того, как его переместили в другой!

Прямая передача

Формулировка задачи

Требуется реализовать функцию, которая передаст принимаемые параметры в другую функцию, не создавая временные переменные, то есть выполнит **прямую передачу**. При этом rvalue ссылки останутся rvalue ссылками, а lvalue ссылки останутся lvalue ссылками.

Прямая передача

Пример

```
1  std::string get_string();
2
3  void foo(std::string&& rvalueString); // 1.
4  void foo(std::string& lvalueString); // 2.
5
6
7  template<class T>
8  void wrapper(T&& arg)
9  {
10     // ...
11 }
12
13 wrapper(get_string()); // 1.
14 std::string str = "something";
15 wrapper(str); // 2.
```

В функции wrapper необходимо вызвать функцию foo, принимающую rvalue ссылку, в том случае, если во wrapper передано rvalue значение. Но если в функцию wrapper передано lvalue значение, необходимо вызвать функцию foo, принимающую lvalue ссылку.

Прямая передача

Пример

```
1  std::string get_string();
2
3  void foo(std::string&& rvalueString); // 1.
4  void foo(std::string& lvalueString); // 2.
5
6
7  template<class T>
8  void wrapper(T&& arg)
9  {
10     // arg is always lvalue
11     foo(std::forward<T>(arg)); // Forward as lvalue or as rvalue,
           depending on T
12 }
13
14 wrapper(get_string()); // 1.
15 std::string str = "something";
16 wrapper(str); // 2.
```

Прямая передача

std::forward

Задача о прямой передаче аргументов решается с помощью std::forward

- std::forward выполняет приведение к типу, который указан (выводится компилятором) внутри угловых скобок
- std::forward ничего не передает!
- std::forward во время выполнения ничего не делает!
- std::forward не генерирует выполнающийся код!

```
1 template<class T>
2 T&& forward(remove_reference_t<T> & a) {
3   return static_cast<T&&>(a);
4 }
```

Прямая передача

Выводы

Прямая передача делает возможным написание шаблонов функций, которые принимают произвольные аргументы и передают их другим функциям так, что целевые функции получают **в точности** те же аргументы, что и переданные исходным функциям.

Rvalue-ссылки делают возможными как семантику перемещения, так и прямую передачу.

Return value optimization

Good sample

```
1 Widget makeWidget() {
2   Widget w;
3   // ...
4   return w;
5 }
```

Bad sample

```
1 Widget makeWidget() {
2   Widget w;
3   // ...
4   return std::move(w);
5 }
```

Никогда не применяйте std::move и std::forward к локальным объектам, которые могут быть объектом оптимизации возвращаемого значения.

The end

Самостоятельное изучение

- Правило свертывания ссылок
- Когда сематика перемещения не работает
- Случаи некорректной работы прямой передачи
- Почему стоит избегать перегрузки универсальных ссылок

Список литературы

- С. Мэйерс - Эффективный и современный C++
- https://isocpp.org/blog/2012/11/universal-references-in-c11-scott-meyers
- http://thbecker.net/articles/rvalue_references/section_01.html
- https://accu.org/index.php/journals/227
- http://alenacpp.blogspot.ru/2008/02/rvo-nrvo.html
- https://www.ibm.com/developerworks/community/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/RVO_V_S_std_move?lang=en