Ссылки и умные указатели

2021

Ссылки

```
X obj;                    // объект
X& ref1 = obj;            // ссылка
const X& ref2 = obj;      // константная ссылка
```

Ссылки

```
X obj;                    // объект
X& ref1 = obj;            // ссылка
const X& ref2 = obj;      // константная ссылка
```

```
X func();                 // функция
X& ref3 = func();         // ошибка
const X& ref4 = func();   // ок
```

Левые и правые значения

Lvalue — значения с именем (слева и справа от =).
Rvalue — значения без имени (только справа от =).

```
X x, y, z;    // ок
x = y;        // ок
x*y = z;      // ошибка
z = x*y;      // ок
```

Ссылки на левые и правые значения

```
X obj;                    // объект
X& ref1 = obj;            // ссылка
const X& ref2 = obj;      // константная ссылка
```

```
X func();                 // функция
X& ref3 = func(); // ошибка
const X& ref4 = func();   // ок
```

Ссылки на левые и правые значения

```
X obj;                    // объект
X& ref1 = obj;            // ссылка
const X& ref2 = obj;      // константная ссылка
```

```
X func();                 // функция
X&& ref3 = func(); // ок
const X& ref4 = func();   // ок
```

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
obj	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	lvalue

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	lvalue
`ref3`	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	lvalue
`ref3`	lvalue

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	lvalue
`ref3`	lvalue
`ref4`	???

Правое или левое значение?

```
      X&  ref1 = obj;
const X&  ref2 = obj;
      X&& ref3 = func();
const X&  ref4 = func();
```

	Lvalue/Rvalue?
`obj`	lvalue
`func()`	rvalue
`ref1`	lvalue
`ref2`	lvalue
`ref3`	lvalue
`ref4`	lvalue

Перемещение с `std::move`

```
template <class T>
typename std::remove_reference<T>::type&&
move(T&& a) {
  return static_cast<typename std::remove_reference<T>::type&&>(a);
}
```

T	Модиф.	Тип аргумента
X&	&	X&
X&&	&	X&
X&	&&	X&
X&&	&&	X&&

Пересылка с `std::forward`

```
template <class T>
T&& forward(typename std::remove_reference<T>::type& a) {
    return static_cast<T&&>(a);
}

template <class T>
T&& forward(typename std::remove_reference<T>::type&& a) {
    return static_cast<T&&>(a);
}
```

T	Модиф.	Тип аргумента
X&	&	X&
X&&	&	X&
X&	&&	X&
X&&	&&	X&&

Конструктор и оператор присваивания

```
template <class T>
class vector {
public:
    vector(vector&&) = default;
    vector& operator=(vector&&) = default;
};
```

Конструктор с перемещением (1)

```
template <class T>
class vector {
    T* first; T* last;
public:
    vector(vector&& x): first(x.first), last(x.last) {
        x.first = nullptr;
        x.last = nullptr;
    }
    ~vector() { delete[] first; }
};
```

Конструктор с перемещением (2)

```
template <class A, class B>
struct pair {

    A first;
    B second;

    // идеальная пересылка (perfect forwarding)
    pair(pair&& x):
    first(std::forward<A>(x.first)),
    second(std::forward<B>(x.second)) {}

};
```

Присваивание с перемещением

```
template <class T>
class vector {
    T* first; T* last;
public:
    vector& operator=(vector&& x) {
        first = x.first;
        x.first = nullptr;
        last = x.last;
        x.last = nullptr;
        return *this;
    }
};
```

Присваивание с обменом

```
template <class T>
class vector {
    T* first; T* last;
public:
    vector& operator=(vector&& x) { swap(x); return *this; }

    void swap(vector& x) {
        std::swap(first, x.first);
        std::swap(last, x.last);
    }
};

void swap(vector& a, vector& b) { a.swap(b); }
```

Обмен с перемещением

```
template <class T>
void swap(T& a, T& b) {
    T tmp = move(a);
    a = move(b);
    b = move(tmp);
}
```

Сравнение forward и move

```
X x, y, z;
x = std::move(y);         // operator=(X&&)
y = z;                    // operator=(const X&)
z = std::forward<X&>(x); // operator=(const X&)
y = std::forward<X>(z);  // operator=(X&&)
```

Идеальная пересылка

```
template <class T, class A1>
std::shared_ptr<T>
make_shared(A1&& a1) {
    return std::shared_ptr<T>(new T(std::forward<A1>(a1)));
}
```

Что такое умный указатель?

```
X* func() {
    X* x = new X;
    ...
    return x;
}
```

Что такое умный указатель?

```
X* func() {
    X* x = new X;
    ...
    return x;
}
```

```
std::unique_ptr<X> func() {
    std::unique_ptr<X> x(new X);
    ...
    return x;
}

std::unique_ptr<X> x = func(); // rvalue
```

Умный указатель (CUDA)

```
template <class T>
T* gpuAlloc(size_t n) {
    void* ptr = nullptr;
    cudaMalloc(&ptr, sizeof(T)*n);
    return static_cast<T*>(ptr);
}

struct gpuDelete {
    void operator()(void* ptr) { cudaFree(ptr); }
};

std::unique_ptr<float,gpuDelete> x(gpuAlloc<float>(100));
```

Искусственный пример

```
void func() {
    std::shared_ptr<int> p1(new int);      // count=1
    {
        std::shared_ptr<int> p2(p1);       // count=2
        {
            std::shared_ptr<int> p3(p1);   // count=3
        }                                  // count=2
    }                                      // count=1
}                                          // count=0
```

Пример с астероидами

```
void neigbours() {
    std::vector<std::shared_ptr<Asteroid>> asteroids(100);
    for (auto& ast : asteroids) {
        // neighbour = ...
        ast.setNeighbour(std::weak_ptr<Asteroid>(neighbour));
    }
}

void collision(std::weak_ptr<Asteroid> p) {
    std::shared_ptr<Asteroid> q = p.lock();
    if (q) {
        // столкновение
    } else {
        // астероид удален
    }
}
```

Статистика

Проект	`unique_ptr`	`shared_ptr`	`delete`
Tungsten	270	442	69
Magnum	213	0	12
Vtestbed	10	0	0
Unistdx	7	0	1
QPP	1	0	0
EOS	1	0	0

Ссылки

- Альтернативное объяснение `std::move` и `std::forward` от создателей стандарта C++ (раздел «Teachability»): N2951 (англ.).
- Еще одно объяснение на английском: C++ `std::move` and `std::forward`.