OTUS

R.20: Use unique_ptr or shared_ptr to represent ownership

Дмитрий Шебордаев, Ведущий специалист

OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

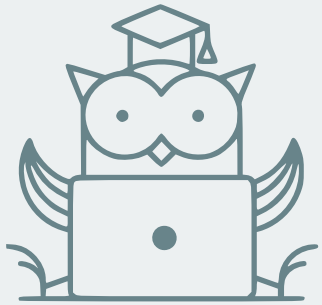
Дмитрий Шебордаев, Ведущий специалист

Введение

Smart pointers

Дмитрий Шебордаев

Ведущий специалист



Сегодня

- Область видимости
- Велосипед
- unique_ptr
- shared_ptr
- weak_ptr

Область видимости

```
auto x = 1;

auto foo() {
    return x;
}

int main() {
    auto w = foo(); // ?
}
```


Область видимости

```
auto x = 1;

auto foo() {
    auto x = 2;
    return x;
}

int main() {
    auto w = foo(); // ?
}
```


Область видимости

```
auto x = 1;

auto foo(int x) {
    auto x = 2;
    return x;
}

int main() {
    auto w = foo(3); // ?
}
```


Область видимости

```
auto x = 1;

auto foo() {
    return x;
}

int main() {
    auto x = 4;
    auto w = foo(); // ?
}
```


Область видимости

```
auto x = 1;

template<typename T>
auto foo(T y) {
    return x + y;
}

int main() {
    auto x = 2;
    auto w = foo(10); // ?
}
```


Время жизни

```
auto str_1 = std::string{};
thread_local auto str_2 = std::string{};

void foo(std::string str_3) {
    static auto str_4 = std::string{};
}

int main() {
    auto str_5 = std::string{};
    foo(std::string{});

    auto str_6 = new std::string{};
    delete arr_6;
}
```


Время жизни

Storage duration

- статическое - глобальные + `static`. Живут процесс
- потоковое - `thread_local`. Живут поток
- автоматическое - всё остальное, аргументы. Живут в области видимости
- динамическое - `new/delete`, `malloc/free`. Как прикажут

OTUS

Сырые/нативные указатели

- `int x;` // ничего не знаем про адрес
- `int *y = &x;` // хотим знать адрес
- `int &z = x;` // хотим знать и не знать одновременно

Сырые/нативные указатели

- не может следить за созданием и уничтожением
- может указывать в неизвестность, nullptr
- может указывать в известность, но чужую

Сырые/нативные указатели

- не может следить за созданием и уничтожением
- может указывать в неизвестность, nullptr
- может указывать в известность, но чужую

Наша цель обернуть динамическое время жизни

Зато своё

```
template<typename T>
class smart_ptr {
public:
    smart_ptr() {
        ptr = new T{};
    }
    T * get() {
        return ptr;
    }
    ~smart_ptr() {
        delete ptr;
    }
public:
    T *ptr;
};
```


unique_ptr

Указатель на объект

- `std::unique_ptr<int> p{new int{42}};`
- `auto p = std::make_unique<int>(12);`
- `*, ->, get()`

Handler

```
int *init() {};

done(int *) {};

std::unique_ptr<int, decltype(&done)>
    ptr{init(), &done};
```


OTUS

Свой деструктор

```
class deleter
{
public:
    void operator()(int *handler)
    {
        done(handler);
    }
};

std::unique_ptr<int, deleter> j{init()};
```


OTUS

Зато своё, но еще лучше

```
template<typename T>
class smart_ptr {
public:
    smart_ptr() {
        counter = new int{1};
        ptr = new T{};
    }
    T * get() {
        return ptr;
    }
    ~smart_ptr() {
        if (--*counter == 0) {
            delete ptr;
            delete counter;
        }
    }
public:
    T *ptr;
    int *counter;
};
```


shared_ptr

- можно копировать
- перемещать дешевле
- всегда хранит два указателя
- std::make_shared лучше
- счетчики потоко безопасны
- можно создать из unique_ptr, но не наоборот

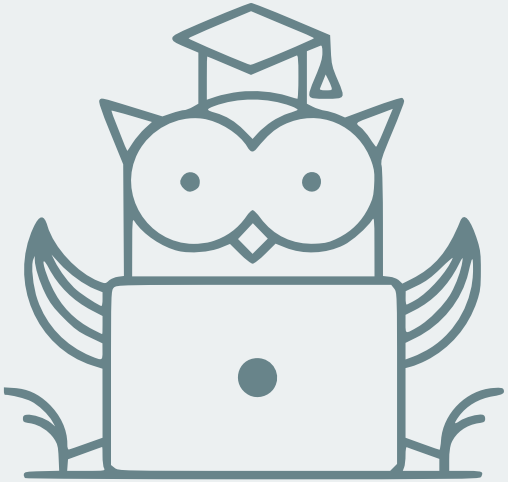
weak_ptr

```
std::weak_ptr<int> weak;
{
    auto ptr = std::make_shared<int>(42);
    weak = ptr;

    auto x = weak.lock();
}
auto x = weak.lock();
```


почитать

- http://en.cppreference.com/w/cpp/memory/unique_ptr
- http://en.cppreference.com/w/cpp/memory/shared_ptr
- http://en.cppreference.com/w/cpp/memory/weak_ptr
- http://www.ozon.ru/context/detail/id/34747131/



Спасибо за внимание!