C++ 11: Lambdas

Иван Крюков

EKB .cpp

Екатеринбург,
28.06.2014

Интрига

- Безымянные функции в месте вызова
- Одноразовые функциональные объекты "на лету"

Old school

```
typedef vector<float> TInfos;
struct TSorter {
    const TInfos& Infos;
    TSorter(const TInfos& infos): Infos(infos) {}
    bool operator()(int a, int b) const {
        return Infos[a] > Infos[b];
    }
};
...
{
   vector<int> order = {0, 1, 2, 3, 4};
   TInfos infos = {0.3, 0.2, 0.7, 0.1, 0.6};
   std::sort(order.begin(), order.end(), TSorter(infos));
}
```

Old school

```
typedef vector<float> TInfos;
struct TSorter {
    const TInfos& Infos;
    TSorter(const TInfos& infos): Infos(infos) {}
    bool operator()(int a, int b) const {
        return Infos[a] > Infos[b];
    }
};
...
{
   vector<int> order = {0, 1, 2, 3, 4};
   TInfos infos = {0.3, 0.2, 0.7, 0.1, 0.6};
   std::sort(order.begin(), order.end(), TSorter(infos));
}
```

λ way

```
typedef vector<float> TInfos;
vector<int> order = {0, 1, 2, 3, 4};
TInfos infos = {0.3, 0.2, 0.7, 0.1, 0.6};

std::sort(
   order.begin(), order.end(),
   [&](int a, int b) {
      return infos[a] > infos[b];
   }
);
```

λ way

```
typedef vector<float> TInfos;
vector<int> order = {0, 1, 2, 3, 4};
TInfos infos = {0.3, 0.2, 0.7, 0.1, 0.6};

std::sort(
   order.begin(), order.end(),
   [&](int a, int b) {
      return infos[a] > infos[b];
   }
);
```

Синтаксис

1. `[ capture-list ] ( params ) mutable(optional) exception -> ret { body }`
2. `[ capture-list ] ( params ) -> ret { body }`
3. `[ capture-list ] ( params ) { body }`
4. `[ capture-list ] { body }`

Синтаксис: **mutable**

```
1.[ capture-list ] ( params ) mutable(optional)
  exception -> ret { body }
```

- позволяет изменять захваченные по значению параметры внутри λ и вызывать их non-const методы

Синтаксис: **возвращаемый тип**

```
2.[ capture-list ] ( params ) -> ret { body }
```

- Нужно указывать явно если в теле > 1 «`return`»

Синтаксис: **параметры**

```
3.[ capture-list ] ( params ) { body }
```

- никаких `auto` (ждем '14)

Синтаксис: **список захвата**

```
4.[ capture-list ] { body }
```

- `[], [=], [&], [&a, b], [=, &a], [&, a], [this]`
- глобальные и `static` переменные доступны всегда

Нельзя:

- захватывать локальные `static` переменные
- члены класса (только через `this`)
- ~~члены анонимных объединений (`unions`)~~

λ != closure

```
struct TSorter {
    const TInfos& Infos;
    TSorter(const TInfos& infos): Infos(infos) {}
    inline bool operator()(int a, int b) const {
        return Infos[a] > Infos[b];
    }
};
```

- λ — это синтаксический сахар, конструкция языка
- closure — function object, run-time

Пример: когда происходит захват?

```
int v = 42;
auto func = [=] {
   std::cout << v << std::endl;
};
v = 14;
Func();
```

Пример: когда происходит захват?

```
int v = 42;
auto func = [=] {
   std::cout << v << std::endl;
};
v = 14;
func();
```

в момент создания! > 42

Пример: когда происходит захват?

```
int v = 42;
auto func = [&] {
   std::cout << v << std::endl;
};
v = 14;
func();
```

в момент создания! > 14

Тренируем захваты

```
int i=1, j=1, k=1;
auto f = [i, &j, &k]() mutable {
    auto m = [i, j, &k]() mutable {
        std::cout << i << j << k << std::endl;
        i=4; j=4; k=4;
    };
    i=3; j=3; k=3;
    m();
};
i=2; j=2; k=2;
f();
std::cout << i << j << k << std::endl;
```

Тренируем захваты

```
int i=1, j=1, k=1;
auto f = [i, &j, &k]() mutable {
    auto m = [i, j, &k]() mutable {
        std::cout << i << j << k << std::endl;
        i=4; j=4; k=4;
    };
    i=3; j=3; k=3;
    m();
};
i=2; j=2; k=2;
f();
std::cout << i << j << k << std::endl;
```

```
> 123
  234
```

Опасность

```
{
    struct Alive {
        std::string happy;
        Alive(): happy("unicorn") {}
        ~Alive() { happy.~string(); }
    };

    std::function<void (void)> bang;
    if (1) {
        Alive a;
        bang = [&a](){ cout << a.happy << endl; };
        bang();
    }
    bang();
}
```

хДыщъ!

main(12324) malloc: *** error for object 0x7fd74a403970: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug

Пример: возвращаемый тип

```
vector<double> dest;
transform(begin(src), end(src),
          back_inserter(dest), [](int v) -> double
{
   if (v < 5)
        return v + 1.0;
   else if (v % 2 == 0)
        return v / 2.0;
   else
        return v * v;
});
```

Пример: auto

```
enum Sides {Left, Right};
vector<pair<int,int>> v{{1,2}, {3,4}, {5,6}};
vector<pair<int,int>> out(v.size());
transform(
    v.begin(), v.end(), out.begin(),
    [](pair<int,int>& p) {
        swap(get<Left>(p), get<Right>(p));
        return p;
    }
);
```

Пример: auto

```
enum Sides {Left, Right};
vector<pair<int,int>> v{{1,2}, {3,4}, {5,6}};
vector<pair<int,int>> out(v.size());
transform(
    v.begin(), v.end(), out.begin(),
    [](auto& p) {
        swap(get<Left>(p), get<Right>(p));
        return p;
    }
);
```

```
error: parameter declared 'auto'
         [](auto& p) {
                  ^
```

λ — объекты первого класса

```
template <class T>
void caller(T f) {
    std::cout << f(1) << std::endl;
}

typedef int(*func)(int);
func f0 = [](int i){ return i*2; };

auto f1 = [=](int i){ return f0(i+3); };
function<int(int)> f2 = [=](int i){return f1(i);};
vector<function<int(int)>> v = {f2};
caller(v[0]);
```

```
> 8
```

Пример: "карринг"

```
float sum(int a, float b) {return a + b;}

auto curry = [](std::function<float(int, float)> s){
    return [s](int a) {
        return [=](float b) {
            return s(a,b);
        };
    };
};
auto curried = curry(sum);
cout << curried(3)(4.5) << endl;
cout << curried(1)(2.1) << endl;

> 7.5
  3.1
```

Пример: "карринг"

```
Breakpoint 1, operator() (__closure=0x7fff5fbfee30,
b=2.0999999) at main.cpp:121
121                 return s(a,b);
(gdb) bt
#0  operator() (__closure=0x7fff5fbfee30, b=2.0999999) at
main.cpp:121
#1  0x0000000100001b11 in f55 () at main.cpp:127
#2  0x0000000100002777 in main () at main.cpp:221
(gdb) print __closure

$1 = (const '__lambda11::__lambda12::__lambda13' * const)
0x7fff5fbfee30
```

Пример: this

```
class Foo {
        int Value;
        void workImpl() {
            cout << "private call " << ++Value << endl;
        }
    public:
        Foo(int v = 0) : Value(v) {}
        void Work() {
            [](){ workImpl(); }();
        }
    };
Foo val(5);
val.Work();
```

> error: 'this' was not captured for this lambda function

Пример: this

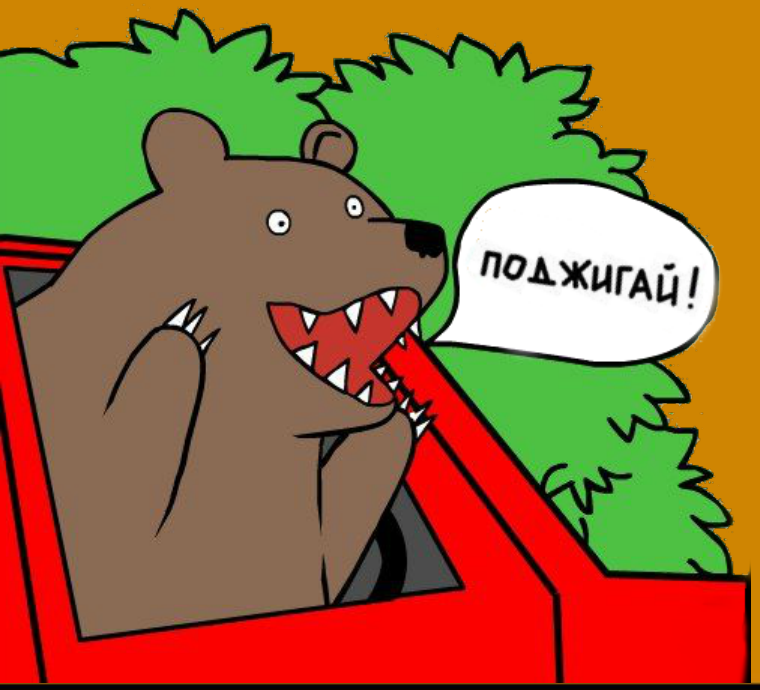
```
class Foo {
        int Value;
        void workImpl() {
            cout << "private call " << ++Value << endl;
        }
    public:
        Foo(int v = 0) : Value(v) {}
        void Work() {
            [this](){ workImpl(); }();
        }
    };
Foo val(5);
val.Work();
```

Пример: invisible this

```
void MyClass::MyFunc(int x) {
    MySomething y;
    auto lambda = [=] {
        f(x);          // x captured by value
        y.g();         // y captured by value
        cout << z ;
        // this captured by value,
        // if z is a member variable...
        // z is captured as if by reference
    };
}
```

Где λ жгут

- std:: (find_*, all_of, transform, bind, for_each, …)
- runtime policies
- callbacks



Где нельзя λ

- template-argument
- default arguments
- constexpr
- typedef
- typeid
- sizeof
- noexcept
- decltype

Итого

- λ выражения - способ создания анонимных функторов
- Повышают читаемость кода
- Способны захватывать переменные в области видимости
- С λ можно auto или std::function
- STL friendly

Discuss, Destroy, Ruin me!

Спасибо за внимание!

@kubykupa