Лекция 9

01/12/2018

Темы

- Лямбда-функции
- Паттерны проектирования

Лямбда-функции и замыкания (1/2)

- Лямбда-функция – вариант реализации функциональных объектов в языке C++11, обязанный своим названием **λ-исчислению** – математической системе определения и применения функций, в которой аргументом одной функции может быть другая функция.
- Лямбда-функции в основном определяются в точке их применения.

Лямбда-функции и замыкания (2/2)

- Лямбда-функции могут использоваться всюду, где требуется **передача вызываемому объекту функции** соответствующего типа.
- Лямбда-функции могут использовать внешние по отношению к ней самой племенные, образующие **замыкание** такой функции.

```
bool foo(int x) { return x % m == 0; }
```

эквивалентно

```
[] (int x) { return x % m == 0; }
```

Правила оформления лямбда-функций

- При преобразовании функции языка C++ в лямбда-функцию необходимо учитывать, что имя функции заменяется в лямбда-функции квадратными скобками [], а возвращаемый тип:
  - не определяется явно (слева)
  - при анализе лямда-функции с телом вида

return expr;

  - при отсутствии в теле однооператорной лямбда-функции оператора return автоматически принимается равным **void**
  - в остальных случаях задается программистом при помощи “хвостового” способа записи:

[](**int** x) -> **int** { **int** y = x; **return** x – y; }

Преимущества лямбда-функций

- Близость – определяются в месте их дальнейшего применения
- Краткость – немногословны и могут быть использованы повторно
- Эффективность – могут встраиваться компилятором в точку определения на уровне объектного кода

Анонимные лямбда-функции

```
std::string s("hello");
std::transform(s.begin(), s.end(), s.begin(),
                    [](unsigned char c) -> unsigned char { return std::toupper(c); });

std::vector<size_t> ordinals;
std::transform(s.begin(), s.end(), std::back_inserter(ordinals),
                    [](unsigned char c) -> size_t { return static_cast<size_t>(c); });
std::cout << s << ':';
for (size_t ord : ordinals) {
        std::cout << ' ' << ord;
}
```

Именованные лямбда-функции

```
auto lt10 = [](int x) { return x < 10; };
int cnt = std::count_if(v1.being(), v1.end(), lt10);
bool b = lt10(300);          // b == false
```

Внешние переменные и замыкание лямбда-функций

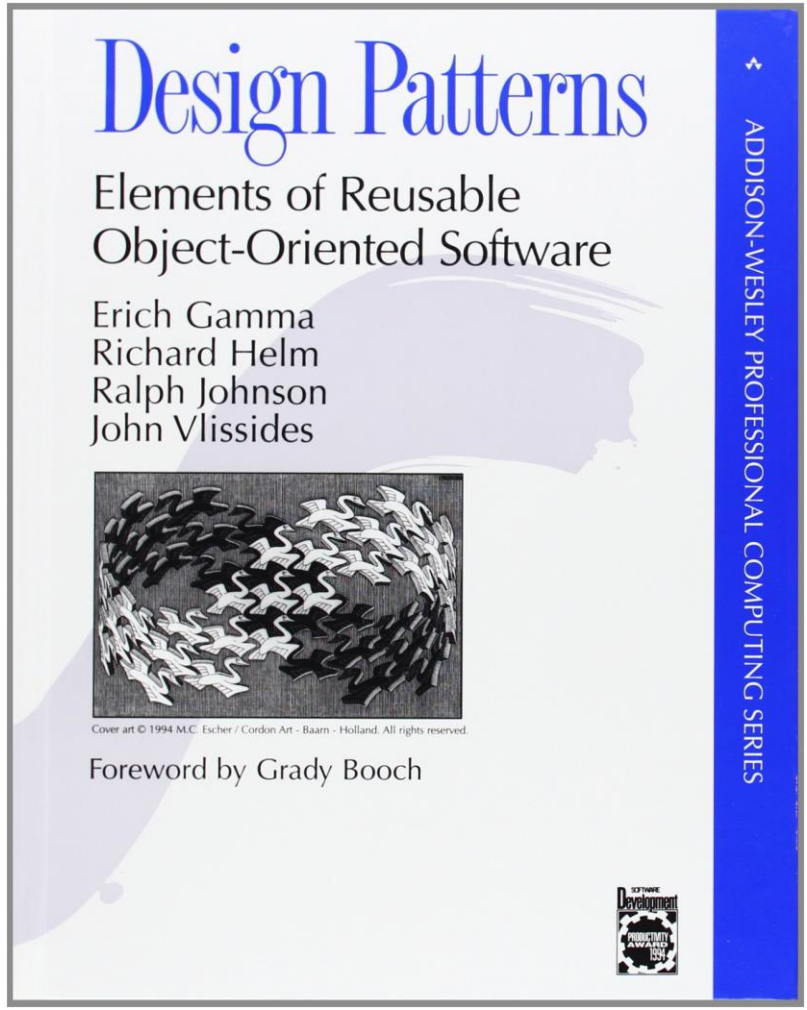
- Внешние по отношению к лямбда-функции переменные, определенные в одной с ней области видимости, могут захватываться лямбда-функцией и входить в ее замыкание. При этом в отношении доступа к переменным действуют следующие правила:
  - [z] – доступ по значению к одно переменной z
  - [&z] – доступ по ссылке к одной переменной z
  - [=] – доступ по значению ко всем переменным
  - [&] – доступ по ссылке ко всем переменным
  - [&, z], [=, &z], [z, &zz] – смешанный вариант доступа.

Внешние переменные и замыкание лямбда-функций

```
std::vector<double> vd; // …
int countN = std::count_if(vs.begin(), vs.end(),
                                    [](double x) {return x >= N; });
// эквивалентно
int countN = 0;
std::vector<double> vd; //…
std::for_each(vd.begin(), vd.end(),
                    [&countN](double x) { countN += x >= N; });
```

Паттерны проектирования

- Обобщенные типовые архитектурные решения
- Gang of Four
- По определению GoF, паттерн – это описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном примере
- Активное использование паттернов позволяет повысить качество кода -> обеспечить качество сопровождения ПО.



Паттерны проектирования

- Делятся на 3 категории:
  - Порождающие – процессы создания объектов
  - Структурные – способы композиции классов (объектов)
  - Поведенческие – взаимодействие классов (объектов) между собой

Паттерны GoF

Порождающие (4)	Структурные (8)	Поведенческие (11)
Фабричный метод (Factory method)	Адаптер (Adapter)	Интерпретатор (Interpreter)
Абстрактная фабрика (Abstract factory)	Декоратор (Decorator)	Шаблонный метод (Template method)
Прототип (Prototype)	Заместитель (Proxy)	Итератор (Iterator)
Строитель (Builder)	Компоновщик (Composite)	Команда (Command)
	Мост (Bridge)	Наблюдатель (Observer)
	Приспособленец (Flyweight)	Посетитель (Visitor)
	Фасад (Façade)	Посредник (Mediator)
		Состояние (State)
		Стратегия (Strategy)
		Хранитель (Memento)
		Цепочка ответственности (Chain of responsibility)