Лекция 15. SFINAE. PImpl.

ИУ8

December 16, 2016

Содержание

План лекции:

- SFINAE
- PImpl

SFINAE

Аббревиатура SFINAE расшифровывается как substitution failure is not an error: ошибочные инстанциации шаблонов не вызывают ошибку компиляции

При определении перегрузок функции ошибочные инстанциации шаблонов не вызывают ошибку компиляции, а отбрасываются из списка кандидатов на наиболее подходящую перегрузку.

- Когда речь заходит о SFINAE, это обязательно связано с перегрузкой функций.
- Это работает при автоматическом выводе типов шаблона (type deduction) по аргументам функции.
- Некоторые перегрузки могут отбрасываться в том случае, когда их невозможно инстанциировать из-за возникающей синтаксической ошибки; компиляция при этом продолжается как ни в чём не бывало, без ошибок.
- SFINAE рассматривает только заголовок функции, ошибки в теле функции не будут пропущены.

Пример

```
1  template<typename T>
2  void clear(T& t, std::enable_if_t<std::is_pod<T>::value>* = 0) {
3    std::memset(&t, 0, sizeof(t));
4  }
5
6  template<typename T>
7  void clear(T& t, std::enable_if_t<!std::is_pod<T>::value>* = 0) {
8    // T must have copy operator
9    t = T{};
10 }
11
12 int main() {
13   int i = 241;
14   double d = 1324.1234;
15   std::string s = "some str";
16   clear(i);
17   clear(d);
18   clear(s);
19 }
```

Еще пример

```
1  template <class T>
2  struct has_iterator {
3    typedef char yes[1];
4    typedef char no[2];
5
6    template <typename C>
7    static yes& test(typename C::iterator *);
8
9    template <typename C>
10   static no& test(...);
11
12   static const bool value = sizeof(test<T>(nullptr)) == sizeof(yes);
13 };
14
15 int main() {
16   std::cout << has_iterator<std::vector<int>>::value << std::endl; //
         true
17   std::cout << has_iterator<int>::value << std::endl;    // false
18 }
```

Применение SFINAE

SFINAE позволяет управлять выбором требуемой специализации или перегрузки для конкретного типа. Это в свою очередь используется для:

- проверки наличия определенного метода в типе
- создания оптимизированных перегрузок функций и методов
- проверки наличия вложенных типов

Идиома PImpl

PImpl - Pointer to implementation.

PImpl - идиома, при которой детали реализации класса выносятся в отдельный класс, доступ к которому осуществляется через указатель.

Пример. GeneralSocket.h

```
class UnixSocketImpl;
class GeneralSocket {
public:
  void connect();
private:
  std::unique_ptr<UnixSocketImpl> impl;
}
```

GeneralSocket.cpp

```
#include "UnixSocketImpl.h"
void GeneralSocket::connect() {
  impl->connect();
}
```

UnixSocketImpl.cpp

```
void UnixSocketImpl::connect() {
  // changing some code
}
```

Плюсы PImpl

- минимизация зависимостей компиляции (а следовательно ускорение компиляции проекта)
- разделение интерфейса и реализации класса
- переносимость кода

Минусы PImpl

- каждое создание объекта требует динамического выделения памяти для объекта, на который ссылается указатель
- лишние затраты по времени
- лишние затраты по памяти

The end

Список литературы

- Шпаргалка по шаблонам проектирования
- Паттерны ООП в метафорах
- Valery Lesin. C++ In-Depth, 2014
- cppreference.com: PImpl
- Герб Саттер - Решение сложных задач на C++. Серия "C++ In-Depth"