Динамическое приведение типов (dynamic_cast) в C++

Обновл. 24 Мар 2021 |

На уроке о **явном преобразовании типов данных** мы рассматривали использование оператора static_cast для конвертации переменных из одного типа данных в другой. На этом уроке мы рассмотрим еще один оператор явного преобразования — dynamic_cast.

Зачем нужен dynamic_cast?

Применяя **полиморфизм** на практике вы часто будете сталкиваться с ситуациями, когда у вас есть **указатель** на родительский класс, но вам нужно получить доступ к данным, которые есть только в дочернем классе. Например:

```
1  #include <iostream>
2  #include <string>
3  class Parent
4  {
5  protected:
6  int m_value;
7  public:
8  Parent(int value)
9  : m_value(value)
10 {
11 }
12 virtual ~Parent() {}
13 };
14 class Child: public Parent
15 {
```

```
16
17
18 protected:
19 std::string m_name;
20 public:
21 Child(int value, std::string name)
22 : Parent(value), m_name(name)
23 {
24 }
25 const std::string& getName() { return m_name; }
26 };
27 Parent* getObject(bool bReturnChild)
28 {
29 if (bReturnChild)
30 return new Child(1, "Banana");
31 else
32 return new Parent(2);
33 }
34 int main()
35 {
36 Parent *p = getObject(true);
37 // Как мы выведем имя объекта класса Child здесь, имея лишь один
   указатель класса Parent?
38
   delete p;
39
   return 0;
40
   }
41
42
```

```
43
44
45
46
47
48
49
```

В этой программе метод getObject() всегда возвращает указатель класса Parent, но этот указатель может указывать либо на объект класса Parent, либо на объект класса Child. В случае, когда указатель указывает на объект класса Child, как мы будем вызывать Child::getName()?

Один из способов — добавить виртуальную функцию getName() в класс Parent (чтобы иметь возможность вызывать переопределение через объект класса Parent). Но, используя этот вариант, мы будем загромождать класс Parent тем, что должно быть заботой только класса Child.

Язык C++ позволяет нам неявно конвертировать указатель класса Child в указатель класса Parent (фактически, это и делает getObject()). Эта конвертация называется **приведением к базовому типу** (или ***«повышающим приведением типа»***). Однако, что, если бы мы могли конвертировать указатель класса Parent обратно в указатель класса Child? Таким образом, мы могли бы напрямую вызывать Child::getName(), используя тот же указатель, и вообще не заморачиваться с виртуальными функциями.

Оператор dynamic_cast

В языке C++ **оператор dynamic_cast** используется именно для этого. Хотя динамическое приведение позволяет выполнять не только конвертацию указателей родительского класса в указатели дочернего класса, это является наиболее распространенным применением оператора dynamic_cast. Этот процесс называется **приведением к дочернему типу** (или ***«понижающим приведением типа»***).

Использование dynamic_cast почти идентично использованию static_cast. Вот функция main() из вышеприведенного примера, где мы используем dynamic_cast для конвертации указателя класса Parent обратно в указатель класса Child:

```
int main()
{
```

```
Parent *p = getObject(true);

    Child *ch = dynamic_cast<Child*>(p); // используем dynamic_cast для конвертации указателя класса Parent в указатель класса Child

    std::cout << "The name of the Child is: " << ch->getName() << '\n';

delete p;

return 0;

}
```

Результат:

```
The name of the Child is: Banana
```

Невозможность конвертации через dynamic_cast

Вышеприведенный пример работает только из-за того, что указатель `p` на самом деле указывает на объект класса Child, поэтому конвертация успешна.

«А что произошло бы, если бы `p` не указывал на объект класса Child?» — спросите Вы. Это легко проверить, изменив аргумент метода getObject() из `true` на `false`. В таком случае getObject() будет возвращать указатель класса Parent на объект класса Parent. Если затем мы попытаемся использовать dynamic_cast для конвертации в Child, то потерпим неудачу, так как подобное преобразование невозможно.

Если dynamic_cast не может выполнить конвертацию, то он возвращает **нулевой указатель**.

Поскольку в коде, приведенном выше, мы не добавили проверку на нулевой указатель, то при выполнении `ch->getName()` мы попытаемся разыменовать нулевой указатель, что, в свою очередь, приведет к неопределенным результатам (или к сбою).

Чтобы сделать программу безопасной, необходимо добавить проверку результата выполнения dynamic_cast:

```
int main()

{

Parent *p = getObject(true);

    Child *ch = dynamic_cast<Child*>(p); // используем dynamic_cast для конвертации указателя класса Parent в указатель класса Child

    if (ch) // выполняем проверку ch на нулевой указатель
```

```
        std::cout << "The name of the Child is: " << ch->getName() << '\n';

    delete p;

    return 0;

    }
```

Правило: Всегда делайте проверку результата динамического приведения на нулевой указатель.

Обратите внимание, поскольку динамическое приведение выполняет проверку во время запуска программы (чтобы гарантировать возможность выполнения конвертации), использование оператора dynamic_cast чуть снижает производительность программы.

Также обратите внимание на **случаи, в которых понижающее приведение с использованием оператора dynamic_cast не работает**:

Наследование типа private или **типа protected**.

Классы, которые не объявляют или не наследуют классы с какими-либо виртуальными функциями (и, следовательно, не имеют **виртуальных таблиц**). В примере, приведенном выше, если бы мы удалили виртуальный деструктор класса Parent, то преобразование через dynamic_cast не выполнилось бы.

Случаи, связанные с **виртуальными базовыми классами** (на **сайте Microsoft** вы можете посмотреть примеры таких случаев и их решения).

Понижающее приведение и оператор static_cast

Оказывается, понижающее приведение также может быть выполнено и через оператор static_cast. Основное отличие заключается в том, что static_cast не выполняет проверку во время запуска программы, чтобы убедиться в том, что вы делаете то, что имеет смысл. Это позволяет оператору static_cast быть быстрее, но опаснее оператора dynamic_cast. Если вы будете конвертировать Parent* в Child*, то операция будет «успешной», даже если указатель класса Parent не будет указывать на объект класса Child. А сюрприз вы получите тогда, когда попытаетесь получить доступ к этому указателю (который после конвертации должен быть класса Child, но, фактически, указывает на объект класса Parent).

Если вы абсолютно уверены, что операция с понижающим приведением указателя будет успешна, то использование static_cast является приемлемым. Один из способов убедиться в этом — использовать виртуальную функцию:

```
1  #include <iostream>
```

```
#include <string>
// Идентификаторы классов
enum ClassID
{
PARENT,
CHILD
// Здесь можно добавить еще несколько классов
};
class Parent
{
protected:
int m_value;
public:
Parent(int value)
: m_value(value)
{
}
virtual ~Parent() {}
virtual ClassID getClassID() { return PARENT; }
};
class Child: public Parent
{
protected:
std::string m_name;
public:
Child(int value, std::string name)
: Parent(value), m_name(name)
```

```
29
30 {
31 }
32 std::string& getName() { return m_name; }
33 virtual ClassID getClassID() { return CHILD; }
34 };
35 Parent* getObject(bool bReturnChild)
36 {
37 if (bReturnChild)
38 return new Child(1, "Banana");
39 else
40 return new Parent(2);
41 }
42 int main()
43 {
44 Parent *p = getObject(true);
45 if (p->getClassID() == CHILD)
46 {
47 // Мы уже доказали, что p указывает на объект класса Child, поэтому
48 никаких проблем здесь не должно быть
49 Child *ch = static_cast<Child*>(p);
50 std::cout << "The name of the Child is: " << ch->getName() << '\n';
51 }
52 delete p;
53 return 0;
54 }
55
```

```
56
57
58
59
60
61
62
63
64
65
```

Но, если вы не уверены в успешности конвертации и не хотите заморачиваться с проверкой через виртуальные функции, вы можете просто использовать оператор dynamic_cast.

Оператор dynamic_cast и Ссылки

Хотя во всех примерах, приведенных выше, мы использовали динамическое приведение с указателями (что является наиболее распространенным), оператор dynamic_cast также может использоваться и со **ссылками**. Работа dynamic_cast со ссылками аналогична работе с указателями:

```
1  #include <iostream>
2  #include <string>
3  class Parent
4  {
5  protected:
6  int m_value;
7  public:
8  Parent(int value)
9  : m_value(value)
10 {
```

```
11
12
13 }
14 virtual ~Parent() {}
15 };
16 class Child: public Parent
17 {
18 protected:
19 std::string m_name;
20 public:
21 Child(int value, std::string name)
22 : Parent(value), m_name(name)
23 {
24 }
25 const std::string& getName() { return m_name; }
26 };
27 int main()
28 {
29 Child banana(1, "Banana");
30 Parent &p = banana;
31 Child &ch = dynamic_cast<Child&>(p); // используем оператор
32 dynamic_cast для конвертации ссылки класса Parent в ссылку класса Child
33 std::cout << "The name of the Child is: " << ch.getName() << '\n';
34 return 0;
35 }
36
37
```

```
38

39

40

41
```

Поскольку в языке C++ не существует «нулевой ссылки», то dynamic_cast не может возвратить «нулевую ссылку» при сбое. Вместо этого, dynamic_cast генерирует исключение типа std::bad_cast (мы поговорим об исключениях чуть позже).

Оператор dynamic_cast vs. Оператор static_cast

Начинающие программисты путают, в каких случаях следует использовать static_cast, а в каких — dynamic_cast. Ответ довольно прост: **используйте оператор dynamic_cast при понижающем приведении, а во всех остальных случаях используйте оператор static_cast**. Однако, вам также следует рассматривать возможность использования виртуальных функций вместо операторов преобразования типов данных.

Понижающее приведение vs. Виртуальные функции

Есть программисты, которые считают, что dynamic_cast — это зло и моветон. Они же советуют использовать виртуальные функции вместо оператора dynamic_cast.

В общем, использование виртуальных функций ***должно*** быть предпочтительнее использования понижающего приведения. Однако **в следующих случаях понижающее приведение является лучшим вариантом**:

Если вы не можете изменить родительский класс, чтобы добавить в него свою виртуальную функцию (например, если родительский класс является частью Стандартной библиотеки C++). При этом, чтобы использовать понижающее приведение, в родительском классе должны уже присутствовать виртуальные функции.

Если вам нужен доступ к чему-либо, что есть только в дочернем классе (например, к **функции доступа**, которая существует только в дочернем классе).

Если добавление виртуальной функции в родительский класс не имеет смысла. В таком случае, в качестве альтернативы, если вам не нужно создавать объект родительского класса, вы можете использовать **чистую виртуальную функцию**.

Оценить статью:

Загрузка...