Оператор static_cast

corob-msft

- 04.11.2016
- Чтение занимает 3 мин

В этой статье

1. Синтаксис
2. Remarks
3. См. также раздел

Преобразует *выражение* в тип *типа-ID*, основанный только на типах, имеющихся в выражении.Converts an *expression* to the type of *type-id*, based only on the types that are present in the expression.

СинтаксисSyntax

```
static_cast <type-id> ( expression )
```

RemarksRemarks

В стандартном языке C++, проверка типа во время выполнения не выполняется, что обеспечивает безопасность преобразования.In standard C++, no run-time type check is made to help ensure the safety of the conversion. В C ++/CX выполняются проверки во время компиляции и во время выполнения.In C++/CX, a compile time and runtime check are performed. Дополнительные сведения см. в разделе Приведение.For more information, see Casting.

static_cast Оператор можно использовать для таких операций, как преобразование указателя на базовый класс в указатель на производный класс.The **static_cast** operator can be used for operations such as converting a pointer to a base class to a pointer to a derived class. Такие преобразования не всегда являются безопасными.Such conversions are not always safe.

В целом, **static_cast** Если требуется преобразовать числовые типы данных, такие как enums, в ints или ints в float, а также вы уверены, какие типы данных участвуют в преобразовании.In general you use **static_cast** when you want to convert numeric data types such as enums to ints or ints to floats, and you are certain of the data types involved in the conversion. **static_cast** преобразования не так надежны **dynamic_cast** , как преобразования, поскольку не **static_cast** выполняет проверку типов во время выполнения **dynamic_cast** .**static_cast** conversions are not as safe as **dynamic_cast** conversions, because

static_cast does no run-time type check, while **dynamic_cast** does. В случае **dynamic_cast** неоднозначного указателя произойдет сбой, а **static_cast** возвращается, как если бы ничего не возникало. это может быть опасно.A **dynamic_cast** to an ambiguous pointer will fail, while a **static_cast** returns as if nothing were wrong; this can be dangerous. Хотя **dynamic_cast** преобразования являются более безопасными, **dynamic_cast** работают только с указателями или ссылками, а проверка типов во время выполнения является дополнительной нагрузкой.Although **dynamic_cast** conversions are safer, **dynamic_cast** only works on pointers or references, and the run-time type check is an overhead. Дополнительные сведения см. в разделе оператор dynamic_cast.For more information, see dynamic_cast Operator.

В следующем примере строка `D* pd2 = static_cast<D*>(pb);` небезопасна, поскольку `D` может иметь поля и методы, не входящие в `B`.In the example that follows, the line `D* pd2 = static_cast<D*>(pb);` is not safe because `D` can have fields and methods that are not in `B`. Однако строка `B* pb2 = static_cast<B*>(pd);` является безопасным преобразованием, поскольку `D` всегда содержит все `B`.However, the line `B* pb2 = static_cast<B*>(pd);` is a safe conversion because `D` always contains all of `B`.

```
// static_cast_Operator.cpp
// compile with: /LD
class B {};

class D : public B {};

void f(B* pb, D* pd) {
   D* pd2 = static_cast<D*>(pb);   // Not safe, D can have fields
                                   // and methods that are not in B.

   B* pb2 = static_cast<B*>(pd);   // Safe conversion, D always
                                   // contains all of B.
}
```

В отличие от dynamic_cast, при преобразовании не выполняется проверка во время выполнения **static_cast** `pb` .In contrast to dynamic_cast, no run-time check is made on the **static_cast** conversion of `pb`. Объект, на который указывает `pb`, может не быть объектом типа `D`, и в этом случае использование `*pd2` может привести ужасным последствиям.The object pointed to by `pb` may not be an object of type `D`, in which case the use of `*pd2` could be disastrous. Например, вызов функции, являющейся членом класса `D`, но не класса `B`, может привести к нарушению прав доступа.For instance, calling a function that is a member of the `D` class, but not the `B` class, could result in an access violation.

dynamic_cast Операторы и **static_cast** перемещают указатель на всю иерархию классов.The **dynamic_cast** and **static_cast** operators move a pointer throughout a class hierarchy. Однако **static_cast** полагается исключительно на информацию, предоставленную в инструкции CAST, и поэтому может быть ненадежной.However, **static_cast** relies exclusively on the information provided in the cast statement and can therefore be unsafe. Пример:For example:

```
// static_cast_Operator_2.cpp
// compile with: /LD /GR
class B {
```

```
public:
   virtual void Test(){}
};
class D : public B {};

void f(B* pb) {
   D* pd1 = dynamic_cast<D*>(pb);
   D* pd2 = static_cast<D*>(pb);
}
```

Если `pb` действительно указывает на объект типа `D`, `pd1` и `pd2` получат одно и то же значение.If `pb` really points to an object of type `D`, then `pd1` and `pd2` will get the same value. Также они получат одно и то же значение, если `pb == 0`.They will also get the same value if `pb == 0`.

Если `pb` указывает на объект типа, `B` а не на полный `D` класс, то **`dynamic_cast`** будет достаточно, чтобы вернуть ноль.If `pb` points to an object of type `B` and not to the complete `D` class, then **`dynamic_cast`** will know enough to return zero. Однако **`static_cast`** полагается на утверждение программиста, которое `pb` указывает на объект типа `D` и просто возвращает указатель на этот предполагаемый `D` объект.However, **`static_cast`** relies on the programmer's assertion that `pb` points to an object of type `D` and simply returns a pointer to that supposed `D` object.

Следовательно, **`static_cast`** может выполнить обратное преобразование неявных преобразований, в этом случае результаты будут неопределенными.Consequently, **`static_cast`** can do the inverse of implicit conversions, in which case the results are undefined. Программисту остается убедиться, что результаты **`static_cast`** преобразования являются надежными.It is left to the programmer to verify that the results of a **`static_cast`** conversion are safe.

Это поведение также применяется к типам, отличным от типов класса.This behavior also applies to types other than class types. Например, **`static_cast`** можно использовать для преобразования из типа int в **`char`** .For instance, **`static_cast`** can be used to convert from an int to a **`char`**. Однако в результате **`char`** может быть недостаточно битов для хранения всего **`int`** значения.However, the resulting **`char`** may not have enough bits to hold the entire **`int`** value. Опять же, программисту остается убедиться, что результаты **`static_cast`** преобразования являются надежными.Again, it is left to the programmer to verify that the results of a **`static_cast`** conversion are safe.

`static_cast` Оператор также можно использовать для выполнения любого неявного преобразования, включая стандартные преобразования и пользовательские преобразования.The **`static_cast`** operator can also be used to perform any implicit conversion, including standard conversions and user-defined conversions. Пример:For example:

```
// static_cast_Operator_3.cpp
// compile with: /LD /GR
typedef unsigned char BYTE;

void f() {
   char ch;
```

```
   int i = 65;
   float f = 2.5;
   double dbl;

   ch = static_cast<char>(i);   // int to char
   dbl = static_cast<double>(f);   // float to double
   i = static_cast<BYTE>(ch);
}
```

static_cast Оператор может явно преобразовать целочисленное значение в тип перечисления.The **static_cast** operator can explicitly convert an integral value to an enumeration type. Если значение типа целого не оказывается в диапазоне значений перечисления, получаемое значение перечисления не определено.If the value of the integral type does not fall within the range of enumeration values, the resulting enumeration value is undefined.

static_cast Оператор преобразует нулевое значение указателя в значение указателя null целевого типа.The **static_cast** operator converts a null pointer value to the null pointer value of the destination type.

Любое выражение может быть явно преобразовано в тип void **static_cast** оператором.Any expression can be explicitly converted to type void by the **static_cast** operator. Тип void назначения может дополнительно включать **const volatile** атрибут, или **__unaligned** .The destination void type can optionally include the **const**, **volatile**, or **__unaligned** attribute.

static_cast Оператор не может привести к отделам **const volatile** атрибуты, или **__unaligned** .The **static_cast** operator cannot cast away the **const**, **volatile**, or **__unaligned** attributes. Сведения об удалении этих атрибутов см. в разделе оператор const_cast .See const_cast Operator for information on removing these attributes.

C++/CLI: Из-за опасности возникновения непроверенных приведений на вершине повторного обнаружения сборщика мусора использование класса **static_cast** должно быть только в критическом для производительности коде, только если вы уверены, что он будет работать правильно.**C++/CLI:** Due to the danger of performing unchecked casts on top of a relocating garbage collector, the use of **static_cast** should only be in performance-critical code when you are certain it will work correctly. Если необходимо использовать **static_cast** в режиме выпуска, замените его safe_cast в отладочных сборках, чтобы убедиться в успешном выполнении.If you must use **static_cast** in release mode, substitute it with safe_cast in your debug builds to ensure success.

См. также разделSee also

Операторы приведенияCasting Operators
Ключевые словаKeywords

Обратная связь

Отправить и просмотреть отзыв по