Оператор dynamic_cast

corob-msft

- 03.02.2020
- Чтение занимает 5 мин

В этой статье

Преобразует операнд `expression` в объект типа `type-id`.Converts the operand `expression` to an object of type `type-id`.

СинтаксисSyntax

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

RemarksRemarks

Параметр `type-id` должен быть указателем или ссылкой на ранее определенный тип класса или "указателем на void".The `type-id` must be a pointer or a reference to a previously defined class type or a "pointer to void". Тип операнда `expression` должен быть указателем, если `type-id` является указателем, или l-значением, если `type-id` является ссылкой.The type of `expression` must be a pointer if `type-id` is a pointer, or an l-value if `type-id` is a reference.

Описание различий между преобразованиями статического и динамического приведения см. в разделе static_cast .See static_cast for an explanation of the difference between static and dynamic casting conversions, and when it is appropriate to use each.

Существует два критических изменения в работе **`dynamic_cast`** управляемого кода:There are two breaking changes in the behavior of **`dynamic_cast`** in managed code:

- **`dynamic_cast`** Указатель на базовый тип упакованного перечисления завершится ошибкой во время выполнения, возвращая 0 вместо преобразованного указателя.**`dynamic_cast`** to a pointer to the underlying type of a boxed enum will fail at runtime, returning 0 instead of the converted pointer.

- **`dynamic_cast`** больше не создает исключение `type-id` , когда является внутренним указателем на тип значения, при этом операция приведения завершается сбоем во время выполнения.**`dynamic_cast`** will no longer throw an exception when `type-id` is an interior pointer to a value type, with the cast failing at runtime. Приведение теперь возвращает значение указателя 0, а исключение не создается.The cast will now return the 0 pointer value instead of throwing.

Если `type-id` является указателем на однозначно доступный прямой или косвенный базовый класс операнда `expression`, то результатом будет указатель на уникальный подобъект типа `type-id`.If `type-id` is a pointer to an unambiguous accessible direct or indirect base class of `expression`, a pointer to the unique subobject of type `type-id` is the result. Пример:For example:

```
// dynamic_cast_1.cpp
// compile with: /c
class B { };
class C : public B { };
class D : public C { };

void f(D* pd) {
   C* pc = dynamic_cast<C*>(pd);   // ok: C is a direct base class
                                   // pc points to C subobject of pd
   B* pb = dynamic_cast<B*>(pd);   // ok: B is an indirect base class
                                   // pb points to B subobject of pd
}
```

Этот тип преобразования называется "восходящим приведением типа", поскольку при нем указатель перемещается вверх по иерархии классов: от производного класса к классу, от которого он является производным.This type of conversion is called an "upcast" because it moves a pointer up a class hierarchy, from a derived class to a class it is derived from. Восходящее приведение типа является неявным преобразованием.An upcast is an implicit conversion.

Если `type-id` является указателем void*, выполняется проверка во время выполнения, чтобы определить фактический тип операнда `expression`.If `type-id` is void*, a run-time check is made to determine the actual type of `expression`. Результатом является указатель на полный объект, на который указывает операнд `expression`.The result is a pointer to the complete object pointed to by `expression`. Пример:For example:

```
// dynamic_cast_2.cpp
// compile with: /c /GR
class A {virtual void f();};
class B {virtual void f();};

void f() {
   A* pa = new A;
   B* pb = new B;
   void* pv = dynamic_cast<void*>(pa);
   // pv now points to an object of type A

   pv = dynamic_cast<void*>(pb);
   // pv now points to an object of type B
}
```

Если `type-id` не является указателем void*, то выполняется проверка во время выполнения, чтобы определить, может ли объект, на который указывает операнд `expression`, быть преобразован в тип, указанный параметром `type-id`.If `type-id` is not void*, a run-time check is made to see if the object pointed to by `expression` can be converted to the type pointed to by `type-id`.

Если тип операнда `expression` является базовым классом типа, заданного параметром `type-id`, то выполняется проверка во время выполнения, чтобы определить, указывает ли операнд `expression` на полный объект типа, заданного параметром `type-id`.If the type of `expression` is a base class of the type of `type-id`, a run-time check is made to see if `expression` actually points to a complete object of the type of `type-id`. Если это так, то результатом является указатель на полный объект типа, заданного параметром `type-id`.If this is true, the result is a pointer to a complete object of the type of `type-id`. Пример:For example:

```
// dynamic_cast_3.cpp
// compile with: /c /GR
class B {virtual void f();};
class D : public B {virtual void f();};

void f() {
   B* pb = new D;   // unclear but ok
   B* pb2 = new B;

   D* pd = dynamic_cast<D*>(pb);   // ok: pb actually points to a D
   D* pd2 = dynamic_cast<D*>(pb2);   // pb2 points to a B not a D
}
```

Этот тип преобразования называется "нисходящим приведением типа", поскольку при нем указатель перемещается вниз по иерархии классов: от заданного класса к производному от него классу.This type of conversion is called a "downcast" because it moves a pointer down a class hierarchy, from a given class to a class derived from it.

В случаях множественного наследования возникают возможности для неоднозначности.In cases of multiple inheritance, possibilities for ambiguity are introduced. Рассмотрим для примера иерархию классов, показанную на следующем рисунке.Consider the class hierarchy shown in the following figure.

Для типов CLR **`dynamic_cast`** результатом является отсутствие операции, если преобразование может быть выполнено неявно, или `isinst` инструкция MSIL, выполняющая динамическую проверку и возвращающая **`nullptr`** в случае сбоя преобразования.For CLR types, **`dynamic_cast`** results in either a no-op if the conversion can be performed implicitly, or an MSIL `isinst` instruction, which performs a dynamic check and returns **`nullptr`** if the conversion fails.

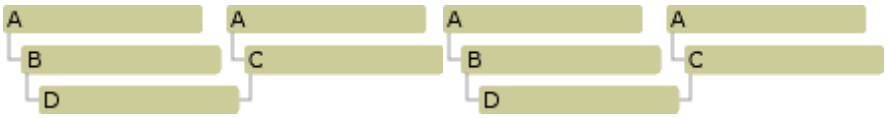
В следующем примере используется **`dynamic_cast`** , чтобы определить, является ли класс экземпляром определенного типа:The following sample uses **`dynamic_cast`** to determine if a class is an instance of particular type:

```
// dynamic_cast_clr.cpp
// compile with: /clr
using namespace System;
```

```
void PrintObjectType( Object^o ) {
   if( dynamic_cast<String^>(o) )
      Console::WriteLine("Object is a String");
   else if( dynamic_cast<int^>(o) )
      Console::WriteLine("Object is an int");
}

int main() {
   Object^o1 = "hello";
   Object^o2 = 10;

   PrintObjectType(o1);
   PrintObjectType(o2);
}
```



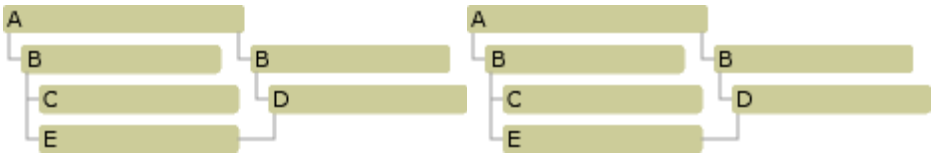
Иерархия классов, показывающая множественное наследованиеClass hierarchy that shows multiple inheritance

Указатель на объект типа `D` можно безопасно привести к `B` или `C`.A pointer to an object of type `D` can be safely cast to `B` or `C`. Однако если в результате приведения `D` указывает на объект `A`, какой экземпляр объекта `A` будет являться результатом?However, if `D` is cast to point to an `A` object, which instance of `A` would result? Это может привести к ошибке неоднозначного приведения.This would result in an ambiguous casting error. Чтобы обойти эту проблему, можно выполнить два однозначных приведения.To get around this problem, you can perform two unambiguous casts. Пример:For example:

```
// dynamic_cast_4.cpp
// compile with: /c /GR
class A {virtual void f();};
class B : public A {virtual void f();};
class C : public A {virtual void f();};
class D : public B, public C {virtual void f();};

void f() {
   D* pd = new D;
   A* pa = dynamic_cast<A*>(pd);   // C4540, ambiguous cast fails at runtime
   B* pb = dynamic_cast<B*>(pd);   // first cast to B
   A* pa2 = dynamic_cast<A*>(pb);   // ok: unambiguous
}
```

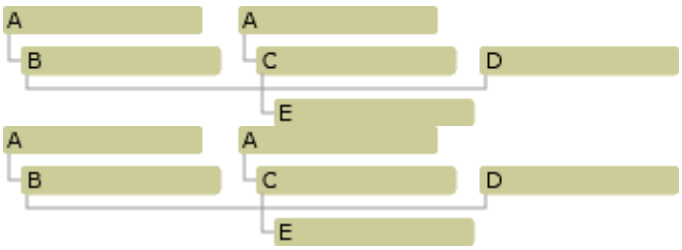
При использовании виртуальных базовых классов могут возникать дополнительные неоднозначности.Further ambiguities can be introduced when you use virtual base classes. Рассмотрим для примера иерархию классов, показанную на следующем рисунке.Consider the class hierarchy shown in the following figure.



Иерархия классов, показывающая виртуальные базовые классыClass hierarchy that shows virtual base classes

В этой иерархии объект `A` является виртуальным базовым классом.In this hierarchy, `A` is a virtual base class. При указании экземпляра класса `E` и указателя на этот `A` подобъект, в **`dynamic_cast`** указатель на `B` ошибку произойдет сбой из-за неоднозначности.Given an instance of class `E` and a pointer to the `A` subobject, a **`dynamic_cast`** to a pointer to `B` will fail due to ambiguity. Необходимо сначала выполнить обратное приведение к полному объекту `E`, затем однозначным образом вернуться вверх по иерархии, чтобы дойти до нужного объекта `B`.You must first cast back to the complete `E` object, then work your way back up the hierarchy, in an unambiguous manner, to reach the correct `B` object.

Рассмотрим для примера иерархию классов, показанную на следующем рисунке.Consider the class hierarchy shown in the following figure.



Иерархия классов, показывающая повторяющиеся базовые классыClass hierarchy that shows duplicate base classes

При использовании заданного объекта типа `E` и указателя на подобъект `D` можно выполнить три преобразования, чтобы перейти от подобъекта `D` к крайнему слева подобъекту `A`.Given an object of type `E` and a pointer to the `D` subobject, to navigate from the `D` subobject to the left-most `A` subobject, three conversions can be made. Можно выполнить **`dynamic_cast`** Преобразование из `D` указателя на `E` указатель, затем преобразование (**`dynamic_cast`** или неявное преобразование) из `E` в `B` и, наконец, неявное преобразование из `B` в `A` .You can perform a **`dynamic_cast`** conversion from the `D` pointer to an `E` pointer, then a conversion (either **`dynamic_cast`** or an implicit conversion) from `E` to `B`, and finally an implicit conversion from `B` to `A`. Пример:For example:

```
// dynamic_cast_5.cpp
// compile with: /c /GR
class A {virtual void f();};
class B : public A {virtual void f();};
class C : public A { };
class D {virtual void f();};
class E : public B, public C, public D {virtual void f();};

void f(D* pd) {
   E* pe = dynamic_cast<E*>(pd);
   B* pb = pe;   // upcast, implicit conversion
   A* pa = pb;   // upcast, implicit conversion
}
```

`dynamic_cast` Оператор также можно использовать для выполнения перекрестного приведения.The **`dynamic_cast`** operator can also be used to

perform a "cross cast." В иерархии классов из предыдущего примера можно выполнить приведение указателя, например, из подобъекта `B` в подобъект `D`, если полный объект имеет тип `E`.Using the same class hierarchy, it is possible to cast a pointer, for example, from the `B` subobject to the `D` subobject, as long as the complete object is of type `E`.

С учетом перекрестного приведения можно выполнить преобразование из указателя на объект `D` в крайний левый подобъект `A` всего за два шага.Considering cross casts, it is actually possible to do the conversion from a pointer to `D` to a pointer to the left-most `A` subobject in just two steps. Можно перекрестное приведение из `D` в `B`, а затем неявное преобразование из `B` в `A`.You can perform a cross cast from `D` to `B`, then an implicit conversion from `B` to `A`. Пример:For example:

```
// dynamic_cast_6.cpp
// compile with: /c /GR
class A {virtual void f();};
class B : public A {virtual void f();};
class C : public A { };
class D {virtual void f();};
class E : public B, public C, public D {virtual void f();};

void f(D* pd) {
   B* pb = dynamic_cast<B*>(pd);   // cross cast
   A* pa = pb;   // upcast, implicit conversion
}
```

Значение указателя NULL преобразуется в значение указателя null целевого типа **`dynamic_cast`** .A null pointer value is converted to the null pointer value of the destination type by **`dynamic_cast`**.

Если при использовании оператора `dynamic_cast < type-id > ( expression )` невозможно точно преобразовать операнд `expression` в тип `type-id`, то проверка во время выполнения приводит к сбою приведения.When you use `dynamic_cast < type-id > ( expression )`, if `expression` cannot be safely converted to type `type-id`, the run-time check causes the cast to fail. Пример:For example:

```
// dynamic_cast_7.cpp
// compile with: /c /GR
class A {virtual void f();};
class B {virtual void f();};

void f() {
   A* pa = new A;
   B* pb = dynamic_cast<B*>(pa);   // fails at runtime, not safe;
   // B not derived from A
}
```

Значением приведения к типу указателя, которое привело к сбою, является пустой указатель.The value of a failed cast to pointer type is the null pointer. Неудачное приведение к ссылочному типу вызывает исключение bad_cast.A failed cast to reference type throws a bad_cast Exception. Если не `expression` указывает на допустимый объект или ссылается на него, `__non_rtti_object` возникает исключение.If `expression` does not point to or reference a valid object,

a `__non_rtti_object` exception is thrown.

Описание typeid исключения см. в разделе typeid `__non_rtti_object` .See typeid for an explanation of the `__non_rtti_object` exception.

ПримерExample

В следующем примере создается указатель базового класса (структура A) на объект (структура C).The following sample creates the base class (struct A) pointer, to an object (struct C). Это (а также тот факт, что они являются виртуальными функциями) порождает полиморфизм времени выполнения.This, plus the fact there are virtual functions, enables runtime polymorphism.

В этом примере также вызывается невиртуальная функция в иерархии.The sample also calls a non-virtual function in the hierarchy.

```
// dynamic_cast_8.cpp
// compile with: /GR /EHsc
#include <stdio.h>
#include <iostream>

struct A {
    virtual void test() {
        printf_s("in A\n");
   }
};

struct B : A {
    virtual void test() {
        printf_s("in B\n");
    }

    void test2() {
        printf_s("test2 in B\n");
    }
};

struct C : B {
    virtual void test() {
        printf_s("in C\n");
    }

    void test2() {
        printf_s("test2 in C\n");
    }
};

void Globaltest(A& a) {
    try {
        C &c = dynamic_cast<C&>(a);
        printf_s("in GlobalTest\n");
    }
    catch(std::bad_cast) {
        printf_s("Can't cast to C\n");
    }
```

```
}

int main() {
    A *pa = new C;
    A *pa2 = new B;

    pa->test();

    B * pb = dynamic_cast<B *>(pa);
    if (pb)
        pb->test2();

    C * pc = dynamic_cast<C *>(pa2);
    if (pc)
        pc->test2();

    C ConStack;
    Globaltest(ConStack);

   // will fail because B knows nothing about C
    B BonStack;
    Globaltest(BonStack);
}
```

```
in C
test2 in B
in GlobalTest
Can't cast to C
```

См. также разделSee also

Операторы приведенияCasting Operators
Ключевые словаKeywords

Обратная связь

Отправить и просмотреть отзыв по