static_cast, dynamic_cast, reinterpret_cast, const_cast区别比较

（使用vs2010所带的编译器） 转载请注明来源 http://www.cnblogs.com/jerry19880126/

隐式转换（implicit conversion）

short a=2000;

int b;

b=a;

short是两字节，int是四字节，由short型转成int型是宽化转换（bit位数增多），编译器没有warning，如下图所示。宽化转换（如char到int，int到long long，int到float，float到double，int到double等）构成隐式转换，编译器允许直接转换。

```
1>ClCompile:
1>  AllKindsOfCast.cpp
1>ManifestResourceCompile:
1>  所有输出均为最新。
```

但若反过来

随笔分类(86)

随笔档案(86)

double a=2000;

short b;

b=a;

此时，是从8字节的double型转成2字节的short型变量，是窄化转换，编译器就会有warning了，如下所示，提醒程序员可能丢失数据。不过需要注意的是，有些隐式转换，编译器可能并不给出warning，比如int到short，但数据溢出却依然会发生。

```
1>ClCompile:
1>  AllKindsOfCast.cpp
1>d:\my_cpp\cast\allkindsofcast.cpp(14): warning C4244: “=”：从“double”转换到“short”，可能丢失数据
1>ManifestResourceCompile:
1>  所有输出均为最新。
```

C风格显式转换（C style explicit conversion）

要去掉上述waring很简单，熟悉C语言的程序员知道，有两种简单的写法（C风格转换与函数风格转换）：

double a=2000.3;

short b;

b = (short) a; // c-like cast notation

b = short (a); // functional notation

如下图所示，此时warning就没了

积分与排名

积分 - 118084

排名 - 3026

最新评论

1. Re:读书笔记_Effective_C++_条款三十一：将文件间的编译依存关系降至最低（第一部分）

真透彻

--Mr_青山君

2. Re:条件覆盖，路径覆盖，语句覆盖，分支覆盖

大大 你好。如果例子为：if A and B then Action1 else Action2if C or D then Action3语句覆盖中，设计的测试用例应该是什么？主要是我不明白if-e.....

--wwachi

3. Re:读书笔记_代码大全_第18章_表驱动法

最后一段代码，score小于中值是不是应该把high变成mid - 1？这里反了吧？

--imanuqiao

4. Re:C++类内存分布

文章很好，很形象，能方便解释一下，接口，在内存中的存在形式吗

--雪域阳光

5. Re:读书笔记_Effective_C++_条款三十九：明智而审慎地使用private继承

理解太到位了，大牛

```
1>ClCompile:
1>  AllKindsOfCast.cpp
1>ManifestResourceCompile:
1>  所有输出均为最新。
```

这种显式转换方式简单直观，但并不安全，举一个父类和子类的例子如下：

```
1 // class type-casting
2 #include <iostream>
3 using namespace std;
4
5 class CDummy {
6 float i,j;
7 CDummy():i(100),j(10){}
8 };
9
10 class CAddition: public CDummy
11 {
12     int *x,y;
13   public:
14     CAddition (int a, int b) { x=&a; y=b; }
15     int result() { return *x+y;}
16 };
17
18 int main () {
19   CDummy d;
20   CAddition * padd;
21   padd = (CAddition*) &d;
22   cout << padd->result();
23   return 0;
24 }
```

--繁星的夜空2012

阅读排行榜

1. 腾讯2012实习生笔试题+答案解析(40895)
2. C++类内存分布(26772)
3. 腾讯2012实习生笔试题2+答案解析(21259)
4. static_cast, dynamic_cast, reinterpret_cast, const_cast区别比较(14617)
5. 阿里巴巴2011笔试题+答案解析(13525)

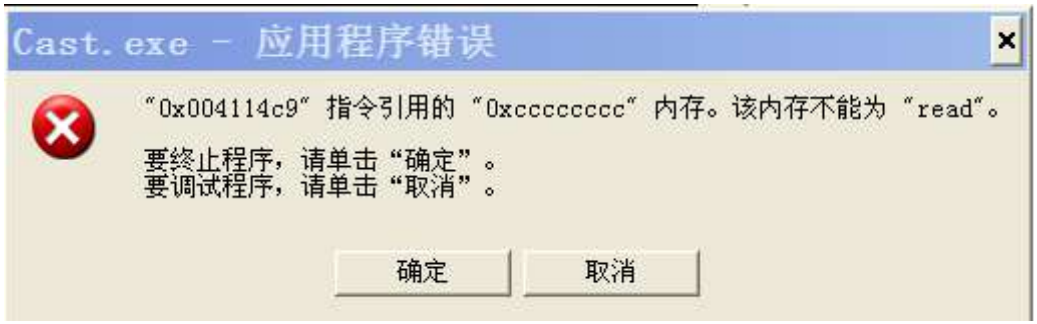
评论排行榜

1. 腾讯2012实习生笔试题+答案解析(45)
2. 腾讯2012实习生笔试题2+答案解析(34)
3. 微软2012实习生笔试题+答案解析(7)
4. C++类内存分布(5)
5. 读书笔记_Effective_C++_条款三十：了解inline的里里外外(5)

推荐排行榜

1. static_cast, dynamic_cast, reinterpret_cast, const_cast区别比较(18)
2. C++类内存分布(16)

编译器不报任何错，但运行结果出错，如下图所示：



究其原因，注意这一句：padd = (CAddition*) &d;

此时父类的指针&d被C风格转换方式强制转成了子类的指针了，后面调用了子类的方法result，需要访问*x，但指针指向的对象本质还是父类的，所以x相当于父类中的i，y相当于父类中的j，*x相当于*i，但i是float型变量（初始化为100），不是地址，所以出错，如果程序员正是鲁莽地对这个地址指向的内存进行写入操作，那将可能会破坏系统程序，导致操作系统崩溃！

这里有一个重要概念，CAddition*是子类的指针，它的变量padd可以调用子类的方法，但是它指向的是父类的对象，也就是说padd指向的内存空间里存放的是父类的成员变量。深入地说，数据在内存中是没有“类型”一说的，比如0x3F可能是字符型，也可能是整型的一部分，还可能是地址的一部分。我们定义的变量类型，其实就是定义了数据应该“被看成什么”的方式。

因此padd类指针实质是定义了取值的方式，如padd->x就是一并取出内存空间里的0号单元至3号单元的值（共4个字节），将其拼成32位并当作指针，padd->y则取出内存空间里的4号单元至7号单元（共4个字节），将其拼成32位并当作int型变量。但实际上padd指向的是父类的对象，也就是前4个字节是float型变量，后4个字节也是float型变量。

从这里可以看出，程序员的这种转换使编译器“理解”出错，把牛当成马了。

从上可见，用C风格的转换其实是不安全的，编译器无法看到转换的不安全。

上行转换（up-casting）与下行转换（down-casting）

看到这个，读者可能会问，哪些转换不安全？根据前面所举的例子，可以看到，不安全来源于两个方面：其一是类型的窄化转化，会导致数据位数的丢失；其二是在类继承链中，将父类对象的地址（指针）强制转化成子类的地址（指针），这就是所谓的下行转换。“下”表示沿着继承链向下走（向子类的方向走）。

类似地，上行转换的“上”表示沿继承链向上走（向父类的方向走）。

我们给出结论，上行转换一般是安全的，下行转换很可能是不安全的。

为什么呢？因为子类中包含父类，所以上行转换（只能调用父类的方法，引用父类的成员变量）一般是安全的。但父类中却没有子类的任何信息，而下行转换会调用到子类的方法、引用子类的成员变量，这些父类都没有，所以很容易“指鹿为马”或者干脆指向不存在的内存空间。

值得一说的是，不安全的转换不一定会导致程序出错，比如一些窄化转换在很多场合都会被频繁地使用，前提是程序员足够小心以防止数据溢出；下行转换关键看其“本质”是什么，比如一个父类指针指向子类，再将这个父类指针转成子类指针，这种下行转换就不会有问题。

针对类指针的问题，C++特别设计了更加细致的转换方法，分别有：

static_cast <new_type> (expression)
dynamic_cast <new_type> (expression)
reinterpret_cast <new_type> (expression)
const_cast <new_type> (expression)

可以提升转换的安全性。

static_cast <new_type> (expression) 静态转换

静态转换是最接近于C风格转换，很多时候都需要程序员自身去判断转换是否安全。比如：

double d=3.14159265;

int i = static_cast<int>(d);

但static_cast已经有安全性的考虑了，比如对于不相关类指针之间的转换。参见下面的例子：

```
// class type-casting
#include <iostream>
using namespace std;

class CDummy {
    float i,j;
};

class CAddition {
    int x,y;
  public:
    CAddition (int a, int b) { x=a; y=b; }
    int result() { return x+y;}
};

int main () {
  CDummy d;
  CAddition * padd;
  padd = (CAddition*) &d;
  cout << padd->result();
```

```
21    return 0;
22 }
```

这个例子与之前举的例子很像，只是CAddition与CDummy类没有任何关系了，但main()中C风格的转换仍是允许的padd = (CAddition*) &d，这样的转换没有安全性可言。

如果在main()中使用static_cast，像这样：

```
1  int main () {
2    CDummy d;
3    CAddition * padd;
4    padd = static_cast<CAddition*> (&d);
5    cout << padd->result();
6    return 0;
7 }
```

编译器就能看到这种不相关类指针转换的不安全，报出如下图所示的错误：

```
1>  AllKindsOfCast.cpp
1>d:\my_cpp\cast\allkindsofcast.cpp(49): error C2440: “static_cast”: 无法从“CDummy *”转换为“CAddition *”
1>          与指向的类型无关；转换要求 reinterpret_cast、C 样式转换或函数样式转换
1>
1>生成失败。
```

注意这时不是以warning形式给出的，而直接是不可通过编译的error。从提示信息里可以看到，编译器说如果需要这种强制转换，要使用reinterpret_cast（稍候会说）或者C风格的两种转换。

总结一下：static_cast最接近于C风格转换了，但在无关类的类指针之间转换上，有安全性的提升。

dynamic_cast <new_type> (expression) 动态转换

动态转换确保类指针的转换是合适完整的，它有两个重要的约束条件，其一是要求new_type为指针或引用，其二是下行转换时要求基类是多态的（基类中包含至少一个虚函数）。

看一下下面的例子：

```
1 #include <iostream>
2 using namespace std;
3 class CBase { };
4 class CDerived: public CBase { };
5
6 int main()
7 {
8 CBase b; CBase* pb;
9 CDerived d; CDerived* pd;
10
11 pb = dynamic_cast<CBase*>(&d);      // ok: derived-to-base
12 pd = dynamic_cast<CDerived*>(&b);   // wrong: base-to-derived
13 }
```

在最后一行代码有问题，编译器给的错误提示如下图所示：

```
1>ClCompile:
1>  AllKindsOfCast.cpp
1>d:\my_cpp\cast\allkindsofcast.cpp(64): error C2683: “dynamic_cast”: “CBase”不是多态类型
1>          d:\my_cpp\cast\allkindsofcast.cpp(55) : 参见“CBase”的声明
1>
1>生成失败。
```

把类的定义改成：

class CBase { virtual void dummy() {} };

class CDerived: public CBase {};

再编译，结果如下图所示：

```
1>ClCompile:
1>  AllKindsOfCast.cpp
1>ManifestResourceCompile:
1>  所有输出均为最新。
```

编译都可以顺利通过了。这里我们在main函数的最后添加两句话：

cout << pb << endl;

cout << pd << endl;

输出pb和pd的指针值，结果如下：

我们看到一个奇怪的现象，将父类经过dynamic_cast转成子类的指针竟然是空指针！这正是dynamic_cast提升安全性的功能，dynamic_cast可以识别出不安全的下行转换，但并不抛出异常，而是

将转换的结果设置成null（空指针）。

再举一个例子：

```
#include <iostream>
#include <exception>
using namespace std;

class CBase { virtual void dummy() {} };
class CDerived: public CBase { int a; };

int main () {
  try {
    CBase * pba = new CDerived;
    CBase * pbb = new CBase;
    CDerived * pd;

    pd = dynamic_cast<CDerived*>(pba);
    if (pd==0) cout << "Null pointer on first type-cast" << endl;

    pd = dynamic_cast<CDerived*>(pbb);
    if (pd==0) cout << "Null pointer on second type-cast" << endl;

  } catch (exception& e) {cout << "Exception: " << e.what();}
  return 0;
}
```

输出结果是：Null pointer on second type-cast

两个dynamic_cast都是下行转换，第一个转换是安全的，因为指向对象的本质是子类，转换的结果使子类指针指向子类，天经地义；第二个转换是不安全的，因为指向对象的本质是父类，“指鹿为马”或指向不存在

的空间很可能发生!

最后补充一个特殊情况，当待转换指针是void*或者转换目标指针是void*时，dynamic_cast总是认为是安全的，举个例子：

```
#include <iostream>
using namespace std;
class A {virtual void f(){}};
class B {virtual void f(){}};

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

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

运行结果如下：

可见dynamic_cast认为空指针的转换安全的，但这里类A和类B必须是多态的，包含虚函数，若不是，则会编译报错。

reinterpret_cast <new_type> (expression) 重解释转换

这个转换是最"不安全"的，两个没有任何关系的类指针之间转换都可以用这个转换实现，举个例子：

class A {};

class B {};

A * a = new A;

B * b = reinterpret_cast<B*>(a);//correct!

更厉害的是，reinterpret_cast可以把整型数转换成地址（指针），这种转换在系统底层的操作，有极强的平台依赖性，移植性不好。

它同样要求new_type是指针或引用，下面的例子是通不过编译的：

double a=2000.3;

short b;

b = reinterpret_cast<short> (a); //compile error!

const_cast <new_type> (expression) 常量向非常量转换

这个转换好理解，可以将常量转成非常量。

```
1 // const_cast
2 #include <iostream>
3 using namespace std;
```

```
 4
 5 void print (char * str)
 6 {
 7   cout << str << endl;
 8 }
 9
10 int main () {
11   const char * c = "sample text";
12   char *cc = const_cast<char *> (c) ;
13   Print(cc);
14   return 0;
15 }
```

从char *cc = const_cast<char *>(c)可以看出了这个转换的作用了，但切记，这个转换并不转换原常量本身，即c还是常量，只是它返回的结果cc是非常量了。

总结

C风格转换是“万能的转换”，但需要程序员把握转换的安全性，编译器无能为力；static_cast最接近于C风格转换，但在无关类指针转换时，编译器会报错，提升了安全性；dynamic_cast要求转换类型必须是指针或引用，且在下行转换时要求基类是多态的，如果发现下行转换不安全，dynamic_cast返回一个null指针，dynamic_cast总是认为void*之间的转换是安全的；reinterpret_cast可以对无关类指针进行转换，甚至可以直接将整型值转成指针，这种转换是底层的，有较强的平台依赖性，可移植性差；const_cast可以将常量转成非常量，但不会破坏原常量的const属性，只是返回一个去掉const的变量。

注：本文中大部分样例来源自C++标准网站：

http://www.cplusplus.com/doc/tutorial/typecasting/

以及微软的MSDN：

http://msdn.microsoft.com/en-us/library/cby9kycs

若有理解出错的地方，望不吝指正。

分类: C++语法

Jerry19880126
关注 - 3
粉丝 - 217
+加关注

18 0

« 上一篇: 搜狗2011笔试题+答案解析
» 下一篇: 华为2012.09.03浙大机试题

posted @ 2012-08-14 16:06 Jerry19880126 阅读(14617) 评论(2) 编辑 收藏

评论列表

#1楼 2014-07-28 18:07 行者无疆!! 回复 引用

总结的很给力，尤其本人不太清楚的dynamic_cast部分，博主给力，已点推荐和收藏，多谢

支持(0) 反对(0)

#2楼 2015-04-04 18:00 ashleylqx 回复 引用

博主总结的真的很好，调理很清晰，例子也很到位，让我学到了很多！
关于reinterpret_cast我有一个小小的疑问，博主说它要求new_type是指针或者引用，并且举了个例子。但是我在书上看到过reinterpret_cast<int>的用法，也在另一篇博文上看到"reinterpret_cast可以转换任意一个32bit整数，包括所有的指针和整数。可以把任何整数转成指针，也可以把任何指针转成整数，以及把指针转化为任意类型的指针"这样的说法。所以我想博主在这点上是不是有点误解？例子编译没通过，是因为double类型不是32bit的？
一点小疑惑，说的不对还请博主多多包涵。

支持(0) 反对(0)