程序设计实习

郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431
刘家瑛 微博 http://weibo.com/pkuliujiaying

强制类型转换

static_cast、interpret_cast const_cast和dynamic_cast

1. static_cast

static_cast用来进用行比较“自然”和低风险的转换，比如整型和实数型、字符型之间互相转换。

static_cast不能来在不同类型的指针之间互相转换，也不能用于整型和指针之间的互相转换，也不能用于不同类型的引用之间的转换。

```
//static_cast示例
#include <iostream>
using namespace std;
class A
{
public:
        operator int()          { return 1; }
        operator char * (){ return NULL; }
};
int main()
{
        A a;
        int n; char * p = "New Dragon Inn";
        n = static_cast<int>(3.14);  // n 的值变为 3
        n = static_cast<int>(a);     //调用a.operator int, n的值变为 1
```

```
    p = static_cast<char*>(a);   //调用a.operator char *,p的值变为 NULL
    n = static_cast<int> (p);    //编译错误，static_cast不能将指针转换成整型
    p = static_cast<char*>(n);   //编译错误，static_cast不能将整型转换成指针
    return 0;
}
```

- 2. reinterpret_cast

reinterpret_cast用来进行各种不同类型的指针之间的转换、不同类型的引用之间转换、以及指针和能容纳得下指针的整数类型之间的转换。转换的时候，执行的是逐个比特拷贝的操作。

```
//reinterpret_cast示例
#include <iostream>
using namespace std;
class A
{
  public:
            int i;
            int j;
            A(int n):i(n),j(n) { }
};
int main()
{
             A a(100);
             int & r = reinterpret_cast<int&>(a); //强行让 r 引用 a
             r = 200;  //把 a.i 变成了 200
             cout << a.i << "," << a.j << endl;  // 输出 200,100
             int n = 300;
```

```
    A * pa = reinterpret_cast<A*> ( & n); //强行让 pa 指向 n
    pa->i = 400;        // n 变成 400
    pa->j = 500;  //此条语句不安全，很可能导致程序崩溃
    cout << n << endl;  // 输出 400
    long long la = 0x12345678abcdLL;
    pa = reinterpret_cast<A*>(la); // la太长，只取低32位0x5678abcd拷贝给pa
    unsigned int u = reinterpret_cast<unsigned int>(pa); //pa逐个比特拷贝到u
    cout << hex << u << endl;  //输出 5678abcd
    typedef void (* PF1) (int);
    typedef int (* PF2) (int,char *);
    PF1 pf1;              PF2 pf2;
    pf2 = reinterpret_cast<PF2>(pf1); //两个不同类型的函数指针之间可以互相转换
}
```

输出结果

200, 100

400

5678abcd

- 3. const_cast

用来进行去除const属性的转换。将const引用转换成同类型的非const引用，将const指针转换为同类型的非const指针时用它。例如：

```
const string s = “Inception”;
string & p = const_cast<string&>(s);
string * ps = const_cast<string*>(&s); // &s的类型是const string *
```

4. dynamic_cast

- dynamic_cast专门用于将多态基类的指针或引用，强制转换为派生类的指针或引用，而且能够检查转换的安全性。对于不安全的指针转换，转换结果返回NULL指针。
- dynamic_cast不能用于将非多态基类的指针或引用，强制转换为派生类的指针或引用

```
//dynamic_cast示例
#include <iostream>
#include <string>
using namespace std;
class Base
{ //有虚函数，因此是多态基类
  public:
              virtual ~Base() { }
};
class Derived:public Base  {  };
int main()
{
            Base b;
            Derived d;
            Derived * pd;
            pd = reinterpret_cast<Derived*> ( &b);
```

```
        if( pd == NULL) //此处pd不会为NULL。
        //reinterpret_cast不检查安全性，总是进行转换
                cout << "unsafe reinterpret_cast" << endl;
                //不会执行
        pd = dynamic_cast<Derived*> ( &b);
        if( pd == NULL)
//结果会是NULL，因为 &b不是指向派生类对象，此转换不安全
             cout << "unsafe dynamic_cast1" << endl;  //会执行
        Base * pb = & d;
        pd = dynamic_cast<Derived*> ( pb); //安全的转换
        if( pd == NULL)  //此处pd 不会为NULL
              cout << "unsafe dynamic_cast2" << endl; //不会执行
        return 0;
}
```

输出结果：

unsafe dynamic_cast1

```
Derived & r = dynamic_cast<Derived&>(b);
```

那该如何判断该转换是否安全呢？

答案：不安全则抛出异常

In-Video Quiz

以下哪段程序是编译不会出错的？

A)struct A {}; struct B:public A {}; A * pa; B* pb; pb = dynamic_cast<B*> (pa);

B)int * p = static_cast<int *> ("this");

C)string s; int * p = reinterpret_cast<int *> (& s);

D)unsigned u = static_cast<unsigned *> ("this");

In-Video Quiz

以下哪段程序是编译不会出错的？

A)struct A {}; struct B:public A {}; A * pa; B* pb; pb = dynamic_cast<B*> (pa);

B)int * p = static_cast<int *> ("this");

C)string s; int * p = reinterpret_cast<int *> (& s);

D)unsigned u = static_cast<unsigned *> ("this");

#C