

Lesson10---C++的类型转换

【本节目标】

- 1. C语言中的类型转换
- 2. C++强制类型转换
- 3. 为什么需要强制类型转换
- 4. RTTI

1. C语言中的类型转换

在C语言中，如果赋值运算符左右两侧类型不同，或者形参与实参类型不匹配，或者返回值类型与接收返回值类型不一致时，就需要发生类型转化，C语言中总共有两种形式的类型转换：隐式类型转换和显式类型转换。

1. 隐式类型转化：编译器在编译阶段自动进行，能转就转，不能转就编译失败
2. 显式类型转化：需要用户自己处理

```
1 void Test ()
2 {
3     int i = 1;
4     // 隐式类型转换
5     double d = i;
6     printf("%d, %.2f\n" , i, d);
7
8     int* p = &i;
9     // 显示的强制类型转换
10    int address = (int) p;
11
12    printf("%x, %d\n" , p, address);
13 }
```

缺陷：转换的可视性比较差，所有的转换形式都是以一种相同形式书写，难以跟踪错误的转换

2. 为什么C++需要四种类型转换

C风格的转换格式很简单，但是有不少缺点的：

1. 隐式类型转化有些情况下可能会出问题：比如数据精度丢失
2. 显式类型转换将所有情况混合在一起，代码不够清晰

因此C++提出了自己的类型转化风格，注意因为C++要兼容C语言，所以C++中还可以使用C语言的转化风格。

3. C++强制类型转换

标准C++为了加强类型转换的可视性，引入了四种命名的强制类型转换操作符：

static_cast、reinterpret_cast、const_cast、dynamic_cast

3.1 static_cast

`static_cast`用于非多态类型的转换（静态转换），编译器隐式执行的任何类型转换都可用`static_cast`，但它不能用于两个不相关的类型进行转换

```
1  int main()
2  {
3      double d = 12.34;
4      int a = static_cast<int>(d);
5      cout<<a<<endl;
6      return 0;
7  }
```

3.2 reinterpret_cast

`reinterpret_cast`操作符通常为操作数的位模式提供较低层次的重新解释，用于将一种类型转换为另一种不同的类型

```
1  typedef void (* FUNC)();
2  int DoSomething (int i)
3  {
4      cout<<"DoSomething" <<endl;
5      return 0;
6  }
7
8  void Test ()
9  {
10     //
11     // reinterpret_cast可以编译器以FUNC的定义方式去看待DoSomething函数
12     // 所以非常的BUG，下面转换函数指针的代码是不可移植的，所以不建议这样用
13     // C++不保证所有的函数指针都被一样的使用，所以这样用有时会产生不确定的结果
14     //
15     FUNC f = reinterpret_cast< FUNC>(DoSomething );
16     f();
17 }
```

3.3 const_cast

`const_cast`最常用的用途就是删除变量的`const`属性，方便赋值

```

1 void Test ()
2 {
3     const int a = 2;
4     int* p = const_cast< int*>(&a );
5     *p = 3;
6
7     cout<<a <<endl;
8 }

```

3.4 dynamic_cast

dynamic_cast用于将一个父类对象的指针/引用转换为子类对象的指针或引用(动态转换)

向上转型：子类对象指针/引用->父类指针/引用(不需要转换，赋值兼容规则) 向下转型：父类对象指针/引用->子类指针/引用(用dynamic_cast转型是安全的)

注意： 1. dynamic_cast只能用于含有虚函数的类 2. dynamic_cast会先检查是否能转换成功，能成功则转换，不能则返回0

```

1 class A
2 {
3 public :
4     virtual void f(){}
5 };
6
7 class B : public A
8 {};
9
10 void fun (A* pa)
11 {
12     // dynamic_cast会先检查是否能转换成功，能成功则转换，不能则返回
13     B* pb1 = static_cast<B*>(pa);
14     B* pb2 = dynamic_cast<B*>(pa);
15
16     cout<<"pb1:" <<pb1<< endl;
17     cout<<"pb2:" <<pb2<< endl;
18 }
19
20 int main ()
21 {
22     A a;
23     B b;
24     fun(&a);
25     fun(&b);
26     return 0;
27 }

```

注意 强制类型转换关闭或挂起了正常的类型检查，每次使用强制类型转换前，程序员应该仔细考虑是否还有其他不同的方法达到同一目的，如果非强制类型转换不可，则应限制强制转换值的作用域，以减少发生错误的机会。**强烈建议：避免使用强制类型转换**

扩展阅读: <http://www.zhihu.com/question/22445339/answer/21386127>

3.5 explicit

explicit关键字阻止经过转换构造函数进行的隐式转换的发生

```
1  class A
2  {
3  public :
4      explicit A (int a)
5      {
6          cout<<"A(int a)" <<endl;
7      }
8
9      A(const A& a)
10     {
11         cout<<"A(const A& a)" <<endl;
12     }
13 private :
14     int _a ;
15 };
16
17 int main ()
18 {
19     A a1 (1);
20
21     // 隐式转换-> A tmp(1); A a2(tmp);
22     A a2 = 1;
23 }
```

4. RTTI

RTTI: Run-time Type identification的简称, 即: 运行时类型识别。

C++通过以下方式支持RTTI:

1. typeid运算符
2. dynamic_cast运算符

5. 常见面试题

1. C++中的4中类型转化分别是: _____、_____、_____、_____
2. 说说4中类型转化的应用场景。