C++隐式类型转换

什么是隐式转换？

众所周知，C++的基本类型中并非完全的对立，部分数据类型之间是可以进行隐式转换的。

所谓隐式转换，是指不需要用户干预，编译器私下进行的类型转换行为。很多时候用户可能都不知道进行了哪些转换。

为什么要进行隐式转换？

C++面向对象的多态特性，就是通过父类的类型实现对子类的封装。

通过隐式转换，你可以直接将一个子类的对象使用父类的类型进行返回。

在比如，数值和布尔类型的转换，整数和浮点数的转换等。

某些方面来说，隐式转换给C++程序开发者带来了不小的便捷。

C++是一门强类型语言，类型的检查是非常严格的。

如果没有类型的隐式转换，这将给程序开发者带来很多的不便。

Post Archives
2013/10 (1)
2012/7 (5)
2010/12 (1)

Gallery
博客配图

Recent Comments
1. Re:编程珠玑（二）：字符反转--杂耍算法
很好，谢谢
--overflow
2. Re:编程珠玑（五）：寻找变位词
/************************************
************************************ >
File Name: test5.cpp.....
--神奕

Top Posts
1. 编程珠玑（一）：前言 && 位图排序(2026)
2. 编程珠玑（二）：字符反转--杂耍算法(1742)
3. 编程珠玑（三）：字符反转--块变换(1342)
4. 编程珠玑（五）：寻找变位词(1174)
5. 编程珠玑（四）：字符反转--求

当然，凡事都有两面性，在你享受方便快捷的一面时，你不得不面对太过智能以至完全超出了你的控制。

风险就在不知不觉间出现。

C++隐式转换的原则

- **基本数据类型 基本数据类型的转换以取值范围的作为转换基础（保证精度不丢失）。**
  隐式转换发生在从小->大的转换中。比如从char转换为int。
  从int-》long。
- **自定义对象 子类对象可以隐式的转换为父类对象。**

C++隐式转换发生条件

- **混合类型的算术运算表达式中。例如：**

```
1 int a = 3;
2 double b = 4.5;
3 a + b; // a将会被自动转换为double类型，转换的结果和b进行加法操作
```

- **不同类型的赋值操作。例如：**

```
1 int a = true; (bool类型被转换为int类型)
2 int * ptr = null;(null被转换为int*类型)
```

- **函数参数传值。例如：**

```
1 void func(double a);
2 func(1); // 1被隐式的转换为double类型1.0
```

- **函数返回值。例如：**

```
1 double add(int a, int b)
```

逆算法(1071)


```
{
    return a + b;
} //运算的结果会被隐式的转换为double类型返回
```

#参考：http://developer.51cto.com/art/201002/183139.htm

#以上四种情况下的隐式转换，都满足了一个基本原则：低精度 –》 高精度转换。

不满足该原则，隐式转换是不能发生的。

当然这个时候就可以使用与之相对于的显式类型转换（又称强制类型转换），使用方法如下：

```
double a = 2.0;
int b = (int)a;
```

使用强制类型转换会导致精度的损失，因此使用时务必确保你已经拥有足够的把握。

隐式转换的风险

隐式转换的风险一般存在于自定义的类构造函数中。

按照默认规定，只有一个参数的构造函数也定义了一个隐式转换，将该构造函数对应数据类型的数据转换为该类对象。

- **例一**
  如下面所示：

```
class String
{
public:
    String ( const char* p ); // 用C风格的字符串p作为初始化值
    //…
}

```

```
String s1 = “hello”; //OK 隐式转换，等价于String s1 = String ( ”hel
```

但是有的时候可能会不需要这种隐式转换，如下：

```
class String
{
public:
    String ( int n ); //本意是预先分配n个字节给字符串
    String ( const char* p ); // 用C风格的字符串p作为初始化值

    //…
}
```

下面两种写法比较正常：
String s2 (10); //OK 分配10个字节的空字符串
String s3 = String (10); //OK 分配10个字节的空字符串

下面两种写法就比较疑惑了：
String s4 = 10; //编译通过，也是分配10个字节的空字符串
String s5 = ‘a’; //编译通过，分配int (‘a’) 个字节的空字符串
s4 和s5 分别把一个int型和char型，隐式转换成了分配若干字节的空字符串，容易令人误解。
#参考：http://blog.csdn.net/smilelance/article/details/1528737

- 例二
如下例：

```
class Test
{
public:
  Test(int a);
  bool isSame(Test other)
  {
```

```
    return m_val == other.m_val;
  }

private:
  int m_val;
}
```

如下调用：
Test a(10);
If(a.isSame(10)) //该语句将返回true

本来用于两个Test对象的比较，竟然和int类型相等了。
这里就是由于发生了隐式转换，实际比较的是一个临时的Test对象。
这个在程序中是绝对不能允许的。

禁止隐式转换

既然隐式转换存在这么多的风险，那如何能够禁止隐式转换的发生呢。

C++中提供了explicit关键字，在构造函数声明的时候加上explicit关键字，能够禁止隐式转换。使用方法如下：

```
class Test
{
explicit Test(int a);
……

}
```

加上该关键字以后，如下的操作是

```
Test(10);
```

如下的操作就变成非法的了：

```
1 Test aa = 10;
```

这样就可以有效的防止隐式转换的发生，从而能够对程序进行精确控制，达到提高品质的目的。

分类: C++

标签: C++, explicit, 隐式类型转换, 类型转换

« 上一篇：编程珠玑（五）：寻找变位词