Lesson05---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

```
 void Test ()
{
     int i = 1;
     // 隐式类型转换
     double d = i;
     printf("%d, %.2f\n" , i, d);

     int* p = &i;
     // 显示的强制类型转换
     int address = (int) p;

     printf("%x, %d\n" , p, address);
}
```

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. 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

2.1 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

```
int main()
{
    double d = 12.34;
    int a = static_cast<int>(d);
    cout<<a<<endl;
    return0;
}
```

2.2 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

```
typedef void (* FUNC)();
int DoSomething (int i)
{
    cout<<"DoSomething" <<endl;
    return 0;
}

void Test ()
{
    //
    // reinterpret_cast可以编译器以FUNC的定义方式去看待DoSomething函数
    // 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
    // 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
    //
    FUNC f = reinterpret_cast< FUNC>(DoSomething );
    f();
}
```


2.3 const_cast

> const_cast最常用的用途就是删除变量的const属性，方便赋值

```
void Test ()
{
    const int a = 2;
    int* p = const_cast< int*>(&a );
    *p = 3;

    cout<<a <<endl;
}
```

2.4 dynamic_cast

dynamic_cast用于将一个父类对象的指针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->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->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

```
class A
{
public :
    virtual void f(){}
};

class B : public A
{};

void fun (A* pa)
{
  // dynamic_cast会先检查是否能转换成功，能成功则转换，不能则返回
  B* pb1 = static_cast<B*>(pa);
  B* pb2 = dynamic_cast<B*>(pa);

  cout<<"pb1:" <<pb1<< endl;
  cout<<"pb2:" <<pb2<< endl;
}

int main ()
{
    A a;
    B b;
    fun(&a);
    fun(&b);
    return 0;
}
```

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

2.5 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

```
class A
{
public :
    explicit A (int a)
   {
        cout<<"A(int a)" <<endl;
```

```
    }

    A(const A& a)
    {
        cout<<"A(const A& a)" <<endl;
    }
private :
    int _a ;
};

int main ()
{
    A a1 (1);

    // 隐式转换-> A tmp(1); A a2(tmp);
    A a2 = 1;
}
```

3. 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
> 2. 显式类型转换将所有情况混合在一起，代码不够清晰

4. RTTI

> RTTI：Run-time Type identification的简称，即：运行时类型识别。
>
> C++通过以下方式来支持RTTI：
>
> 1. **typeid运算符**
> 2. **dynamic_cast运算符**

【总结】

知识块	知识点	分类	掌握程度
隐式及显式类型转换	隐式类型转换	**考点型**	**掌握**
	显式类型转换：　(类型)变量	**考点型**	**掌握**
强制类型转换	static_cast:用于非多态类型的转换（静态转换）;不能用于两个不相关的类型进行转换	**考点型**	**掌握**
	const_cast:删除变量的const属性，方便赋值	**考点型**	**掌握**
	reinterpret_cast:用于将一种类型转换为另一种不同的类型,存在较大的风险	**考点型**	**掌握**
	dynamic_cast：用于将一个父类对象的指针转换为子类对象的指针或引用(动态转换)；只能用于含有虚函数的类	**考点型**	**掌握**
explicit	阻止经过转换构造函数进行的隐式转换的发生	**考点型**	**掌握**
为什么C++需要四种类型转换	1.C风格转换太过随意，可以在任意类型之间转换 2.C风格转换没有统一的关键字和标示符，做代码排查时容易遗漏和忽略 3.C++对类型转换做了细分，提供了四种不同类型转换，以支持不同需求的转换 4.C++类型转换有了统一的标示符，利于代码排查和检视	**应用型**	**掌握**
RTTI	**运行时类型识别**的功能由 typeid运算符、dynamic_cast运算符来实现	**应用型**	熟悉
	使用场景：想使用基类对象的指针或引用执行某个派生类操作，并且该操作不是虚函数	**应用型**	熟悉

【作业】

> 写一篇博客，归纳总结本节课内容。