

[1]内联函数

[2]内联函数与带参数宏的区别

[3]新的类型转换运算符

(1) `const_cast<T>(expr)`

(2) `static_cast<T>(expr)`

(3) `reinterpret_cast<T>(expr)`

(4) `dynamic_cast<T>(expr)`

[1]内联函数

(1) 什么是内联函数呢？简单的说就是用 **inline** 关键词所修饰的函数。那么程序当中什么时候要用到内联函数呢？要明白这点首先要明白函数调用的过程，当程序执行函数调用时，系统要建立**栈空间**，**保护现场**，**传递参数**，以及**控制程序执行的转移**。这些工作需要**时间和空间**的开销，有些情况下，函数本身的功能很简单，代码很短，但是使用的频率很高，程序频繁的调用该函数所花费的时间很多，从而使得程序执行的效率很低。（以空间换时间的方法）。频繁的调用使得 CPU 的利用率不高。

(2) 为了提高效率，一种解决方法就是不使用函数，直接将函数的代码嵌入到程序当中，但是这种方法也有缺陷，一是相同的代码重复编写，二是程序可读性往往没有使用函数的好。

(3) 为了提高效率和可读性的矛盾，C++提供了另一种方法，既定义**内联函数**，方法就是在定义函数时用修饰词 **inline**。内联函数编译的时候也是由编译器进行展开的，这意味着我们在调用内联函数的时候，不涉及函数的开销（如 1），因为这些代码编译的时候，被展开了，也就是说这些替换功能由编译器展开，而不是由我们程序员手工的将这些代码嵌入到程序当中，由编译器完成的。（重要知识点）

总结：我们什么时候用内联函数呢？**代码体短小精悍，并且使用频率比较高。**

内联函数的定义和声明是很简单的：只需要在函数的前面加关键字 **inline**

```
inline int max(int a,int b)
{
    return a>b?a:b;
}
#define MAX(a,b) (a)>(b)?(a):(b)
```

在 C 语言中我们要实现这个目的，可以使用**宏定义**来实现。这两者都能达到同样的目的，提高效率。因为他们本质上都是进行**展开**的。并没有涉及函数调用。那么两者有什么区别呢？实际上还是有一定的区别的。

[2]内联函数与带参数宏的区别

(1) 内联函数的调用要求实参和形参的类型要一致（也就是说他会进行**类型的检查**），另外内联函数会先对实参表达式进行求值，然后传递给形参，而宏调用时，只可用实参简单的代替形参。

(2) 内联函数是在**编译**的时候，在调用的地方将代码展开，而宏则是在**预处理**的时候进行替换。

(3) C++中建议使用 **inline** 代替带参数的宏。

这里我们总结一下：

1. 宏有两个功能（常量和带参数的宏（类似于函数调用））

C++推荐常量用 **const** 或 **enum**（**枚举**）替换，带参数的宏用 **inline** 内联函数替换。

也就是 C++**高级编程**建议用 **const/enum/inline** 替换宏。

对于低层次的编程，比如说框架程序的设计，宏还是很灵活的。关于宏我们后面会讲解一些宏的作用。一些比较灵活的用法。

[3]新的类型转换运算符

- (1) `const_cast<T>(expr)`
- (2) `static_cast<T>(expr)`
- (3) `reinterpret_cast<T>(expr)`
- (4) `dynamic_cast<T>(expr)`

C 语言的转换（旧式转换）：

- (1) `(T) expr`
- (2) `T(expr)`

C++ 转换（新式转换）：

- (1) `const_cast<T>(expr)`
- (2) `static_cast<T>(expr)`
- (3) `reinterpret_cast<T>(expr)`
- (4) `dynamic_cast<T>(expr)`

`dynamic_cast<T>(expr)` 执行“安全向下”转型操作，也就是说支持运行时识别指针或所指向的对象，这是一个唯一无法用旧式语法来进行转型的操作。这一般是在派生类和基类之间进行转型操作的。这个运算符在我们后面讲到类的继承的时候再说。

我们今天先介绍前三种：

- (1) `const_cast<T>(expr)`

➤ 用于移除对象的常量性（cast away the constness）

➤ `const_cast` 一般用来移除指针或引用。这句话该怎么理解呢？我们举一个例子：

The screenshot shows a C++ IDE with a code editor and an output window. The code in the editor is as follows:

```
#include <iostream>
using namespace std;
int main(void)
{
    const int val=100;           //val是一个常量
    int n=const_cast<int>(val);  //const_cast的第一个功能是移除常量性
                                //也就是将常量转变为变量，这明显是矛盾的
                                //除非我们定义一个新的变量n，来接受它
                                //这个时候n和val本身没有任何关系，也就是说val还是不可改变
                                //所以说移除变量的常量性没有任何的意义，既然没有任何意义
                                //所以编译就会出错：无法将“const int”转换为“int”

    return 0;
}
```

The output window shows the following error message:

```
1> 1.cpp
1>d:\cppnote\05\05\1.cpp (6): error C2440: “const_cast”: 无法从“const int”转换为“int”
1>      转换是有效的标准转换，可以隐式执行或通过使用 static_cast、C 样式转换或函数样式转换执行
1>
1>生成失败。
1>
1>已用时间 00:00:02.11
===== 生成: 成功 0 个, 失败 1 个, 最新 0 个, 跳过 0 个 =====
```

```
#include <iostream>
using namespace std;
int main(void)
{
    const int val=100;           //val是一个常量
    //int n =const_cast<int>(val); //const_cast的第一个功能是移除常量性
    //也就是将常量转变为变量,这明显是矛盾的
    //除非我们定义一个新的变量n,来接受它
    //这个时候n和val本身没有任何关系,也就是说val还是不可改变
    //所以说移除变量的常量性没有任何的意义,既然没有任何意义
    //所以编译就会出错:无法将“const int”转换为“int”

    int n=val; //但是这是可以的,这表示将val里面的值赋值给n变量,不涉及任何的类型的转换
    return 0;
}
```

输出

显示输出来源(S): 生成

```
1>----- 已启动生成: 项目: 05, 配置: Debug Win32 -----
1>生成启动时间为 9/25/2016 2:48:34 PM。
1>InitializeBuildStatus:
1> 正在对“Debug\05.unsuccessfulbuild”执行 Touch 任务。
1>ClCompile:
1> 1.cpp
1>LinkEmbedManifest:
1> 05.vcxproj -> D:\CppNote\05\Debug\05.exe
1>FinalizeBuildStatus:
1> 正在删除文件“Debug\05.unsuccessfulbuild”。
1> 正在对“Debug\05.lastbuildstate”执行 Touch 任务。
1>
1>生成成功。
1>
1>已用时间 00:00:04.78
===== 生成: 成功 1 个, 失败 0 个, 最新 0 个, 跳过 0 个 =====
```

也就说 const_cast 一般不用于对象，一般用于指针或者引用。我们可以下用指针的例子来进行转型。

```
//int n =const_cast<int>(val); //const_cast的第一个功能是移除常量性
//也就是将常量转变为变量,这明显是矛盾的
//除非我们定义一个新的变量n,来接受它
//这个时候n和val本身没有任何关系,也就是说val还是不可改变
//所以说移除变量的常量性没有任何的意义,既然没有任何意义
//所以编译就会出错:无法将“const int”转换为“int”

int n=val; //但是这是可以的,这表示将val里面的值赋值给n变量,不涉及任何的类型的转换
//*****
int *p=&val; //我们应知道&val它的类型应该是const int *
return 0;
}
```

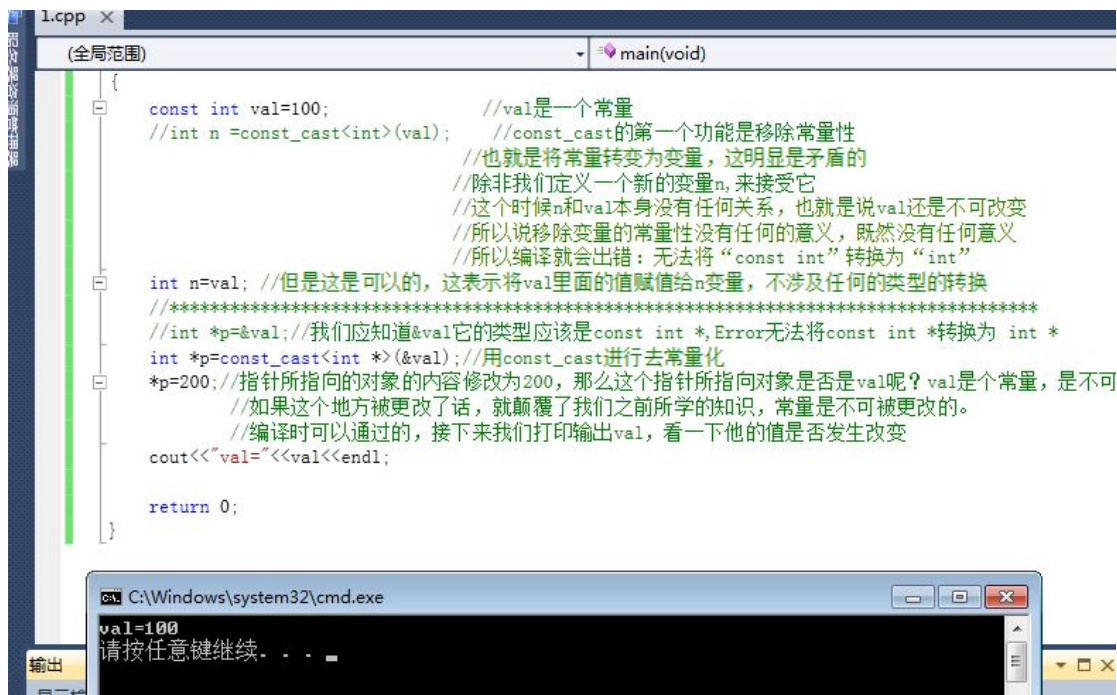
输出

显示输出来源(S): 生成

```
1>----- 已启动生成: 项目: 05, 配置: Debug Win32 -----
1>生成启动时间为 9/25/2016 2:53:09 PM。
1>InitializeBuildStatus:
1> 正在创建“Debug\05.unsuccessfulbuild”，因为已指定“AlwaysCreate”。
1>ClCompile:
1> 1.cpp
1>d:\cppnote\05\05\1.cpp (14): error C2440: “初始化”: 无法从“const int *”转换为“int *”
1>    转换丢失限定符
1>
1>生成失败。
1>
1>已用时间 00:00:00.74
===== 生成: 成功 0 个, 失败 1 个, 最新 0 个, 跳过 0 个 =====
```

Error:无法将 const int *转换为 int *

这个时候我们就需要去掉这个常量性呢？我们就可以用 const_cast 来进行转型。



```
1.cpp x
(全局范围) main(void)

{
    const int val=100;           //val是一个常量
    //int n =const_cast<int>(val); //const_cast的第一个功能是移除常量性
    //也就是将常量转变为变量，这明显是矛盾的
    //除非我们定义一个新的变量n，来接受它
    //这个时候n和val本身没有任何关系，也就是说val还是不可改变
    //所以说移除变量的常量性没有任何的意义，既然没有任何意义
    //所以编译就会出错：无法将“const int”转换为“int”

    int n=val; //但是这是可以的，这表示将val里面的值赋值给n变量，不涉及任何的类型的转换
    //*****
    //int *p=&val; //我们应知道&val它的类型应该是const int *,Error无法将const int *转换为 int *
    int *p=const_cast<int *>(&val); //用const_cast进行去常量化
    *p=200; //指针所指向的对象的内容修改为200，那么这个指针所指向对象是否是val呢？val是个常量，是不可
    //如果这个地方被更改了话，就颠覆了我们之前所学的知识，常量是不可被更改的。
    //编译时可以通过的，接下来我们打印输出val，看一下他的值是否发生改变

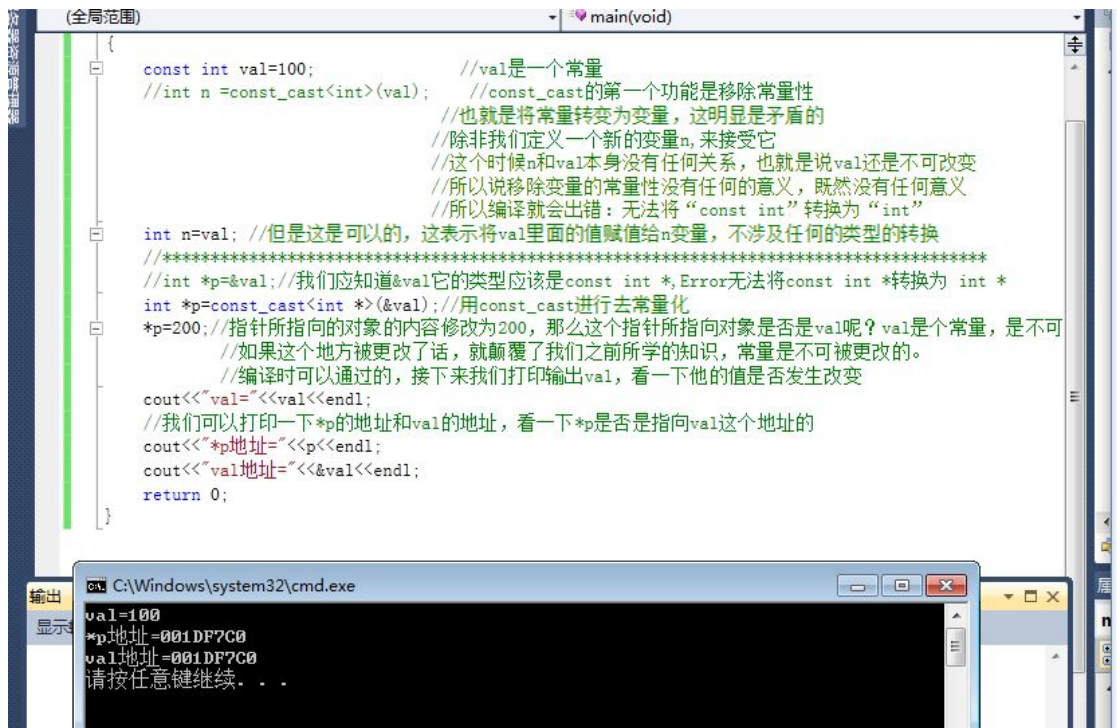
    cout<<"val="<<val<<endl;

    return 0;
}

C:\Windows\system32\cmd.exe
val=100
请按任意键继续. . .
```

可以看到 val 的值没有发生改变，这就是我们要说的第三点。

- 使用 const_cast 去除 const 限定的目的不是为了修改其内容。也就是说上面的例子中，*p 这个指针并不是指向 val 这个变量的。我们可以打印一下*p 和 val 的地址，看一下是否一样。



```
(全局范围) main(void)

{
    const int val=100;           //val是一个常量
    //int n =const_cast<int>(val); //const_cast的第一个功能是移除常量性
    //也就是将常量转变为变量，这明显是矛盾的
    //除非我们定义一个新的变量n，来接受它
    //这个时候n和val本身没有任何关系，也就是说val还是不可改变
    //所以说移除变量的常量性没有任何的意义，既然没有任何意义
    //所以编译就会出错：无法将“const int”转换为“int”

    int n=val; //但是这是可以的，这表示将val里面的值赋值给n变量，不涉及任何的类型的转换
    //*****
    //int *p=&val; //我们应知道&val它的类型应该是const int *,Error无法将const int *转换为 int *
    int *p=const_cast<int *>(&val); //用const_cast进行去常量化
    *p=200; //指针所指向的对象的内容修改为200，那么这个指针所指向对象是否是val呢？val是个常量，是不可
    //如果这个地方被更改了话，就颠覆了我们之前所学的知识，常量是不可被更改的。
    //编译时可以通过的，接下来我们打印输出val，看一下他的值是否发生改变

    cout<<"val="<<val<<endl;
    //我们可以打印一下*p的地址和val的地址，看一下*p是否是指向val这个地址的
    cout<<"*p地址="<<p<<endl;
    cout<<"val地址="<<&val<<endl;

    return 0;
}

C:\Windows\system32\cmd.exe
val=100
*p地址=001DF7C0
val地址=001DF7C0
请按任意键继续. . .
```

可以看一下他确实是指向了这个常量（注意实验的结果证明老师讲的前面的：***p 并不是指向 val 是不正确的**），但是他不可更改里面的内容。这里*p=200; 应该是赋值给的临时空间，我们该这么理解。我们前面举了一个指针的例子，下面我们在举一个引用的例子吧：


```
*p=200; //指针所指向的对象的内容修改为200, 那么这个指针所指向对象是否是val呢? val是一个
//如果这个地方被更改了话, 就颠覆了我们之前所学的知识, 常量是不可被更改的。
//编译时可以通过的, 接下来我们打印输出val, 看一下他的值是否发生改变

cout<<"val="<<val<<endl;
//我们可以打印一下*p的地址和val的地址, 看一下*p是否是指向val这个地址的
cout<<"*p地址="<<p<<endl;
cout<<"val地址="<<val<<endl;
//*****
//const_cast去除引用的例子以下是引用的例子
const int val2=200;
//int& refval2=val2; //我们知道非const的引用不能指向const引用
//所以我们需要去掉这个常量性
int& refval2=const_cast<int&>(val2); //用const_cast更改常量性
refval2=300; //一样的我们对这个引用重新赋值
cout<<"val2="<<val2<<endl;

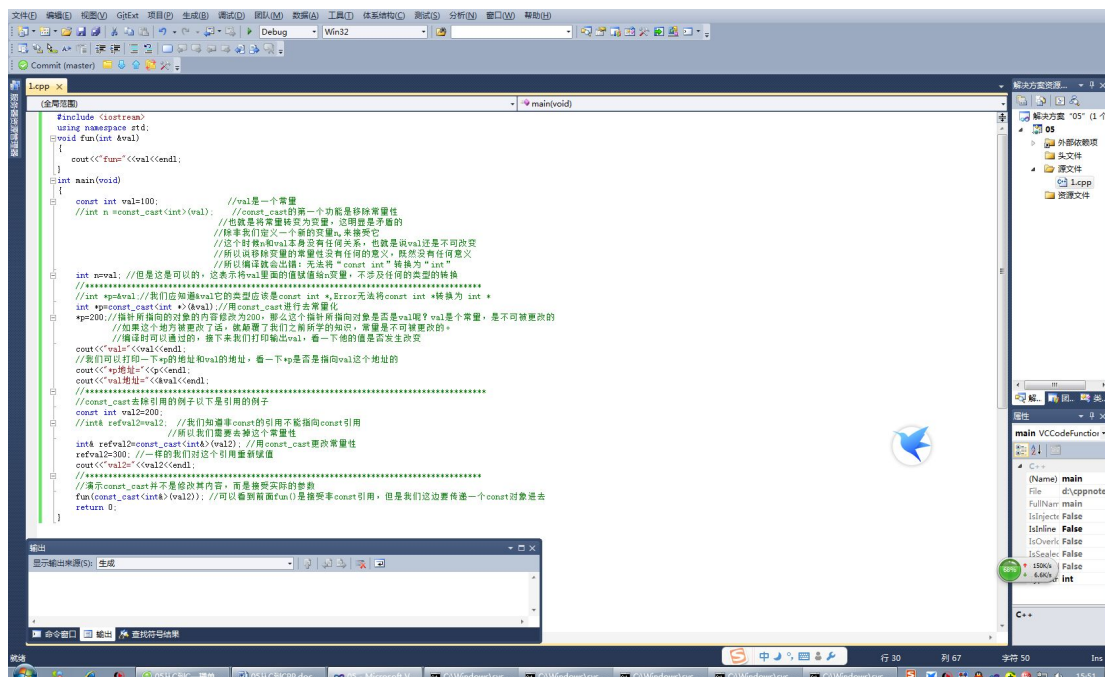
return 0;
}
```

Output:

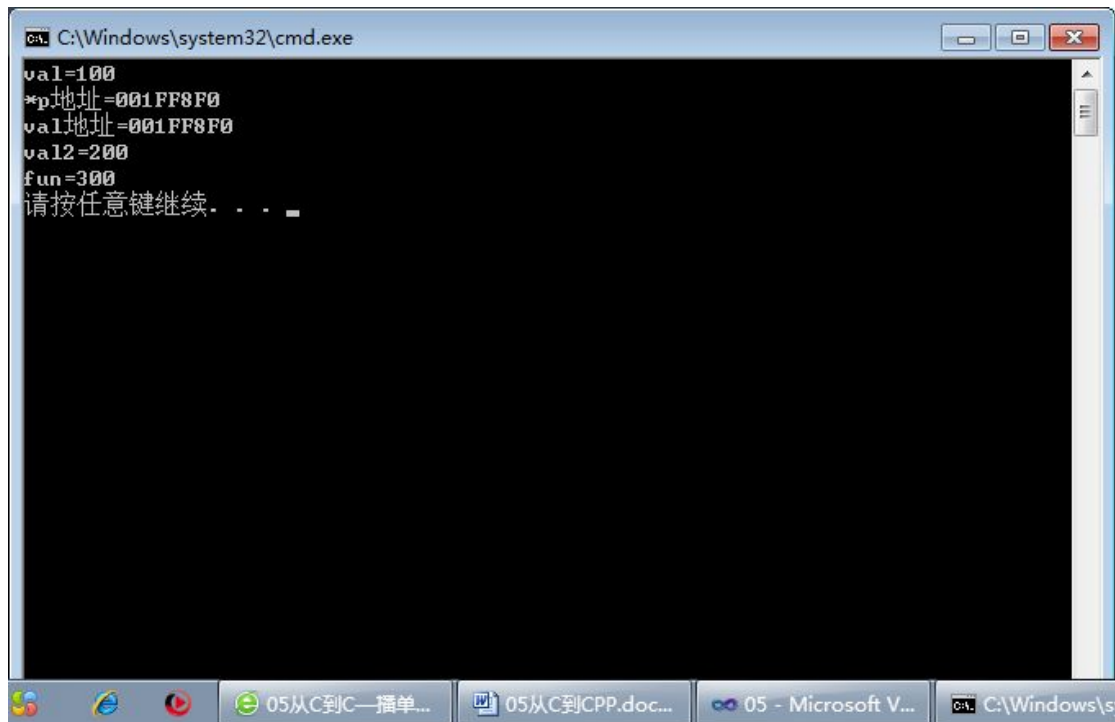
```
val=100
*p地址=0021FC7C
val地址=0021FC7C
val2=200
请按任意键继续. . .
```

可以看到对引用进行更改之后, 并没有更改引用所指向的常量的值。也就是说这个时候也是有一个临时对象的。

➤ 也就是说使用 `const_cast` 去掉 `const` 限定的目的不是为了修改它的内容, 而是为了函数能够接受这个实际的参数。我们举一个例子:



得到



```
C:\Windows\system32\cmd.exe
val=100
*p地址=001FF8F0
val地址=001FF8F0
val2=200
fun=300
请按任意键继续. . .
```

这里输出是 300，不是 200，说明 `const_cast<int&>val2` 已经是一个是去常量性之后的变量，作为一个整体（整体作为去常量性后对应的变量，就好理解了），然后引用 `refval` 指向这个变量，但是 `val2` 的值始终是不变的，这里老师讲的感觉有点出乎他的预料。

总结：

- (1) `const_cast` 用来移除常量性
- (2) `const_cast` 一般用于指针或引用，我们对于一个常量移除本身没有任何意义。没有任何的相关性。
- (3) `const_cast` 去除常量性不是为了修改它所指向的内容，仅仅是为了去除 `const` 限定，为了函数能够接受这个实际的参数。

接下来我们看一下第二种转型的用法：static_cast

- (1) 编译器隐式执行的任何类型的转换都可以由 `static_cast` 来完成。讲到这边，我们要说一下什么是编译器的隐式转换，编译器的转换从大方面就分为两种：**隐式转换**和**显式转换**。我们今天所说的类型转换运算符都属于显式转换的范畴，那么隐式转换是由编译器自动完成的，一般来说，它都是安全的，比如说将精度比较低的转换成精度比较高的，`int a; short b; a=b;` 这样的转换精度是不会丢失的。这种转换一般都称作是隐式转换。当然隐式转换是我们没有必要显式来完成的。
- (2) 当一个较大的算术类型赋值给较小的类型时，可以用 `static_cast` 进行强制转换，或者说是显式转换，（显式转换=强制转换）。我们举一个例子：

```
(全局范围)
#include <iostream>
using namespace std;
//static_cast将精度较高的类型转换为精度较低的类型
int main(void)
{
    int n=static_cast<int>(3.14);
    cout<<"n="<<n<<endl;
    return 0;
}
```

```
CA. C:\Windows\system32\cmd.exe
n=3
请按任意键继续. . .
```

(3) 可以将无类型指针 void *转换为有类型指针，例如：

```
#include <iostream>
using namespace std;
//static_cast将精度较高的类型转换为精度较低的类型
int main(void)
{
    int n=static_cast<int>(3.14);
    cout<<"n="<<n<<endl;
    //*****
    //以下是将无类型指针void*转化为有类型指针
    void *p=&n;
    int *p2=static_cast<int*>(p);

    return 0;
}
```

(4) 可以将基类型指针转成派生类指针。后面在讲解

(5) 无法将 const 转换为 nonconst，这个只能有 const_cast 才能办到。其他三种无法办到。

➤ reinterpret_cast

它通常为操作数的位模式提供较低层的重新解释，也就是说将数据以二进制的形式进行重新的解释。

```
int i;
char *p="This is a example"
i=reinterpret_cast<int>(p); //将这字符串指针转换成 32 位的整数（对 32 位系统而言），
//指针也是 32 位的，转换之后的值，i 与 p 完全相同，也就是说
//按二进制的值来转换值是完全相同的，整数的值等于这个地址
//的值

//第二种情况
int *ip; //定义一个整形指针，但是没有指定一个变量
char *pc=reinterpret_cast<char*>(ip);
//这里对整形指针重新解释，转换成字符指针，这时候 pc 和 ip 一样，它都是指向整形的数据对象，而不是字符串对象，比如说求长度的时候，strlen 可能会出现运行时错误，因为它实际所指向的对象并不是字符串，
```

尽量避免强制类型转换

➤ 尽量避免强制类型转换，也就是显式转换

➤ 如果无法避免的话，尽量使用新式转换（强制类型转换）

一共有四种，前三种可以由旧式转换体态，只有最后一种 **dynamic_cast<T>(expr)** 不能由旧式转换代替。对于去常量化，只能用 **const_cast<T>(expr)**，对于 **reinterpret_cast<T>(p)** 要慎重，要明白所指向的是什么数据类型。