C风格的强制类型转换(Type Cast)很简单，不管什么类型的转换统统是：TYPE b = (TYPE)a。

C++风格的类型转换提供了4种类型转换操作符来应对不同场合的应用。

**一. const\_cast**

常量指针被转化成非常量指针，并且仍然指向原来的对象。  
常量引用被转换成非常量引用，并且仍然指向原来的对象。

**去掉类型的 const 或 volatile 属性**。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7429096)

1. struct SA {
2. **int** i;
3. };
4. const SA ra;
5. //ra.i = 10; //直接修改const类型，编译错误
6. const SA\* pa = new SA;
7. SA &rb = const\_cast <SA&>(ra); //常引用->非常引用
8. rb.i = 10;
9. SA \*pb = const\_case <SA\*>(pa); //常指针->非常指针
10. pb->i = 10;

struct SA {

int i;

};

const SA ra;

//ra.i = 10; //直接修改const类型，编译错误

const SA\* pa = new SA;

SA &rb = const\_cast <SA&>(ra); //常引用->非常引用

rb.i = 10;

SA \*pb = const\_case <SA\*>(pa); //常指针->非常指针

pb->i = 10;

**二. static\_cast**

类似于C风格的强制转换。无条件转换，静态类型转换。用于：

1. 基类和子类之间转换：其中子类指针转换成父类指针是安全的；但父类指针转换成子类指针是不安全的。(基类和子类之间的动态类型转换建议用dynamic\_cast)  
2. 基本数据类型转换。enum, struct, int, char, float等。static\_cast 不能进行无关类型（如非基类和子类）指针之间的转换。  
3. 把空指针转换成目标类型的空指针。  
4. 把任何类型的表达式转换成 void类型。  
5. static\_cast 不能去掉类型的 onst、volitale属性(用const\_cast)。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7429096)

1. **int** n = 6;
2. **double** d = static\_cast<**double**>(n); // 基本类型转换
3. **int** \*pn = &n;
4. **double** \*d = static\_cast<**double** \*>(&n) //无关类型指针转换，编译错误
5. void \*p = static\_cast<void \*>(pn); //任意类型转换成void类型

int n = 6;

double d = static\_cast<double>(n); // 基本类型转换

int \*pn = &n;

double \*d = static\_cast<double \*>(&n) //无关类型指针转换，编译错误

void \*p = static\_cast<void \*>(pn); //任意类型转换成void类型

**三. dynamic\_cast**

动态类型转换，主要用于基类和子类之间的转换，运行时类型安全检查(转换失败返回NULL)：

1. 安全的基类和子类之间转换。  
2. 必须要有虚函数。  
3. 相同基类不同子类之间的交叉转换。但结果是NULL。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7429096)

1. class A{
2. public:
3. **int** m\_iNum;
4. virtual void foo(){}; //基类必须有虚函数。保持多态特性才能使用dynamic\_cast
5. };
6. class B: public A{
7. public:
8. **char** \*m\_szName[100];
9. void bar(){};
10. };
11. class C: public A{
12. };
13. //pb 是指向子类的指针
14. A\* pb = new B();
15. B\* pd1 = static\_cast <B \*>(pb); //子类->子类，静态类型转换，正确但不推荐
16. B\* pd2 = dynamic\_cast<B \*>(pb); //子类->子类，动态类型转换，正确
17. //pb 是指向父类的指针
18. A\* pb2 = new A();
19. B\* pd21 = static\_cast <B \*>(pb2); //父类->子类，静态类型转换，危险！pd21 可以访问子类m\_szName成员，这将越界。
20. B\* pd22 = dynamic\_cast<B\*>(pb2); //父类->子类，动态类型转换，安全的！pd22 将是一个NULL 指针。
21. //相同基类不同子类的转换
22. B\* pb = new B;
23. pb->m\_iNum = 100;
24. D\* pd1 = static\_cast <D \*>(pb); //编译出错
25. D\* pd2 = dynamic\_case<D \*>(pb); //pd2 为 null

class A{

public:

int m\_iNum;

virtual void foo(){}; //基类必须有虚函数。保持多态特性才能使用dynamic\_cast

};

class B: public A{

public:

char \*m\_szName[100];

void bar(){};

};

class C: public A{

};

//pb 是指向子类的指针

A\* pb = new B();

B\* pd1 = static\_cast <B \*>(pb); //子类->子类，静态类型转换，正确但不推荐

B\* pd2 = dynamic\_cast<B \*>(pb); //子类->子类，动态类型转换，正确

//pb 是指向父类的指针

A\* pb2 = new A();

B\* pd21 = static\_cast <B \*>(pb2); //父类->子类，静态类型转换，危险！pd21 可以访问子类m\_szName成员，这将越界。

B\* pd22 = dynamic\_cast<B\*>(pb2); //父类->子类，动态类型转换，安全的！pd22 将是一个NULL 指针。

//相同基类不同子类的转换

B\* pb = new B;

pb->m\_iNum = 100;

D\* pd1 = static\_cast <D \*>(pb); //编译出错

D\* pd2 = dynamic\_case<D \*>(pb); //pd2 为 null

**四. reinterpret\_cast**

仅仅重新解释类型，但没有进行二进制的转换：

1. 转换的类型必须是一个指针、引用、算术类型、函数指针或者成员指针。  
2. 在比特位级别上进行转换。它可以把一个指针转换成一个整数，也可以把一个整数转换成一个指针（先把一个指针转换成一个整数，在把该整数转换成原类型的指针，还可以得到原先的指针值）。但不能将非32bit的实例转成指针。  
3. 最普通的用途就是在函数指针类型之间进行转换。  
4. 很难保证移植性。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7429096)

1. **int** doSomething(){return 0;};
2. typedef void(\*FuncPtr)(); //FuncPtr is 一个指向函数的指针，该函数没有参数，返回值类型为 void
3. FuncPtr funcPtrArray[10]; //10个FuncPtrs指针的数组 让我们假设你希望（因为某些莫名其妙的原因）把一个指向下面函数的指针存入funcPtrArray数组：
4. funcPtrArray[0] = &doSomething;// 编译错误！类型不匹配，reinterpret\_cast可以让编译器以你的方法去看待它们：funcPtrArray
5. funcPtrArray[0] = reinterpret\_cast<FuncPtr>(&doSomething); //不同函数指针类型之间进行转换

int doSomething(){return 0;};

typedef void(\*FuncPtr)(); //FuncPtr is 一个指向函数的指针，该函数没有参数，返回值类型为 void

FuncPtr funcPtrArray[10]; //10个FuncPtrs指针的数组 让我们假设你希望（因为某些莫名其妙的原因）把一个指向下面函数的指针存入funcPtrArray数组：

funcPtrArray[0] = &doSomething;// 编译错误！类型不匹配，reinterpret\_cast可以让编译器以你的方法去看待它们：funcPtrArray

funcPtrArray[0] = reinterpret\_cast<FuncPtr>(&doSomething); //不同函数指针类型之间进行转换

**五. 总结**

去const属性用const\_cast。

基本类型转换用static\_cast。  
多态类之间的类型转换用daynamic\_cast。  
不同类型的指针类型转换用reinterpret\_cast。