1. 类型转换

类型转换(cast)是将一种数据类型转换成另一种数据类型。例如，如果将一个整型值赋给一个浮点类型的变量，编译器会暗地里将其转换成浮点类型。

转换会带来问题，比如在转换指针时，我们很可能将其转换成一个比它更大的类型，但这可能会破坏其他的数据。

1. 静态转换（static\_cast）
2. 用于基本数据类型之间的转换，如把int转换成char，把char转换成int。这种转换的安全性也要开发人员来保证。

如：

Int a;

Double b = static\_cast<double>(a);

1. 用于[类层次结构](http://baike.baidu.com/view/2405425.htm" \t "http://baike.baidu.com/_blank)中基类和子类之间指针或引用的转换。没有继承关系的类是无法转换的。

上行转换（将子类的指针或引用转换为基类表示）是安全的，具体原因多态中已经讲过。

下行转换（把基类指针或引用转换成子类表示）是不安全的。

如：

Class Animal() {};

Class Dog : public Animal() {};

Animal \* animal1 = NULL;

Dog \* dog1 = NULL;

Animal \* animal2 = static\_cast<Animal \*>(dog1);

三. 动态转换（dynamic\_cast）

基础类型是不能转换的，如double转int都是不能转换的。

因为动态转换十分严格，如果有失去精度或者不安全的转换都是不允许的。

因此，在基类和子类的转换中，上行转换是安全的，可以用动态转换；而下行转换是不安全的，因此不能用动态转换。

但是，通过多态可以让下行转换通过动态转换实现。因为使用多态，父类指针也是用子类的内存大小进行构造的，不论基类，子类怎么转换，指针都不会越界。如下所示：

Class Base

{  
virtual void func() {}; //虚函数

}

Class Child

{

Virtual void func() {};

}

Base \* base = new Child;

Child \* child = dynamic\_cast<Child\*>(base);

四. 常量转换（const\_cast）

该运算符用来修改类型的const属性，为非常量添加const属性，为常量去掉const属性。

**注意:**不能直接对非指针和非引用的变量使用const\_cast操作符去直接移除它的const.

使用实例：

Const int \*p = NULL;

Int \*newp = const\_cast<int \*>(p);

Int \*p2 = NULL;

Const int \* newp2 = const\_cast<const int \*>(p2);

五．重新解释转换（reinterpret\_cast）

最不安全的一种转换机制，可以将一种数据类型转换成另外一种类型，非常危险，所以不推荐使用。如把int转换为int\*

Int a = 10;

Int \*p = reinterpret\_cast<int \*>(a);