# Cast Operator

在C中，cast operator操作的结果不是左值。

在C++中，cast operator的结果包括下述3种情况：

1、如果Type是一个左值，或者是一个函数类型右值，那么返回结果就是一个左值。

2、如果Type是一个对象的右值引用，返回结果是一个即将消失的右值

3、其他情况下，返回结果均是右值(或者是纯右值)

# reinterpret\_cast

reinterpret\_cast主要用于处理无关类型之间的转换，这个值会与原值有着完全相同的比特位：

IBM的C++指南中明确告诉我们reinterpret\_cast可以，或者说应该在什么地方用其来进行类型转换。

首先，reinterpret\_cast<Type>(expression)的返回结果只有以下三种类型：

1. 如果Type是一个左值，或者是对于函数的右值引用，那么其结果是一个左值
2. 如果Type是一个object的右值引用，那么结果是一个xvalue(消亡值)
3. 其他情况下，结果均为一个rvalue(或者prvalue，即纯右值)

reinterpret\_cast主要用于下述场景：

1. 将一个指针转换为足够容纳它大小的整型(至少要容纳指针的字节数)
2. 将一个整型或者枚举类型转为一个指针
3. 将一个函数指针转换为另一类的函数指针
4. 将一类对象的引用转换为另一类对象的引用
5. 将一个指向数据成员的指针转换为一个指向不同class的数据成员的指针

总结来说，reinterpret\_cast用于在任意指针(或者引用)类型之间转换；以及指针与足够大的整型间转换(不能损失比特位)；而从整型转换为指针时无视大小(毕竟指针和长整型的大小相同)。

实际上reinterpret\_cast并不仅仅局限于上述几种类型转换，只不过错误使用reinterpret\_cast很容易导致程序不安全，所以推荐转换范围限定在上述几种(IBM C++编程规范推荐)

# static\_cast

static\_cast是将给定表达式转换为特定类型的操作符。在编译期检查转换的安全性。

对空指针应用static\_cast运算符号可以将其转换为目标类型的空指针值。

static\_cast的返回结果只有以下三种：(和reinterpret\_cast一致)

1. 如果Type是左值，或者是一个函数的右值引用，那么返回结果是左值
2. 如果Type是对一个对象的右值引用，那么返回结果是xvalue(将消亡的)
3. 其他情况下，Type均是右值(或者纯右值)

static\_cast主要用于下列转换中:

1. 将一个A类的左值转换为B&&，且该结果是一个即将消亡的右值
2. 将一个A类的指针右值转换为B类的指针右值
3. 将A类型的左值或者xvalue转换为B类型的右值引用，结果是一个即将消亡的B类引用。
4. 如果A类的右值引用(xvalue)能够被绑定为B类的右值引用，就将一个A类的左值引用转换为B&&。
5. 将一个表达式e转换为类型T，如果类型T允许直接初始化。

为了支持上述前三转换，需要满足以下条件：

1. A是B的基类
2. 存在从A类指针到B类指针的标准转换
3. A不是B的虚基类 && A不是虚基类的基类

static\_cast 主要用于非多态类型间的转换，不提供运行期的检查来确保转换安全性。总结而言为：1、主要用于类层次结构中，基类和子类之间指针和引用的转换(上行转换即把子类转换为父类，是安全的；下行转换，把父类转换为子类，是不安全的！！！如果type是类型的引用，由于其在运行期才能得到具体信息，所以不会报错) 2、用于基本数据类型间的转换 3、将void指针类型转换为目标类型的指针

# dynamic\_cast

dynamic\_cast会在运行期对以下转换进行安全检测：

1. 将一个指向基类的指针转换为指向子类的指针
2. 将一个基类的左值引用转换为子类的左值引用
3. 将一个基类的xvalue引用转换为子类的rvalue引用

由于上述保证，程序可以安全地使用类层次结构。

dynamic\_cast的type-id必须是类指针、引用或者void\*;

dynamic\_cast遵循以下规则(表达式dynamic\_cast<T>(v))：

1. 如果T是一个指针类型，那么v必须是一个右值(纯右值)，此时返回值是T类型的右值
2. 如果T是一个类型的左值引用，那么v必须是一个左值，返回结果是T的左值引用
3. 如果T是一个右值引用，那么返回值是T的一个即将消亡的右值引用

如果T是指针，而本运算符失败，就返回T类型的空指针。如果T是引用，而本运算符失败，就抛出异常std::bad\_cast。

dynamic\_cast要求运行时类型信息(RTT)，必须在编译时通过编译器选项指定。

我们仅可以在多态类上使用dynamic\_cast运算符执行向下转换。本运算符使用多态类生成的运行时信息。

除了dynamic\_cast外，其余cast都不执行运行期安全检测。

dynamic\_cast和static\_cast二者都可以用于层次结构的类型转换，但是dynamic\_cast提供的是运行期的安全检查，而static\_cast提供的是编译期的安全检查，此时二者效果几乎是一样的。而在进行向下类型转换时，dynamic\_cast具有检查类型的功能(会报错bad\_alloc)，比static\_cast更安全。

# const\_cast

const\_cast主要用于移除一个对象的常量性或者易变性(其余转型操作都不能去除)。

如果T1类型的右值引用(包括prvalue)可以转换为T2类型后消去常量性，那么我们也可以进行下列转换：

1. 将T1类型的常量表达式转换为T2类型的即将消亡的右值引用
2. 将指向T1类型的数据成员的右值指针(包括prvalue)转换为T2类型的指针。

对于const数据我们更要这样保证：绝对不对const数据进行重新赋值。可是，如果我们不想修改const变量的值，那我们又为什么要去const呢？

原因是，我们可能调用了一个参数不是const的函数，而我们要传进去的实际参数确实const的，但是我们知道这个函数是不会对参数做修改的。于是我们就需要使用const\_cast去除const限定，以便函数能够接受这个实际参数。

# 总结

dynamic\_cast主要用于执行“安全的向下转型“。其Type-id必须时一个指针、引用或者void\*。

static\_cast在用于类的层次结构种进行类型转换时，几乎与dynamic\_cast一样，但是，不提供运行期的类型检查保证。其一般也广泛用于基本数据类型的强制隐式转换。

reinterpret\_cast主要用于处理无关类型间的转换，转换后的值会与原值有相同的比特位，其必须保证转换后的类型足以容纳原类型的所有比特位。

const\_cast主要用于去除类型的常量性或者易变性的场合。