C++强制类型转换：

<https://blog.csdn.net/u014624623/article/details/79837849>

1. static\_cast:用于静态类型转换（非多态类型）

static\_cast< new\_type >(expression)

1. 基本数据类型的转换，与C中的隐式类型类似，但不能用于不相关类型的转换，如整型转换为整型指针。
2. static\_cast不能转换掉const,volatile
3. 基类与派生类的转换

例：程序gaode\_test

class Base

{

public:

virtual void func()

{

*cout* << "Print Base" << *endl*;

}

};

class Derived : public Base

{

public:

void func()

{

*cout* << "Print Derived" << *endl*;

}

};

int i = 10;

double d1 = i; // 隐式类型转换b

double d2 = static\_cast<double>(i);

double d3 = 4.5;

double\* pd = &d3;

void\* d = static\_cast<void\*>(pd); // 指针转换为void指针

double\* pd1 = static\_cast<double\*>(d); // void指针转换为其它指针

*cout* << \*pd1 << *endl*;

int a = 10;

const int b = static\_cast<const int>(a);

const int c = 5;

// int d = static\_cast<int>(c); // error，static\_cast不能转换掉const

// int\* p = static\_cast<int\*>(a); // error，类型转换无效，不能从int转换为int\*

//void p; // error，不允许不完整类型

// void\* p = static\_cast<int\*>(a); // error，类型转换无效

void\* p = static\_cast<int\*>(&a);

Base bb;

Derived dd;

Base& bb1 = static\_cast<Base&>(dd); // 上行，安全

Base\* pb = static\_cast<Base\*>(&dd); // 上行，安全

pb->func(); // Print Derived

Derived& dd1 = static\_cast<Derived&>(bb); // 下行，不安全

Derived\* pdd = static\_cast<Derived\*>(&bb); // 下行，不安全

pdd->func(); // 在VS上为Print Base，但其它编译器不一定

1. dynamic\_cast:动态类型转换，只能用于完整类类型或void指针和引用的转换，并且相互转换的类类型必须构成多态。

在下行转换时，dynamic\_cast相比static\_cast要安全，转换的结果会有所不同

例：程序gaode\_test1

int a = 9;

// double b = dynamic\_cast<double>(a); // error，dynamic\_cast只能用于转换指针和引用

// double\* pd = dynamic\_cast<double\*>(&a); // error，dynamic\_cast只能用于完整类类型或void的指针和引用

Base b;

Derived d;

Base& bb = dynamic\_cast<Base&>(d); // 上行，安全与static\_cast相同

Base\* pb = dynamic\_cast<Base\*>(&d); // 上行，与static\_cast相同

pb->func(); // Print Derived

Derived\* pd = dynamic\_cast<Derived\*>(&b); // 下行，pd为nullptr

Derived& d1 = dynamic\_cast<Derived&>(b); // 下行，会抛出异常

从上面的程序可以看出，在基类与派生类的上行转换中，派生类的对象被强制转换为基类的指针或引用，但调用的虚函数仍然是派生类的，这是由于虚函数指针的缘故，派生类对象地址中存储的是派生类虚函数的函数指针。

Qt信号与槽

模板实现单例模式

内联函数与宏定义的区别