默认构造函数：

例：程序C++11\_test5

class Test {

public:

// 使用默认构造函数

Test() = default;

};

委托构造函数：使用它所属类的其他构造函数执行它自己的初始化过程。

委托构造函数的执行顺序是：先执行被委托的构造函数，再执行委托构造函数本身。

例：程序C++11\_test5

class Test1

{

public:

Test1(*string* no, int count) : book\_no\_(no), count\_(count)

{

*cout* << "constructor Test1(string no, int count)" << *endl*;

}

// 委托构造函数

// 初始化列表只能是被委托的构造函数

// 不能有其他的初始化操作

Test1() : Test1("", 0)

{

*cout* << "default constructor" << *endl*;

}

Test1(*string* no) : Test1()

{

*cout* << "constructor Test1(string no)" << *endl*;

*cout* << no << " is not used" << *endl*;

}

private:

*string* book\_no\_;

int count\_;

};

// 委托构造

Test1 t1;

Test1 t2("1234");

输出为：

constructor Test1(string no, int count)

default constructor

constructor Test1(string no, int count)

default constructor

constructor Test1(string no)

1234 is not used

constexpr构造函数：

类的构造函数不能是const（需要在构造函数初始化类成员变量；对象只有构造完成才具有所谓的const属性）。

但类的构造函数可以是constexpr函数。

例：程序C++11\_test5

class Test2

{

public:

// constexpr必须初始化所有的成员变量

// 初始化值是一条常量表达式或constexpr

// 构造函数（针对某些类类型的成员变量）

// 必须在初始化列表中初始化，之后还可以在

// 构造函数体内初始化

constexpr Test2(bool hw, bool io, bool other)

: hw\_(hw), io\_(io), other\_(other) , count\_(count)

{

count\_ = count;

}

private:

bool hw\_;

bool io\_;

bool other\_;

int count\_;

};

// constexpr

Test2 t3(false, false, false, 3);

Test2 t4;