句柄类产生的原因：类的多态性，触发动态绑定时，需要的是基类的指针或引用。通过句柄类，可以采用对象的方式来触发动态绑定。

例：程序class\_test30

// 句柄类

#include <iostream>

class Base // 基类

{

public:

virtual void func() const

{

std::cout << "Base::func()" << std::endl;

}

virtual void Print()

{

std::cout << "Base::Print()" << std::endl;

}

// clone()函数是虚函数，参考：C++学习-类的多态性.docx中虚函数的要求

// 其主要作用是在句柄类Handle拷贝未知对象时，能够正确的拷贝对象的副本

virtual Base\* clone() const

{

return new Base(\*this);

}

};

class Dervie : public Base // 派生类

{

public:

void func() const

{

std::cout << "Dervie::func()" << std::endl;

}

void Print()

{

std::cout << "Dervie::Print()" << std::endl;

}

Dervie\* clone() const

{

return new Dervie(\*this);

}

};

class Handle // 句柄类，类似一个智能指针

{

public:

Handle() : p(NULL), use(new std::size\_t(1)) // 引用计数初始化为

{

}

Handle(const Handle& rhs) : p(rhs.p), use(rhs.use)

{

++\*use; // 这里同样会改变rhs中use指针所指向的对象的值，rhs是一个引用

}

// 拷贝未知类型的构造函数，不是拷贝构造函数，而是带参数的构造函数

Handle(const Base& b) : p(b.clone()), use(new std::size\_t(1))

{

}

Handle& operator=(const Handle&);

const Base\* operator->() const // 重载箭头操作符，const版本

{

if (p)

{

return p;

}

else

{

throw std::logic\_error("unbound Handle");

}

}

Base\* operator->() // 非const版本

{

if (p)

{

return p;

}

else

{

throw std::logic\_error("unbound Handle");

}

}

const Base& operator\*() const // 重载解引用操作符

{

if (p)

{

return \*p;

}

else

{

throw std::logic\_error("unbound Handle");

}

}

Base& operator\*() // 非const版本的重载解引用操作符

{

if (p)

{

return \*p;

}

else

{

throw std::logic\_error("unbound Handle");

}

}

~Handle()

{

decr\_use();

}

private:

void decr\_use()

{

if (--\*use == 0)

{

delete p;

delete use;

p = NULL;

use = NULL;

}

}

private:

Base\* p;

std::size\_t\* use;

};

Handle& Handle::operator=(const Handle& rhs)

{

++\*rhs.use;

decr\_use();

p = rhs.p;

use = rhs.use;

return \*this;

}

void f(Handle handle)

{

handle->func();

handle->Print();

}

int main(int argc, char\* argv[])

{

Base b;

Dervie d;

Handle handle1(b);

// handle1->func()相当于执行

// (handle1.operator->())->func()

handle1->func(); // 执行基类Base的func()

handle1->Print(); // 执行基类Base的Print()

Handle handle2(d);

handle2->func(); // 执行派生类Derive的func()

handle2->Print(); // 执行派生类Derive的Print()

f(b); // 执行Base类的函数

f(d); // 执行Derive类的函数

return 0;

}

可以看出，通过句柄类Handle，能够以对象的方式来触发动态绑定。Handle类的对象，接收一个Base型对象时，执行基类的函数，接收一个Derive型对象时，执行派生类的函数