函数的覆盖和隐藏都发生在类的继承过程中。

无论是函数的覆盖还是隐藏，必须是在基类中是public或protected成员，否则根本构不成覆盖或隐藏。

函数的覆盖：在基类和派生类中定义的虚函数，并且能够触发动态绑定（参考C++学习14-类的多态性中虚函数的要求）。

函数的隐藏：在基类和派生类中定义的同名函数，不属于覆盖的都属于隐藏。

在基类和派生类中使用了同名函数和同名数据成员，无论覆盖还是隐藏，在派生类作用域中，基类的同名函数和同名数据成员会被屏蔽，如果希望使用基类中成员，需要使用作用域操作符”::”。

例：程序class\_test21

// 类继承中的函数覆盖和隐藏

#include <iostream>

class Base

{

public:

virtual void xfn(int i)

{

std::cout << "Base::xfn(int i)" << std::endl;

}

void yfn(float f)

{

std::cout << "Base::yfn(float f)" << std::endl;

}

void zfn()

{

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

}

void tfn(int i)

{

std::cout << "Base::tfn(int i)" << std::endl;

}

};

class Derive : public Base

{

public:

virtual void xfn(int i) // 虚函数，函数覆盖

{

std::cout << "Derive::xfn(int i)" << std::endl;

}

void yfn(int c) // 隐藏屏蔽基类Base的函数

{

std::cout << "Derive::yfn(int c)" << std::endl;

}

void zfn() // 隐藏屏蔽基类Base的函数

{

std::cout << "Derive::zfn()" << std::endl;

}

virtual void tfn() // 隐藏屏蔽基类的Base函数，虽然定义为虚函数，但形参不同，不触发动态绑定

{

std::cout << "Derive::tfn()" << std::endl;

}

};

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

{

Derive d;

Base\* b\_ptr = &d;

Derive\* d\_ptr = &d;

b\_ptr->xfn(5); // 调用Derive类的xfn()函数

d\_ptr->xfn(5); // 调用Derive类的xfn()函数

b\_ptr->yfn(3.5); // 调用Base类的yfn()函数

d\_ptr->yfn(4); // 调用Derive类的yfn()函数

b\_ptr->zfn(); // 调用Base类的zfn()函数

d\_ptr->zfn(); // 调用Derive类的zfn()函数

b\_ptr->tfn(5); // 调用Base类的tfn()函数

// d\_ptr->tfn(8); // error，派生类型的对象无法直接调用基类中的同名函数

d\_ptr->Base::tfn(8); // ok，可以通过作用域操作符调用基类中的同名函数

d\_ptr->tfn();

return 0;

}

类中函数的重载：只发生在一个类中。

类中函数重载与普遍函数重载相同，而且虚函数也可以被重载。

例：

// 类继承中的函数重载

#include <iostream>

class Base

{

public:

virtual void fcn()

{

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

}

};

class Derive1 : public Base

{

public:

void fcn(int i) // 隐藏屏蔽了基类的fcn()函数，不触发动态绑定

{

std::cout << "Derive1::fcn(int i)" << std::endl;

}

};

class Derive2 : public Derive1

{

public:

void fcn()

{

std::cout << "Derive2::fcn()" << std::endl;

}

void fcn(int i) // 虚函数也可以被重载

{

std::cout << "Derive2::fcn(int i)" << std::endl;

}

};

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

{

Base b;

Derive1 d1;

Derive2 d2;

Base\* b\_ptr = &b;

Base\* b1\_ptr = &d1;

Base\* b2\_ptr = &d2;

b\_ptr->fcn(); // Base::fcn()

// 尽管b1\_ptr和b2\_ptr指向的分别是Derive1对象和Derive2对象

// 但由于它们本身是Base\*类型的变量，所以不能调用Derive1和

// Derive2的自有成员函数

b1\_ptr->fcn(); // Base::fcn()，不触发动态绑定

b2\_ptr->fcn(); // Derive2::fcn()，触发动态绑定

return 0;

}

C++类继承中函数的调用遵循以下步骤：

1. 首先确定进行函数调用的对象、引用或指针的静态类型（即编译时的类型）。基类的对象、引用或指针无法调用派生类的自有函数，无论给其赋值对象是何种类型。
2. 在该类中查找函数，如果找不到，就在类的继承链往上找，直到找到。如果找不到，则调用是错误的。
3. 一旦查找到该函数，进行常规类型检查（形参类型是否匹配，返回值是否正确等）。
4. 假定函数调用合法，如果函数是虚函数且是通过基类引用或指针调用，则触发动态绑定，否则编译器直接调用该类的函数。