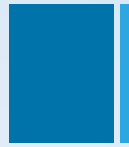




C++

第十五讲 虚函数与多态性（一）

C++备课组 丁盟



自我介绍

丁盟

QQ : 2622885094





上一讲教学目标

- 掌握C++中运算符重载的含义
- 掌握C++中常见的运算符的重载形式
- 了解C++中类类型转换的几种手段



本讲教学目标

- 掌握C++中虚函数的概念
- 理解重载、隐藏、覆盖

1

引述

2

多态的核心虚函数

3

多层次、多重继承中的动态多态性

4

隐藏、覆盖、重载

引述

❖ 多态的实现：函数绑定

函数绑定就是函数的入口地址同函数调用相联系的过程，绑定就是要计算被调用函数的入口地址，并将该地址存放到函数调用指令的地址码部分。

❖ C++通过函数的重载和运算符重载实现静态多态

编译时即可确定
调用哪个函数

引述

静态绑定

```
class OverLoad {
public:
    int foo();           // 1
    int foo() const;    // 2
    int foo(int aFir,
            int aSec); // 3
    int foo(const int *arg); // 4
    int foo(int *arg); // 5
    int foo(const int &aRef); // 6
    int foo(int &aRef); // 7
private:
    int m_iVal;
    const int m_kiVal;
};
```

```
int main(void) {
    OverLoad obj;
    const OverLoad kObj;
    obj.foo();
    kObj.foo();
    obj.foo(1,2);
    int iVal = 100;
    const int *kp = &iVal;
    int *p = &iVal;
    obj.foo(kp);
    obj.foo(p);
    const int &kRef = iVal;
    obj.foo(kRef);
    obj.foo(iVal);
    return 0; }
```

引述

- ❖ C++中通过虚函数实现动态多态

什么是动态多态？

如何定义虚函数实现动态多态？

- ❖ 回忆隐藏（基类和子类有同名成员时）

指向子类对象的基类指针或引用只能访问基类的成员

派生类默认访问派生类的同名成员

派生类要访问继承来的同名成员必须加"基类名::"

引述

```
class Base {
public:
    Base(double arg=0);
    double get() const;
private:
    double m_dVal;
};
class Derived:public Base {
public:
    Derived(double arg=1);
    double get() const;
    double getParent();
private:
    double m_dVal1;
};
```

```
Base::Base(double arg)
    :m_dval(arg) {
}
double Base::get() const {
    return m_dVal;
}
Derived::Derived(double arg) {
    m_dVal1 = arg;
}
double Derived::get() const {
    return m_dVal1;
}
double Derived::getParent() {
    return Base::get();
}
```

引述

```
int main(void) {  
    Base bObj;  
    Base *bp = &bObj;  
    cout << bp->get() << endl;  
  
    Derived dObj;  
    bp = &dObj;  
    cout << bp->get() << endl;  
    cout<<dObj.get()<<endl;  
    cout<<dObj.getParent()<<endl;  
    return 0;  
}
```

基类指针无论指向的是基类的对象还是派生类的对象，总是调用基类的同名成员函数

0
0
1
0

静态绑定

引述

❖ 动态多态的效果是：

基类指针（引用）指向基类对象时调用基类的同名函数；基类指针（引用）指向派生类对象时调用派生类的同名函数

引述

```
class Base {
public:
    Base(double arg=0);
    virtual double get() const;
private:
    double m_dVal;
};

class Derived:public Base {
public:
    Derived(double arg=1);
    double get() const;
private:
    double m_dVal1;
};
```

```
Base::Base(double arg)
    :m_dVal(arg) {
}

double Base::get() const {
    return m_dVal;
}

Derived::Derived(double arg) {
    m_dVal1 = arg;
}

double Derived::get() const {
    return m_dVal1;
}
```

引述

```
int main(void) {  
    Base bObj;  
    Base *bp = &bObj;  
    cout << bp->get() << endl;  
  
    Derived dObj;  
    bp = &dObj;  
    cout << bp->get() << endl;  
  
    return 0;  
}
```

基类指针指向基类对象
时调用基类的同名函数，
指向派生类对象时调用
派生类的同名函数

0
1

动态绑定

1

引述

2

多态的核心虚函数

3

多层次、多重继承中的动态多态性

4

隐藏、覆盖、重载

多态的核心虚函数

❖ 虚函数的声明方法：

virtual <返回类型> <成员函数名>(形式参数表);

❖ 注意： virtual声明时出现 实现时**不能再有**

```
class Base {  
public:  
    virtual double get() const;  
private:  
    double m_dVal;  
};  
virtual double Base::get() const {  
    return m_dVal;  
}
```

多态的核心虚函数

❖ virtual在基类中必须有，在派生类中可以省略

```
class Base {  
public:  
    Base(double arg=0);  
    virtual double get();  
private:  
    double m_dVal;  
};
```

```
class Derived:public Base {  
public:  
    Derived(double arg=1);  
    virtual double get();  
private:  
    double m_dVal1;  
};
```

为了增强可读性，不推荐在派生类中省略

多态的核心虚函数

- ❖ 虚函数在基类和其各层派生类中的原型要求**保持完全一致（返回值、函数名、参数表、const）**

```
class Base {  
public: virtual double get(int arg=3)const;  
private: double m_dVal;  
};
```

```
class Derived:public Base {  
public: virtual double get(int arg=5)const;  
private: double m_dVal1;  
};
```

多态的核心虚函数

❖ 有一项不一致就不能实现动态多态

```
class Base {  
public:    virtual int get() const;  
private: int m_iVal;  
};
```

```
class Derived:public Base {  
public:    virtual double get() const;  
private:  double m_dVal;  
};
```

定义为虚函数时，若仅返回值不同是不允许的，将产生编译错误！

多态的核心虚函数

- ❖ 派生类必须**公有继承基类**，这是赋值兼容的前提，派生类只有公有继承基类，才允许基类的指针指向派生类对象，基类的引用才是派生类对象的别名。

```
class Derived : protected Base
{
public:
    Derived(double arg=1);
    virtual double get() const;
private:
    double m_dVal1;
};
```

```
int main(void) {
    Base bObj;
    Base *bp = &bObj;
    cout << bp->get() << endl;
    Derived dObj;
    bp = &dObj; //Error
    cout << bp->get() << endl;
    return 0;
}
```

多态的核心虚函数

- ❖ 只有类的成员函数才能声明为虚函数。因为，虚函数仅适用于有继承关系的类，所以普通函数不能声明为虚函数。

```
virtual int foo()  
{  
    cout << "foo是全局函数!" << endl;  
}
```

会出现编译错误！

多态的核心虚函数

- ❖ **静态成员函数不能是虚函数**。因为，静态成员函数是属于类的而不受限于某个对象。

```
class Base
{
public:
    virtual static int get() const;
};
```

会出现编译错误！

多态的核心虚函数

- ❖ **内联函数不能是虚函数**，即使虚函数在类的内部定义，编译时仍将其看作是非内联的。

```
class Base
{
public:
    Base(double arg=0);
    virtual inline double get() const;
};
```

编译时视inline不存在！

多态的核心虚函数

- ❖ **构造函数不能为虚函数**。构造函数的功能是初始化对象，因此语法上限制构造函数不能为虚函数。

```
class Base
{
public:
    virtual Base(double arg=0);
    virtual double get() const;
};
```

编译错误！

多态的核心虚函数

- ❖ **析构函数常常设置为虚函数**。如果基类的析构函数是虚函数，那基类的各级派生类的析构函数均自动成为虚函数（无论名字是否相同）。
- ❖ 若基类指针指向派生类对象时，当删除该指针时，就会调用派生类的析构函数，而后派生类的析构函数又自动调用基类的析构函数，这样整个派生类的析构函数都被完全释放。

多态的核心虚函数

```
class Base {
public:
    virtual ~Base() {
        cout << "~Base() "
              << endl;
    }
};

class Derived:public Base {
public:
    Derived() {
        m_p = new char[1000];
    }
};
```

如果基类的析构函数
不是虚函数会怎样？

```
~Derived() {
    delete [] m_p;
    cout << "~Derived()"
          << endl;
}

private:
    char * m_p;
};

int main(void) {
    Base *bp = new Derived;
    delete bp;
    return 0;
}
```

~Base()
~Derived()

1

引述

2

多态的核心虚函数

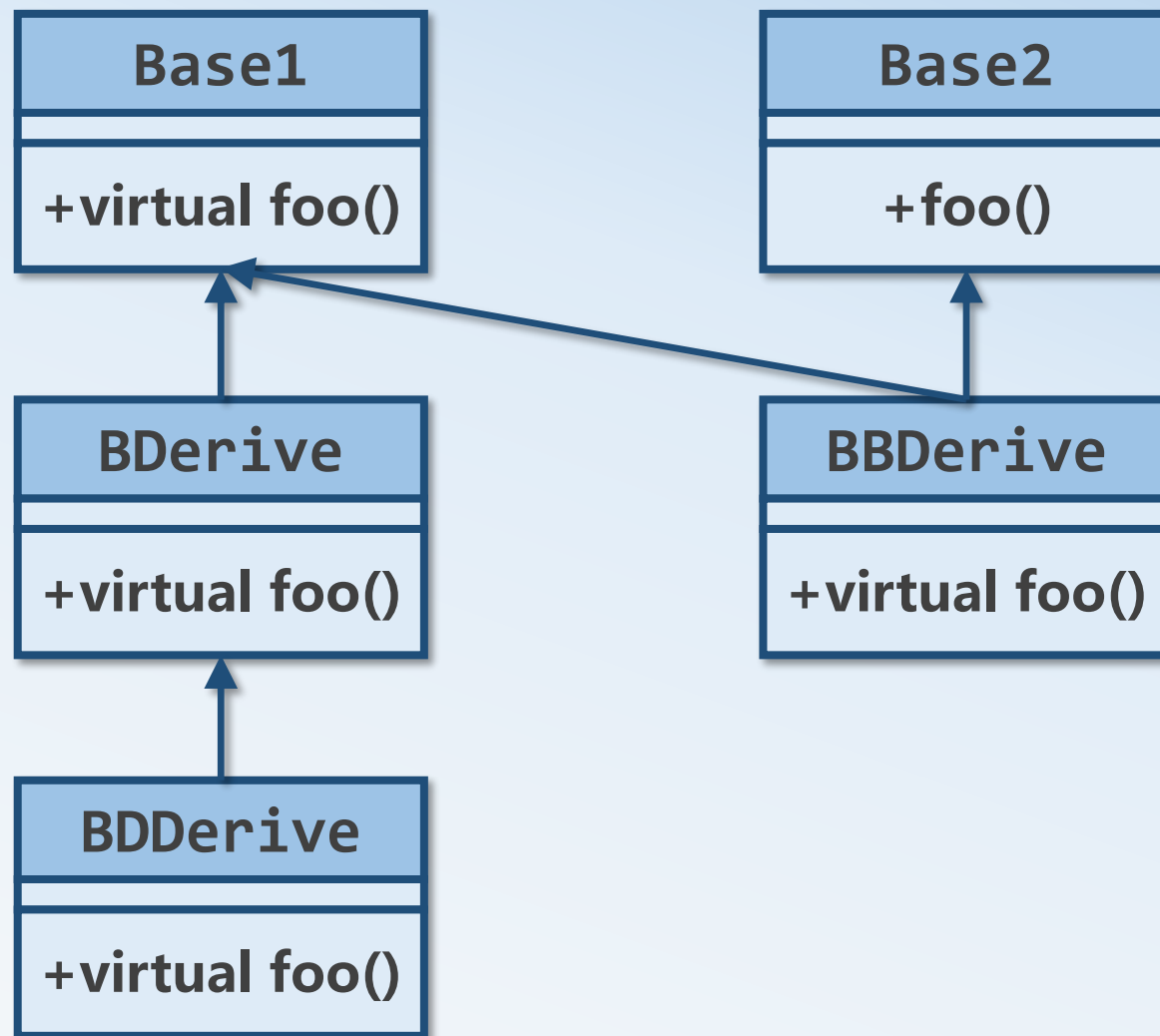
3

多层次、多重继承中的动态多态性

4

隐藏、覆盖、重载

多层次、多重继承中的动态多态性



多层次、多重继承中的动态多态性

```
Base1 b1, *p1;  
Base2 b2, *p2;  
BDerive bd;  
BDDerive bdd;  
BBDerive bbd;  
p1=&b1;    p1->foo(); // 调Base1基类函数  
p1=&bd;    p1->foo(); // 调BDerive类函数  
p1=&bbd;   p1->foo(); // 调派生类BBDerive函数有多态性  
p2=&b2;    p2->foo(); // 调Base2基类函数, 有无多态性  
p2=&bbd;   p2->foo(); // 调Base2基类函数有无多态性  
p1=&bdd;   p1->foo(); // 调派生类BDDerive函数有多态性
```

1

引述

2

多态的核心虚函数

3

多层次、多重继承中的动态多态性

4

隐藏、覆盖、重载

隐藏、覆盖、重载

❖ 函数的隐藏

隐藏规则：

1. 派生类的函数跟基类的函数同名，**其他**不完全相同，此时不论有没有virtual关键字，基类函数将被隐藏。（注意有virtual仅返回值类型不同的情况将产生编译错误）
2. 派生类的函数跟基类的函数同名，且其余参数完全一致但基类没有virtual关键字，此时基类函数也将被隐藏。

隐藏、覆盖、重载

```
class Base {  
public:  
    void g(float x) {  
        cout << "Base::g(float) "  
            << x << endl;  
    }  
    void h(float x) {  
        cout << "Base::h(float) "  
            << x << endl;  
    }  
};
```

```
class Derived : public Base {  
public:  
    void g(float x) {  
        cout << "Derived::g(int) "  
            << x << endl;  
    }  
    void h(float x) {  
        cout << "Derived::h(float) "  
            << x << endl;  
    }  
};
```

隐藏、覆盖、重载

```
int main(void) {  
    Derived d;  
    Base *pb = &d;  
    Derived *pd = &d;  
    pb->g(3.14f);           // Base::g(float)  
    pd->g(3.14f);           // Derived::g(int)  
    pd->Base::g(3.14f);     // Base::g(float)  
    pb->h(3.14f);           // Base::h(float)  
    pd->h(3.14f);           // Derived::h(float)  
    pd->Base::h(3.14f);     // Base::h(float)  
    return 0;  
}
```


隐藏、覆盖、重载

❖ 函数的覆盖

覆盖规则：

1. 指派生类与基类的成员函数之间。
2. 基类函数必须有virtual关键字。
3. 基类和派生类同名函数的原型完全相同（返回值、函数名、参数表、const）。

隐藏、覆盖、重载

❖ 覆盖是通过虚函数表实现的，覆盖也成为重写。

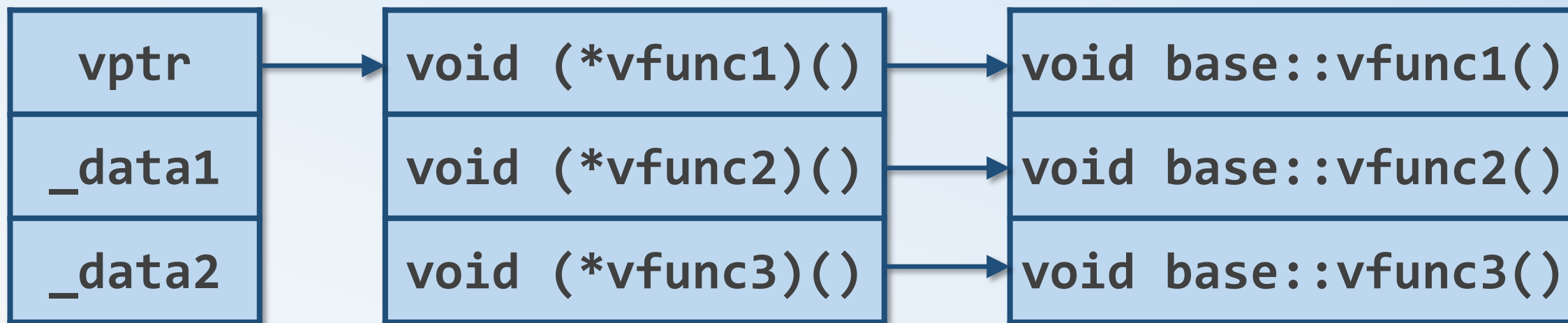
```
class base
{
public:
    func();
    virtual void vfunc1();
    virtual void vfunc2();
    virtual void vfunc3();
private:
    int _data1;
    int _data2;
};
```

隐藏、覆盖、重载

- base对象实例在内存中占据的空间是这样的:

base对象实例

虚函数表



隐藏、覆盖、重载

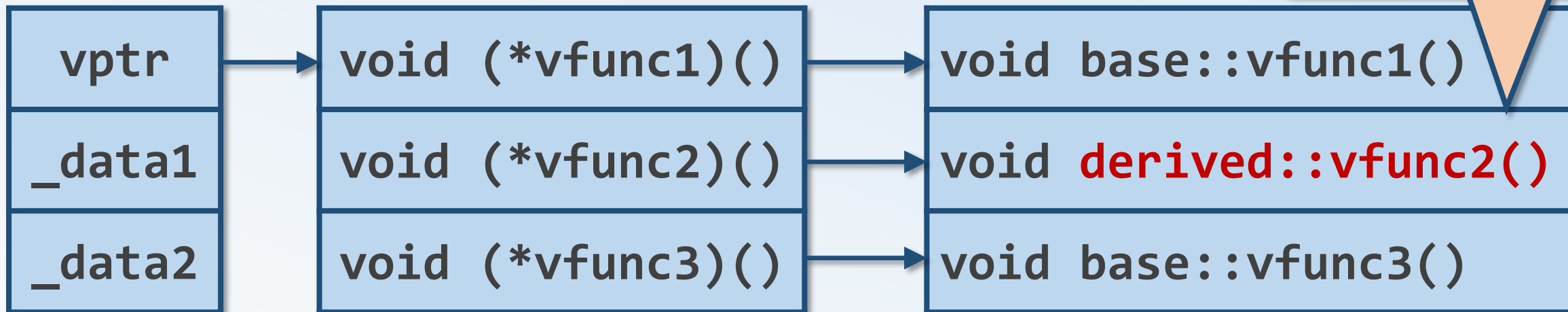
- 当派生类改写了虚函数时,虚函数表相应的被修改了:

```
class derived: public base {  
public:  
    void vfunc2();  
};
```

这里变了!

base对象实例

虚函数表



隐藏、覆盖、重载

➤ 所以当写下如下程序的时候:

```
void main(void)
{
    Derived d;
    Base * pb = &d;
    pb->vfunc2(); // Derived::vfunc2(void)
}
```

隐藏、覆盖、重载

❖ 函数的重载

重载规则：

1. 相同的作用域（不同类的同名函数不是重载）。
2. 函数名相同。
3. 参数个数、类型、顺序、const限定的指针或引用、是否为常成员函数。
4. virtual及返回值等其他因素不能作为重载依据

隐藏、覆盖、重载

❖ 重载性重载规则：

1. 构造函数可以重载
2. 析构函数不能重载
3. 一般成员函数可以重载

隐藏、覆盖、重载

```
class Person {  
public:  
    Person(string aName,bool aSex,int aAge);// 构造函数重载  
    Person(string aName,bool aSex);  
    Person(string aName);  
    Person();  
    void set(string aName,bool aSex,int aAge);// 一般成员函数重载  
    void set(bool aSex,int aAge);  
    void set(int aAge);  
protected:  
    string m_sName; // 姓名  
    bool m_bSex; // 性别  
    int m_iAge; // 年龄  
};
```


隐藏、覆盖、重载

```
int main(void) {  
    Person p1; // 调用无参数的构造函数  
    p1.set("张三", true, 22);  
    p1.display();  
  
    Person p2("李四", true); // 调用带参数的构造函数  
    p2.set(23);  
    p2.display();  
  
    Person p3("王五"); // 调用带参数的构造函数  
    p3.set(false, 24);  
    p3.display();  
    return 0;  
}
```

隐藏、覆盖、重载

类A中的f(a)被隐藏的情况：

class A	class B:public A
f(a)	f(a)

```
B b;  
b.f(a);
```

class A	class B:public A
f(a)	f(a,b)

```
B b;  
b.f(a,b);  
b.f(a); // 错误
```

类B中的f(a,b)被隐藏的情况：

class A	class B:public A
f(a)	f(a,b)

```
A *a = new B();  
a->f(a,b); // 错误
```

类A中的f(a)被覆盖的情况：

class A	class B:public A
virtual f(a)	f(a)

```
B b;      A *a = &b;  
a->f(a);  b.f(a);
```



本讲教学目标

- 掌握C++中虚函数的概念
- 理解重载、隐藏、覆盖



THANKS

