



程序设计实习

郭炜 微博 <http://weibo.com/guoweiofpku>

<http://blog.sina.com.cn/u/3266490431>

刘家瑛 微博 <http://weibo.com/pkuliujiaying>



使用多态的游戏程序实例(P250)

游戏《魔法门之英雄无敌》

游戏中有很多种怪物，每种怪物都有一个类与之对应，
每个怪物就是一个对象。



类: CSoldier



类CPhonex

类: CDragon



类: CAngel

游戏《魔法门之英雄无敌》

怪物能够互相攻击，攻击敌人和被攻击时都有相应的动作，动作是通过对象的成员函数实现的。



游戏《魔法门之英雄无敌》

游戏版本升级时，要增加新的怪物——雷鸟。
如何编程才能使升级时的代码改动和增加量较小？



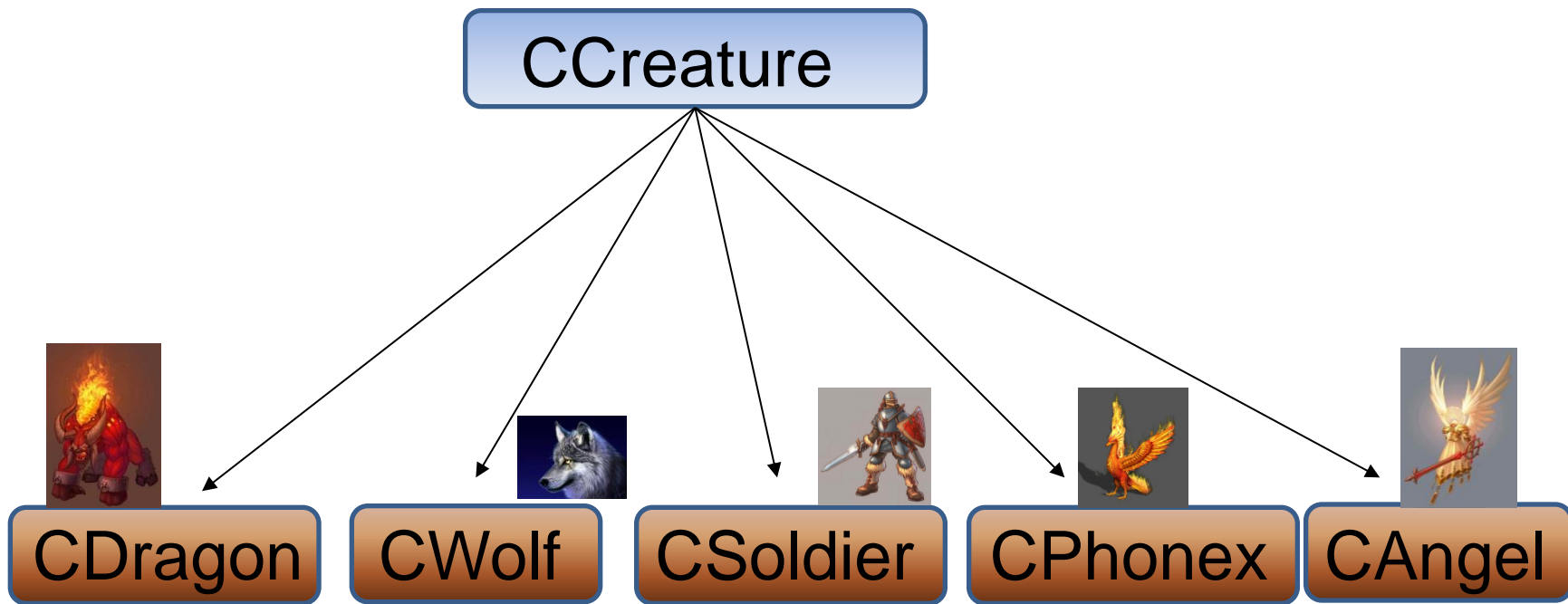
新增类：CThunderBird

基本思路：

- 为每个怪物类编写 `Attack`、`FightBack`和 `Hurted`成员函数。
- `Attact`函数表现攻击动作，攻击某个怪物，并调用被攻击怪物的 `Hurted`函数，以减少被攻击怪物的生命值，同时也调用被攻击怪物的 `FightBack`成员函数，遭受被攻击怪物反击。
- `Hurted`函数减少自身生命值，并表现受伤动作。
- `FightBack`成员函数表现反击动作，并调用被反击对象的`Hurted`成员函数，使被反击对象受伤。

基本思路：

设置基类 C Creature，并且使 C Dragon，C Wolf 等其他类都从 C Creature 派生而来。



非多态的实现方法

```
class class C Creature {
    protected:          int nPower ; //代表攻击力
                        int nLifeValue ; //代表生命值
};
class C Dragon:public C Creature {
    public:
        void Attack(C Wolf * pWolf) {
            . . . 表现攻击动作的代码
            pWolf->Hurted( nPower);
            pWolf->FightBack( this);
        }
        void Attack( C Ghost * pGhost) {
            . . . 表现攻击动作的代码
            pGhost->Hurted( nPower);
            pGohst->FightBack( this);
        }
}
```


非多态的实现方法

```
void Hurted ( int nPower) {  
    . . . . 表现受伤动作的代码  
    nLifeValue -= nPower;  
}  
void FightBack( CWolf * pWolf) {  
    . . . . 表现反击动作的代码  
    pWolf ->Hurted( nPower / 2);  
}  
void FightBack( CGhost * pGhost) {  
    . . . . 表现反击动作的代码  
    pGhost->Hurted( nPower / 2 );  
}  
}
```

➤有n种怪物，CDragon 类中就会有n个 **Attack** 成员函数，以及 n个**FightBack** 成员函数。对于其他类也如此。

非多态的实现方法的缺点



- 如果游戏版本升级，增加了新的怪物雷鸟 CThunderBird，则程序改动较大。

- 所有的类都需要增加两个成员函数：

```
void Attack( CThunderBird * pThunderBird) ;  
void FightBack( CThunderBird * pThunderBird)
```

- 在怪物种类多的时候，工作量较大有木有！！！！

抓狂啦！



多态的实现方法

//基类 C Creature:

```
class C Creature {  
    protected :  
        int m_nLifeValue, m_nPower;  
    public:  
        virtual void Attack( C Creature * pCreature) {}  
        virtual void Hurted( int nPower) { }  
        virtual void FightBack( C Creature * pCreature) { }  
};
```

- 基类只有一个 Attack 成员函数；也只有一个 FightBack 成员函数；所有 C Creature 的派生类也是这样。

多态的实现方法

//派生类 CDragon:

```
class CDragon : public C Creature {  
    public:  
        virtual void Attack( C Creature * pCreature);  
        virtual void Hurted( int nPower);  
        virtual void FightBack( C Creature * pCreature);  
};
```

多态的实现方法

```
void CDragon::Attack(CCreature * p)
{
    ...表现攻击动作的代码
    p->Hurted(m_nPower); //多态
    p->FightBack(this); //多态
}

void CDragon::Hurted( int nPower)
{
    ...表现受伤动作的代码
    m_nLifeValue -= nPower;
}

void CDragon::FightBack(CCreature * p)
{
    ...表现反击动作的代码
    p->Hurted(m_nPower/2); //多态
}
```

多态实现方法的优势



- 如果游戏版本升级，增加了新的怪物雷鸟 CThunderBird……

只需要编写新类CThunderBird， 不需要在已有的类里专门为新怪物增加：

```
void Attack( CThunderBird * pThunderBird) ;  
void FightBack( CThunderBird * pThunderBird) ;
```

成员函数，**已有的类可以原封不动**，没压力啊！！



原理

CDragon Dragon; CWolf Wolf; CGhost Ghost;

CThunderBird Bird;

Dragon.Attack(& Wolf); //(1)

Dragon.Attack(& Ghost); //(2)

Dragon.Attack(& Bird); //(3)



- 根据多态的规则，上面的(1)，(2)，(3)进入到CDragon::Attack函数后，能分别调用：

CWolf::Hurled

CGhost::Hurled

CBird::Hurled

```
void CDragon::Attack(CCreature * p)
{
    p->Hurled(m_nPower); //多态
    p->FightBack(this); //多态
}
```