C#中易混淆概念:解析里氏替换原则,虚方法 - 文章 - 伯 乐在线



原文出处: zhiqiang21

今天开始上班了。这几天研究学习了一下思维导图,感觉用它整理自己的知识非常的方便。所以,以后写博客完成一个知识点,都会用思维导图做一个总结。也能让大家对所要读的内容有一个整体的把握。

我用的思维导图软件是FreeMind(免费的,但是得装JDK),因为刚开始学习使用,很多操作技巧不是很熟练,做出来的导图估计也不是很好,希望大家见谅。

首先, 里氏替换原则。

这是理解多态所必须掌握的内容。对于里氏替换原则维基百科给出的定义如下:

里氏替换原则的内容可以描述为: "派生类(子类)对象能够替换其基类(超类)对象被使用。" 以上内容并非利斯科夫的原文,而是译自<mark>罗伯特·马丁</mark>(Robert Martin)对原文的解读。其原文为:

为什么子类可以替换父类的位置,而程序的功能不受影响呢?

当满足继承的时候,父类肯定存在非私有成员,子类肯定是得到了父类的这些非私有成员(假设,父类的的成员全部是私有的,那么子类没办法从父类继承任何成员,也就不存在继承的概念了)。既然子类继承了父类的这些非私有成员,那么父类对象也就可以在子类对象中调用这些非私有成员。所以,子类对象可以替换父类对象的位置。

来看下面的一段代码:

```
C#
```

```
class Program
{
    static void Main(string[] args)
    {
        Person p = new Person();
        Person p1 = new Student();
        Console. ReadKey();
    }
    class Person
    {
        //父类的私有成员
        private int nAge;
        public Person()
```

Console. WriteLine ("我是Person构造函数,我是一个人!");

```
public void Say()
                     Console. WriteLine ("我是一个人!");
       class Student : Person
              public Student()
                      Console. WriteLine ("我是Student构造函数,我是一个学生!");
              public void SayStude()
                     Console. WriteLine ("我是一个学生!");
       }
       class SeniorStudent : Student
              public SeniorStudent()
                      Console. WriteLine ("我是SeniorStudent构造函数,我是一个高中生!");
              public
                       void SaySenior()
                     Console. WriteLine ("我是一个高中生!");
       }
我们运行打印出的结果是:
  O I SIM
  static void Main(string[] args)
      Person p = new Person();
                                ■ file:///D:/百度网盘/练习Demo/visualDemo/CCharp难理
      Person p1 = new Student();
                                  ₽Student≀
      Console.ReadKey();
```

根据前面的构造函数的知识很容易解释这个结果。那么我们在Main()函数中添加如下的代码:

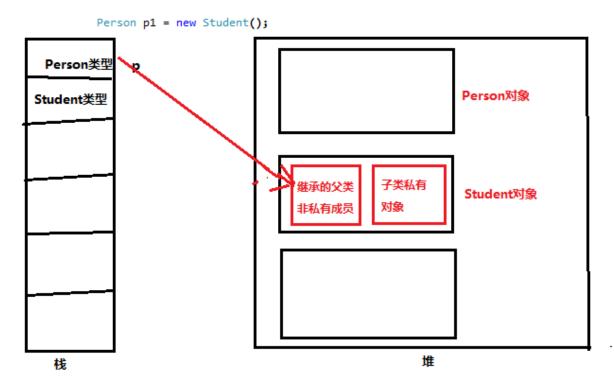
http://blog.jobbole.com/101265/

}

C#

```
static void Main(string[] args)
                   Person p = new Person();
                   p. Say();
                   Person p1 = new Student();
                   p1. Say();
                   Console. ReadKey();
在访问的过程中,可以发现p只可以访问父类的say
   static void Main(string[] args)
      Person p = new Person();
                      ent();
      Pe ⊕ Equals
      父类对象只可以访问父类的方法
        }
}
          Say
                       void Person.Say()
        4 个引用
而p1也只可以访问父类的Say方法
     0 15175
     static void Main(string[] args)
        Person p = new Person();
        p.Say();
        Person p1 = new Student();
        p1.
        }
                        可以看到p1只可以访问父类的Say方法
          5 个引用
                         void Person.Say()
  class Perso
```

其实在上面的代码中,就满足了里氏替换原则。子类的Student对象,替换了父类Person对象的位置。那么它们在内存中发生了些什么呢?如下图:



由上可以知道,当一个父类的变量指向一个子类对象的时候只能通过这个父类变量调用父类成员,子类独有的成员无法调用。

同理我们可以推理出,子类的变量是不可以指向一个父类的对像的



但是当父类变量指向一个子类变量的时候,可以不可以把父类的变量转化成子类的对象呢?看下图

```
Person p - new Person();
   13
                p.Say();
  14
                 //满足里氏替换原则子类对象替换父类对象的位置
   15
                 Person p1 - new Student();
                 //将指向子类对象的变星转化成子类类型
   16
                                               ■ file:///D:/百度网盘/练习Demo/visualDemo/CCh
   17
                Student s2 = (Student)p1;
                                                         勾造函数,我<del>是</del>·
                                                   Person构造函数
显示输出来源(S): 生成
                                                  Student构造函数,
      已启动全部重新生成: 项目:生氏替换,配置:Debig Amy CPU:
1>D:\严度网盘\练习Demo\visualDemo\CCharp难理解概念总结1/里氏替换/里
  |里氏替换||>> D:/百度网盘/练习Demo/visualDemo/CCkarp难理解概念总统
       全部重新生成。 成功 1 个,失败 0 个,跳过 0 个
                                          ,运行不报错 ,说明可以将指向子类对象的父类变量转化成子类类型
```

关于引用类型的两种转换方式:

由上面的代码我们已经知道了一种转换,就是在变量钱直接加需要转换的类型,如下代码:

```
С#
```

打印结果如下:

```
Person p = new Person();
p.Say();

Person p1 = new Student();
p1.Say();

Student s = new Student();
s.Say();
Console.ReadKey();
}
```

我们很明显的可以发现,第二个表达式满足里氏替换原则, p1. Say()执行的应该是父类的Say()方法, 但是这里却执行了子类的Say()方法。

这就是子类使用override关键字的Say()方法覆盖了父类的用Virtual关键字修饰的Say()方法。

我们使用动态图片看一下调试过程,

①首先是没有使用任何关键字:

```
1 7175179
    27
                 public Person()
    28
                    Console.WriteLine("我是Person构造函数,我是一个人!");
    29
    30
                 }
    31
    32
                 2 个引用
                 public void Say()
    33
    34
    35
                    Console.WriteLine("我是一个人!");
    36
                }
    37
    38
             }
    39
             4 个引用
    40
             class Student : Person
    41
    42
                 1 个引用
    43
                 public void Say()
    44
                    Console.WriteLine("我是一个学生!");
    45
    46
                 }
    47
100 %
```

由上可以看出直接跳入父类,执行了父类的Say()方法;

②再看使用virtual和override关键字的动态调试图片,如下:

```
CIASS Program
 o =
 9
         {
            0 个引用
10 🖹
             static void Main(string[] args)
11
                Person p = new Person();
12
13
                p.Say();
14
15
                 Person p1 = new Student();
16
                 p1.Say();
17
18
                Student s = new Student();
19
                s.Say();
20
                Console.ReadKey();
21
            }
22
        }
23
        5 个引用
24
         class Person
25
26
             private int nAge;
             1 个引用
27
             public Person()
28
                Console.WriteLine("我是Person构造函数,我是一个人!");
29
```

可以看到直接到子类去执行override关键字修饰的Say()方法。

那么如果父类使用virtual关键字修饰,而子类没有重写该方法时会怎么样呢?如下面的代码:

```
C#
class Program
{
    static void Main(string[] args)
    {
        Person p1 = new Student();
        p1. Say();
}
```

```
Console. ReadKey();
      }
      class Person
             private int nAge;
             public Person()
                    Console. WriteLine ("我是Person构造函数,我是一个人!");
             //这里定义了一个虚方法
             public virtual void Say()
                    Console. WriteLine ("我是一个人!");
      class Student : Person
             //子类中没有出现override关键字修饰的方法
             public void SayStude()
                    Console. WriteLine ("我是一个学生!");
执行结果如下:
    0 1151 #3
    static void Main(string[] args)
       Person p1 = new Student ().
                             ■ D:\百度网盘\练习Demo\visualDemo\CCharp难理解
       Console.ReadKey();
                                  Person构造函数,我是
    }
}
3 个引用
```

所以,如果子类找不到override方法,则会回溯到该子类的父类去找是否有override方法,知道回溯到自身的虚方法,并执行。

虚方法知识总结的思维导图如下:

## 使用virtual关键字修饰虚方法

虚方法

使用override关键字在子类中覆写父类的虚方法

如果子类覆写了父类的虚方法,那么通过父类调用父类的虚方法的时候,回去调用子类重写后的方法

如果子类中找不到override覆写后的方法, 则回溯到父类去调用父类的虚方法

拿高薪,还能扩大业界知名度!优秀的开发工程师看过来 ->《高薪招募讲师》

1 赞 1 收藏 评论



## 合作联系

Email: <a href="mailto:bd@Jobbole.com">bd@Jobbole.com</a>

QQ: 2302462408 (加好友请注明来意)

## 更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子

头条 - 分享和发现有价值的内容与观点

相亲 - 为IT单身男女服务的征婚传播平台

资源 - 优秀的工具资源导航

翻译 - 翻译传播优秀的外文文章

文章 - 国内外的精选文章

设计 - UI,网页,交互和用户体验

iOS - 专注iOS技术分享

安卓 - 专注Android技术分享

前端 - JavaScript, HTML5, CSS

Java - 专注Java技术分享

Python - 专注Python技术分享