目 录

1 多态 **2**
1.1 绑定 2
1.2 多态的缺陷 3
1.2.1 private 函数无法动态绑定 3
1.2.2 域无法动态绑定 3
1.3 协变返回类型 4

1 多态

多态又称为动态绑定，和 C++ 的多态类似。在讨论多态之前，先感受一下多态的特性。例子如下：

```
class Instrument
{
    public void play()
    {
        System.out.println("Instrument.play()");
    }
}

class Wind extends Instrument
{
    public void play()
    {
        System.out.println("Wind.play()");
    }
}

public class Music
{
    public static void tune(Instrument i)
    {
        i.play();
    }
    public static void main(String[] args)
    {
        Wind flute = new Wind();
        // tune接受Instrument类型
        // 将Wind转为Instrument类型
        // 输出的是: Wind.play()
        tune(flute);
    }
}
```

从这个例子可以看出一个多态的现象：虽然 tune 函数接受一个 Instrument 引用，但是它知道这个 Instrument 引用指向的是 Wind 对象。正是动态绑定实现了这项特性。

1.1 绑定

将一个函数调用和一个函数主体关联起来称为绑定。绑定有两种类型，如下：

- 前期绑定。在程序执行前就将一个函数调用和一个函数主体关联起来。
- 后期绑定，又称为动态绑定。在程序运行时根据对象的类型进行绑定。

Java 中除了 static 方法和 final 方法，其他所有方法都是后期绑定的。

1.2 多态的缺陷

1.2.1 private 函数无法动态绑定

程序不能对 private 函数进行动态绑定。这是因为 private 函数是 final 函数，而且导出类无法覆盖基类中的 private 函数。例子如下：

```
class PrivateOvrride
{
    private void f()
    {
        System.out.println("private f()");
    }

    public static void main(String[] args)
    {
        PrivateOvrride po = new Derived();
        po.f(); // 不会指向Derived类中的f()，而是指向PrivateOvrride类中的f()
    }
}

public class Derived extends PrivateOvrride
{
    public void f()
    {
        System.out.println("public f()");
    }
}
```

为了避免造成混乱的代码，导出类中的函数名不要和基类中的 private 函数名相同。

1.2.2 域无法动态绑定

和 C++ 一样，Java 中域是无法动态绑定的。也就是说，程序无法根据对象的类型选择相应的域。例子如下：

```
class Super
{
    public int field = 0;
}

Sub extends Super
{
    public int field = 1;
}

public class FieldAccess
{
    public static void main(String[] args)
    {
        Super sup = new Sub();
        System.out.println(sup.field); // 输出0
```

```
17              Sub sub = new Sub();
18              System.out.println(sub.field); // 输出1
19          }
20      }
```

为了避免造成混乱的代码，不要把基类中的域和导出类的域赋予相同的名字。

1.3 协变返回类型

协变返回类型表明，子类覆写基类方法时，返回的类型可以是基类方法返回类型的子类。

```
1   class Grain
2   {
3       public String toString()
4       {
5           return "Grain";
6       }
7   }
8
9   class Wheat extends Grain
10  {
11      public String toString()
12      {
13          return "Wheat";
14      }
15  }
16
17  class Mill
18  {
19      Grain process()
20      {
21          return new Grain();
22      }
23  }
24
25  class WheatMill extends Mill
26  {
27      Wheat process()
28      {
29          return new Wheat();
30      }
31  }
32
33  public class CovariantReturn
34  {
35      public static void main(String[] args)
36      {
37          Mill m = new Mill();
38          Grain g = m.process();
39          System.out.println(g); // 输出Grain
40          m = new WheatMill();
41          g = m.process();
```

```
42                 System.out.println(g); // 输出Wheat
43             }
44         }
```