浅析 Java 中的封装性

摘要： oop 思想是整个应用型软件开发的核心，不管使用.net 还是 java，只有掌握了 oop 思想，才能快速有效地为客户提供企业级的解决方案，而封装性又是 oop 思想中不可缺少的基础组成部分，所以在使用 java 时，理解和掌握封装性是非常重要的，对以后学习和掌握其他面向对象的编程技术也尤为重要。

关键词： java；封装性；软件开发

中图分类号：tp311.5 文献标识码：a 文章编号：

1. 引言

目前， 面向对象的思想被软件开发界广为追捧， 其实，面向对象思想不是高深的理论， 而是根据前人大量编程项目总结出来的一套分析和解决编程方面问题的方法。以面向对象思想为指导， 可以优化 java 代码的结构， 更可以让数据库访问模块变得富有弹性—即让代码和模块能更好地适应项目需求的频繁变化。面向对象编程（object oriented programming， 简称 oop）描述的是对象之间的相互作用。oop 编程主要有 3 大思想，分别是封装、继承、多态。在面向对象编程中， 类作为最小程序单元， 就像以往面向过程编程中函数作为最小程序单元一样。

封装性是面向对象程序设计的原则之一。它规定对象应对外部环境隐藏它们的内部工作方式。良好的封装提高了代码的模块性， 它防止了对象之间不良的相互影响， 这样就使得未来的开发工作变

得容易。

2. 定义

封装就是隐藏实现细节， 将属性私有化， 提供公有方法访问私有属性。

类是基于面向对象思想编程语言的基础， 程序员可以把具有相同业务性质的代码封装到一个类里， 通过接口方法向外部代码提供服务， 同时向外部代码屏蔽类里服务的具体实现方式。对象是类的实例， 类一般是个抽象名词， 比如“人类”， 而对象则是具体的物质存在， 比如“张三” 这个人。在现实生活中， 经常会遇到“类” 和“对象” 这些概念， 比如封装了能实现“全自动洗衣机” 功能的洗衣机类。这里， 用户可以使用面板上的按钮，使用该类里提供的洗衣等服务，并且， 由于该类实现了封装， 所以在使用的时候， 用户可以不用了解其中的自动洗衣原理以及实现机制。

类是同一种类型的对象的抽象， 是某种类型对象的概述和定义；而对象则是某个类的实例化结果或者叫一种类型的实体。在使用面向对象的思想进行软件开发的过程中， 首先得抽出项目的实体-对象模型：

即首先是实体类的定义、封装。

在 java 中， 最基本的封装单元是类， 一个类定义着将由一组对象所共享的行为(数据和代码) .一个类的每个对象均包含它所定

义的结构与行为， 这些对象就好象是一个模子铸造出来的。在定义一个类时， 需要指定构成该类的代码与数据，

特别是类所定义的对象叫做成员变量或实例变量。操作数据

的代码叫做成员方法。方法定义怎样使用成员变量， 这意味着类的行为和接口要由操作实例数据的方法来定义。由于类的用途是封装复杂性， 所以类的内部有隐藏实现复杂性的机制。所以 java 中提供了私有和公有的访问模式， 类的公有接口代表外部的用户应该知道或可以知道的每件东西。私有的方法数据只能通过该类的成员代码来访问， 这就可以确保不会发生不希望的事情。

3.必要性

建构具有良好封装性能的对象的第一个步骤就是 getter/setter 对访问私有数据域。通过一个控制的机制来要求那些要对对象的域进行读和写的其他对象这样去做， 可以加强法定值和内部数据的连贯性。如果有一个域名为 duration，它应该保存一个正整数，当另一个对象试图进行 setduration (-4) 时，就可以给出一个 illegalargumentexception。如果将 duration 成员设置为公有，就不能防止某人调用 yourobject.duration=-4 并搞乱数据的内部连贯性。通过一个例子来理解这个问题： 编写一个教师类， 要求：

(1) 具有属性： 姓名、年龄

(2) 具有行为： 自我介绍

(3) 教师的最小年龄要求： 22 岁

```
public class teacher {
public string name; // 教员姓名
public int age; //年龄
/**
* 返回自我介绍的内容
*/
public string introduction () {
return “ 大家好！我是” + name + “，我今年” + age+” “ ;
}}
```

编写一个测试类， 要求： 实例化一个教员对象， 并对其初始化， 并在控制台输出该教员的自我介绍。

```
public class teachertest {
public static void main (string [ ] args) {
teacher teacher = new teacher () ;
teacher.name = “ 张三” ;
teacher.age =9;
system.out.println (teacher.introduction ()) ;
}
}
```

运行结果为： 大家好！ 我是张三， 我今年 9 岁

因为没对属性进行封装， 所以运行结果不满足教员的最小年龄

要求， 要满足此要求， 就需要通过对属性的封装来实现，修改 teacher 类如下：

```
public class teacher {
private string name; // 教员姓名
private int age; //年龄
public int getage () {
return age;
}
public void setage (int age) {
if (age<22) {
system.out.println (“ 错误！最小年龄应为 22 岁！”) ;
this.age = 22; //如果不符合年龄要求，则赋予默认值
} else {
this.age = age;
}}
public string getname () {
return name; // 返回教员姓名
}
public void setname (string name) {
this.name = name; // 设定教员姓名
}
```

```
public string introduction () {
return “ 大家好！我是” + name + “，我今年” + age+” 岁” ;
}}
```

现在同样执行以上测试类 teachertest，运行结果为：错误！最小年龄应为 22 岁！

大家好！ 我是张三， 我今年 22 岁

4. 封装

从以上事例可得出： 使用封装， 增加了数据访问限制，增强了程序的可维护性， 其封装步骤为：

（1） 修改 name、age 属性的可见性为 private 来限制直接对属性的访问。

（2） 为每个属性创建一对赋值(setter) 方法和取(getter)方法， 用于间接对这些属性的访问。

（3） 在 setter 和 getter 方法中， 加入对属性的存取限制。

5. 避免 java 封装性问题