面向对象分析与设计

作业一

程序设计语言的 00 特性

调研报告

学号: 1601214522

姓名: 孙鹏晖

日期: 2017/3/3

研究背景

面向对象程序设计(Object-Oriented Programming, OOP)是一种程序设计范型，也是一种程序设计开发的方法. OOP 将问题域中的概念合理抽象为类和对象，并将类和对象作为程序的基本单元，封装程序的数据和方法，以提高软件的重用性，灵活性和扩展性.

而今面向对象的方法已经几乎深入到计算机软件领域的所有分支，我们提到面向对象的概念，不仅是指一些具体的软件开发技术和策略，而且是一套关于如何看待软件系统与现实世界的关系，用什么观点来研究问题并进行问题求解和如何进行软件系统构造的软件方法学.

面向对象的语言出现之后，面向对象的思想得到了迅速的发展，1967 年的 Simula67 语言首先引入了类的概念和继承机制，是面向对象的先驱. 1972 年发布的 Smalltalk-72 正式使用了面向对象的术语标志着面向对象程序设计方法正式形成. 此后面向对象的语言逐渐走向繁荣，例如 C++, Objective-C, Java, Python 等，既有纯 OO 型语言也有混合型 OO 语言，面向对象方法如今已经得到了巨大发展，面向对象方法也从编程发展到设计、分析，进而发展到整个软件生命周期.

调研目的

1. 了解并熟悉面向对象方法和工具的发展历程及发展现状;
2. 熟悉并掌握面向对象思想和方法的主要特征和相关概念内容;
3. 针对目前种类繁多的面向对象程序设计语言，选择两种自己不甚熟悉和比较熟悉的程序设计语言，在对比分析其 OO 特性的过程中，体会面向对象思想.

调研要求

面向对象已经为很多程序设计语言所支持，但不同语言的面向对象特征和关注重点并不尽相同. 请调研至少 2 种你之前了解甚少的语言，了解其设计思想和面向对象特征，并和一门你比较熟悉的语言如 Java, C++, C# 做比较，给出代码示例，提交一份调研报告.

调研范围包含但不限于:

Objective-C, Go, Python, Ruby, Scala, Swift, Erlang, PHP, Smalltalk, Java, C++, C#

研究内容

1. 面向对象特性

面向对象设计是一种把面向对象的思想应用于软件开发过程中，指导开发活动的系统方法，建立在对象概念之上的方法学. 将问题域中的客观实体进行抽象，对象是由数据和方法组成的抽象封装体，对应于客观实体，一个对象类定义了具有相似性质的一组对象，面向对象程序设计具有抽象性，封装性，继承性和多态性等特征.

抽象：指从事物中舍弃个别的、非本质的特征，而抽取共同的、本质特征的思维方式.

封装：将数据和代码捆绑到一起，避免了外界的干扰和不确定性. 对象的某些数据和代码可以是私有的，不能被外界访问，以此实现对数据和代码不同级别的访问权限.

继承：让某个类型的对象获得另一个类型的对象的特征. 通过继承可以实现代码的重用，从已存在的类派生出的一个新类将自动具有原来那个类的特性，同

时，它还可以拥有自己的新特性.

多态：指一般类和特殊类可以有相同格式的属性或操作，但这些属性或操作具有不同的含义，即具有不同的数据类型或表现出不同的行为.

2. 面向对象语言

根据作业要求，选择两种之前了解甚少的语言进行调研，在此选择 C#和 PHP 语言作为研究对象. 同时选择自己比较熟悉的 Java 语言进行对比分析.

PHP 不是一种纯面向对象的编程语言，而是一种混合型语言，因此使用 PHP 语言进行编程时既可以使用面向对象方法，也可以使用传统的过程化方法. 而 C#与 Java 相似，其中的类型都来自于根对象 Object，可以被认为是纯面向对象的编程语言.

3. 面向对象分析

在分析开始之前，我们需要在一个具体的问题域中对各种面向对象的编程语言进行面向对象相关特性的分析，因此我们假设问题的情境如下:

在一所古老的学校里，有一群人，这群人里有些是老师，也有些是学生，老师有职称，学生有学号，每个老师和同学都有自己的名字，老师可以创建一门新的课程，学生可以选择自己要学的课程，学生和老师都可以查看自己的课程信息.

其实这个问题描述就是一个超级简化版的选课系统，虽然被当做例子举了无数次，但这样一个简单的问题情境很有利于我们对面向对象方法和特性进行理解. 我们基于这样一个简单清晰的例子，来调研几种不同的程序设计语言的 OO 特性.

抽象性:

在问题情境中，“有一群人”这样的描述，人这个客观实体在自然界中是事实存在的，我们都是人，我们身边也有其他的人，但是我们在程序设计的时候不能把人这样一个客观实体塞进程序代码中，因此我们需要从人这样一个客观实体中进行抽象，舍弃一些与问题无关的特性，突出问题相关属性，用这样一个抽象的类来表示现实世界中存在的人，一个表示人的类可以这样定义:

```
//PHP
class Person{
}
```

在 C#中定义一个人的类:

```
//C#
class Person{
}
```

而在 Java 中定义一个人的类写法是:

```
//Java
class Person{
}
```

PHP，C#以及 Java 三种编程语言的类的定义写法基本一致，以 class 为关键字，通过这种方式，我们就由现实世界人这个客观实体抽象出了面向对象的程序设计中的人这个类的概念.

封装性:

封装性是面向对象编程的重要特性之一，封装性就是把对象的属性和方法结合成一个相对独立的单位，并尽可能隐藏内部的实现细节，一方面把对象的全部属性和方法结合在一起，形成一个不可分割的独立整体，也即对象；另一方面对外形成一个边界，尽可能隐藏对象内部的实现细节从而做到信息隐蔽. 例如在我们的例子中，每个人都有自己的名字，也都可以查看自己的课程信息，我们要给每个人添加这些信息，并且我们不希望这些信息随便被别人访问到，那我们就需要对这些属性和方法进行一定的封装，例如:

```
//PHP
class Person{
    private $name; //人的名字
    public function viewCourse(){ //查看课程信息方法
    }
}
```

在 PHP 中被 private 关键字修饰的类属性和方法不能在类外被直接访问，没有访问控制的默认是 public，即在任何地方都可以被直接访问. 此外还有 protected 关键字限制只能在本类或者派生类中可以被访问.

在 C#语言中:

```
//C#
class Person{
    private string name; //人的名字
    //查看课程信息方法(虚方法用于子类重写)
    public virtual string viewCourse(){
```

```
    }
}
```

在 C#中被 private 修饰的类属性和方法只能在类内被访问，使用 public 修饰的内容可以在类内类外被访问，使用 protected 修饰的内容可以在类内和派生类中被访问.

在 Java 语言中:

```
//Java
class Person{
    private String name; //人的名字
    public String viewCourse(){ //查看课程信息方法
    }
}
```

除了类似于上述的 private, public 和 protected 之外, Java 对于无访问控制关键字修饰的内容默认为包访问权限.

继承性:

继承性在面向对象领域有着极其重要的作用, 继承是指建立一个新的派生类, 从先前定义的类中继承数据和方法，也可以重新定义或加进新数据方法. 继承性是子类自动共享父类数据结构和方法的机制，是类间的一种关系. 定义和实现一个类, 可以在已经存在的类的基础上进行, 把已存在的类的内容作为自己的内容, 并加入新的内容. 比如我们在上面的例子中定义了 Person 类，这个类里面有成员属性 name 和成员方法 viewCourse(), 而实际上我们的问题域中真正的角色是

老师和学生，因为老师和学生的也是 Person，所以它们也有成员属性 name 以及成员方法 viewCourse()，这时就可以让 Teacher 类和 Student 类来继承 Person 类，Teacher 类和 Student 类把 Person 类里面的属性继承过来，不用重新声明，而 Student 类里面还有学号属性和选课方法，所以在 Student 类里面除了继承 Person 类的属性和方法外还要加上 Student 类特有的”学号属性“和”选课方法“，这样一个 Student 类就声明完成了．通过继承机制，可以利用已有的数据类型来定义新的数据类型，所定义的新的数据类型不仅拥有新定义的成员，而且还同时拥有旧的成员．我们称已存在的用来派生新类的类为基类，又称为父类或超类，由基类派生出的新类称为派生类或子类．

通过以上分析我们通过不同语言的代码的具体实现来观察不同语言的继承性的实现．

在 PHP 中：

```
//PHP
class Teacher extends Person{ //Teacher 类继承 Person 类
    private $title; //老师的职称
    function createCourse($courseInfo){ //创建课程方法
        ……
    }
}
//#############################################################
class Student extends Person{ //Student 类继承 Person 类
    private $ID; //学生学号
```

```
    function selectCourse($courseID){ //学生选课方法
        ......
    }
}
```

在 C#中:

```
//C#
class Teacher:Person{ //Teacher 类继承 Person 类
    private string title; //老师的职称
    public string createCourse(string courseInfo){ //创建课程方法
        ......
    }
}
//############################################################
class Student:Person{ //Student 类继承 Person 类
    private string ID;
    public string selectCourse(string courseID){ //选课方法
        ......
    }
}
```

C#在继承性的实现上采用了类似于 C++的实现方法，使用操作符:表示继承.

在 Java 语言中:

```
//Java
```

```
class Teacher extends Person{ //Teacher 类继承 Person 类
    private String title; //老师职称
    public String createCourse(String courseInfo){ //创建课程方法
        ……
    }
}
//#############################################################
class Student extends Person{ //Student 类继承 Person 类
    private String ID; //学生学号
    public String selectCourse(String courseID){ //选课方法
        ……
    }
}
```

Java 语言在继承性实现上与 PHP 一样采用了 extends 关键字方式，并且我们在 Teacher 类和 Student 类继承 Person 基类基础上添加了各自新的方法和属性.

面向对象中的继承性使所建立的软件具有开放性、可扩充性，它简化了对象、类的创建工作量，增加了代码的可重用性. 采用继承，提供了类的规范的等级结构. 使公共的特性能够共享，提高了软件的重用性.

在不同的程序设计语言中，继承性的具体实现方式有所不同，例如，在 C++语言中，一个派生类可以从一个基类派生，也可以从多个基类派生. 从一个基类派生的继承称为单继承；从多个基类派生的继承称为多继承. 而我们调研的 PHP，C#和 Java 语言都属于单继承类型语言.

多态性:

多态是面向对象的一个重要特性，从某种角度来看，封装和继承都是在为多态性进行准备．多态是指允许不同类的对象对同一消息做出响应，即同一消息可以根据发送对象的不同而采用多种不同的行为方式．实现多态的技术称为动态绑定，是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法．要实现多态性，程序代码要实现类的继承，子类对父类方法进行重写，最后实例化对象使父类引用指向子类对象．我们通过调研的几种语言来观察多态性的体现.

```
//PHP
class Teacher extends Person{
    private $title; //老师的职称
    function createCourse($courseInfo){ //创建课程方法
        ……
    }
    function viewCourse(){ //重写查看课程信息方法
        echo “老师的课程信息…”；
        ……
    }
}
//#############################################################
class Student extends Person{
```

```
    private $ID; //学生学号
    function selectCourse($courseID){ //学生选课方法
        ……
    }
    function viewCourse(){ //重写查看课程信息方法
        echo “学生的课程信息”;
        ……
    }
}
//#############################################################
function printCourseInfo($Obj){ //多态性测试函数
    if($Obj instanceof Person){
        $Obj->viewCourse();
    }
    ……
}
//#############################################################
printCourseInfo(new Teacher()); //输出” 老师的课程信息”
printCourseInfo(new Student()); //输出” 学生的课程信息”
```

PHP 中 new 出的子类对象在测试函数中仍然是一个父类对象，因此动态绑定时会绑定对应的子类对象的成员函数，实现多态性.

在 C#中:

```
//C#
class Teacher:Person{
    private string title; //老师的职称
    public string createCourse(string courseInfo){ //创建课程方法
        ……
    }
    public override string viewCourse(){ //重写基类虚方法
        Console.WriteLine(“老师课程信息…”);
        ……
    }
}
//#########################################################
class Student:Person{
    private string ID;
    public string selectCourse(string courseID){ //选课方法
        ……
    }
    public override string viewCourse(){ //方法重写
        Console.WriteLine(“学生课程信息…”);
        ……
    }
}
```

```
//#########################################################
Person p1 = new Teacher();
Person p2 = new Student();
p1.viewCourse(); //输出”老师课程信息”
p2.viewCourse(); //输出”学生课程信息”
```

C#与 Java 在语法上有很多相似之处，通过 new 实例化对象之后，利用父类指针指向该子类对象，在子类对象中重写父类对象的虚方法，利用动态绑定即可寻找到对应的子类对象的成员方法，实现多态性.

```
//Java
class Teacher extends Person{
    private String title; //老师职称
    public String createCourse(String courseInfo){ //创建课程方法
        ……
    }
    public String viewCourse(){ //重写查看课程信息方法
        System.out.println(“老师课程信息…”);
        ……
    }
}
//#########################################################
class Student extends Person{
    private String ID; //学生学号
```

```
    public String selectCourse(String courseID){ //选课方法
        ……
    }
    public String viewCourse(){ //重写方法
        System.out.println( “学生课程信息…” );
        ……
    }
}
//#############################################################
Person p1 = new Teacher(); //实例化 Teacher 对象
Person p2 = new Student(); //实例化 Student 对象
p1.viewCourse(); //输出” 老师课程信息”
p2.viewCourse(); //输出” 学生课程信息”
```

在 Java 语言中，通过重写 Teacher 类和 Student 类在其基类 Person 类中的 viewCourse()方法，在实例化对象时将父类指针指向子类对象，在调用该函数时 Java 便可以通过动态绑定的方式找到对应子类的方法从而实现多态.

收获总结

通过本次调研，查阅了许多关于面向对象分析与设计方法相关资料，了解并熟悉了面向对象方法和工具的发展历程和发展现状，加深了对面向对象思想和方法的主要特性及其概念的理解. 通过对面向对象程序设计语言的调研，其中包括两种自己之前不了解的语言(C#和 PHP)以及一种自己比较熟悉的编程语言

Java，通过实际代码的分析和对比，分别分析了不同程序设计语言在实现面向对象的各个特性上的具体设计思想，通过实际例子加深了对面向对象中抽象，封装，继承和多态的理解，并且对 PHP 和 C#语言的基本语法有了一定的了解，不同的语言有相似甚至相同的部分，也有各自不同的特色之处，但从本质上来讲，都很好的体现了面向对象的设计和思想，本次调研达到了实验的目的，收获丰富.