作业1：程序设计语言的OO特性调研报告

中国工程物理研究院 王春柳

**调研问题：**不同语言的面向对象特征和关注重点并不尽相同，请调研至少两种语言的设计思想和面向对象特征，并与自己熟悉的语言做对比。

**调研内容：**Python、Ruby和Java的设计思想和面向对象特征

***注：完全面向对象语言应该具备的几种性质：***1.封装/信息隐藏2.继承3.多态/动态绑定4.所有预定义类型皆对象5.所有操作都由向对象发送消息实现6.所有用户定义的类型都是对象

**第一章 Python**

**一、Python的设计思想：**

可扩充性可说是Python作为一种编程语言的特色。新的内置模块（module）可以用C 或 C++写成。而我们也可为现成的模块加上Python的接口。Python可以使用户避免过分的语法的羁绊而将精力主要集中到所要实现的程序任务上。

Python也被称为是一门清晰的语言。因为它的作者在设计它的时候，总的指导思想是，为每件事情提供一种有且只提供一种方法。

Python语言是一种清晰的语言的另一个意思是，它的作者有意的设计限制性很强的语法，使得不好的编程习惯（例如if语句的下一行不向右缩进）都不能通过编译。这样有意的强制程序员养成良好的编程习惯。其中很重要的一项就是Python的缩进规则。

一个和其他大多数语言（如C）的区别就是，一个模块的界限，完全是由每行的首字符在这一行的位置来决定的（而C语言是用一对花括号{}来明确的定出模块的边界的，与字符的位置毫无关系）。这一点曾经引起过争议。因为自从C这类的语言诞生后，语言的语法含义与字符的排列方式分离开来，曾经被认为是一种程序语言的进步。不过不可否认的是，通过强制程序员们缩进（包括if，for和函数定义等所有需要使用模块的地方），Python确实使得程序更加清晰和美观。

另外Python在其他部分的设计上也坚持了清晰划一的风格，这使得Python称为一门易读性、易维护性好，并且被大量用户所欢迎的、用途广泛的语言。Python直接编写的程序段有时运行效率甚至高于用C编写的程序。

**二、Python的面向对象特征：**不满足备注中第一条和第五条。一些操作以方法的形式实现，而另一些则是用的全局函数；有些人也认为Python缺乏private或者隐藏属性，违背了封装/信息隐藏的原则。Python既支持面向过程的编程也支持面向对象的编程。在“面向过程”的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在“面向对象”的语言中，程序是由数据和功能组合而成的对象构建起来的。与其他主要的语言如 C++和Java相比，Python以一种非常强大又简单的方式实现面向对象编程。

**1）封装：**顾名思义，封装即把我们想要的内容封装到某个地方，因此在这里记录两个问题，一是如何封装、二是如何调用封装的内容。

①封装数据

class Company:

def \_\_init\_\_(self, dept, leader):

self.dept = dept

self.leader = leader

if \_\_name\_\_ == "\_\_main\_\_":

# 将公司名和Leader分别封装到对象obj1、obj2中self的dept和leader属性中

obj1 = Company("A","Kevin")

obj2 = Company("B","Jone")

②调用数据

调用数据又可以分为两种方式：通过对象直接调用、通过self间接调用

class Company:

def \_\_init\_\_(self, dept, leader):

self.dept = dept

self.leader = leader

def show(self):

print self.dept

print self.leader

if \_\_name\_\_ == "\_\_main\_\_":

obj1 = Company("A","Kevin")

obj2 = Company("B","Jone")

# 通过对象直接调用封装的数据

print obj1.dept

print obj1.leader

# 通过self来间接调用，self即为对象本身

obj1.show()

obj2.show()

总结封装特性：即通过类的构造方法将我们想要的内容封装到对象中，调用时通过对象直接调用、或通过self指向的类对象来间接调用。

**2）继承：**顾名思义继承即为拥有了某些不属于自己的东西，Python当中类的继承同样如此，通过继承可以为我们的类扩展更多的功能，同时在代码模块化开发、升级的过程中，继承也是一种规模化、高效化的生产方式。不同于其他语言，Python中的类还具有多继承的特性，即可以有多个父类，下面通过代码来展示类的单继承、多继承以及继承过程中类成员方法的寻找顺序。

①单继承

class Scale:

def check(self):

if self.count\_person > 500:

print "%s is big company." %self.name

else:

print "%s is small company." %self.name

class Company(Scale):

def \_\_init\_\_(self, name, count):

self.name = name

self.count\_person = count

if \_\_name\_\_ == "\_\_main\_\_":

my\_company = Company("ABC", 800)

# Company类中并没有check方法，代码会向上自动检测父类中是否存在check方法，结果在Scale类中找到了

my\_company.check()

②多继承

class Scale:

def check(self):

if self.count\_person > 500:

return "%s is big company." %self.name

else:

return "%s is small company." %self.name

class Detail:

def show(self, scale):

print "%s, This company has %s persons." %(scale, self.count\_person)

class Company(Scale, Detail):

def \_\_init\_\_(self, name, count):

self.name = name

self.count\_person = count

if \_\_name\_\_ == "\_\_main\_\_":

my\_company = Company("ABC", 800)

company\_scale = my\_company.check()

my\_company.show(company\_scale)

上述代码中Company分别继承了父类Scale、Detail，在实例化对象my\_company后，我们分别用到了父类中的check和show方法。

③类的继承顺序

与其说类的继承顺序，不如说是对象执行方法时，该方法在类中的查找顺序，之所以用到继承，那么一定是需要用到父类当中的某个方法，那么这个方法在类中就要遵循一定的查找顺序，简而言之可以概括为：先下后上、从左到右，下面通过代码来进行说明。

class Scale:

def check(self):

if self.count\_person > 500:

print "%s is big company." %self.name

else:

print "%s is small company." %self.name

class Detail:

def show(self, scale):

print "%s, This company has %s persons." %(scale, self.count\_person)

class Company(Scale, Detail):

def \_\_init\_\_(self, name, count):

self.name = name

self.count\_person = count

def show(self):

print "This is display from class Company, our company has %s persons." %self.count\_person

if \_\_name\_\_ == "\_\_main\_\_":

my\_company = Company("ABC", 800)

my\_company.check()

my\_company.show()

1.创建实例化对象 my\_company，并封装了公司名称、人数等基本内容到对象中。

2.调用类中的check方法，此时程序发现本类中没有check方法，于是开始向上寻找，最终在Detail类中找到。

3.调用类中的show方法，此时程序发现本类中存在show方法，直接调用。

**3）多态：**Python里面原生态的支持多态，比如传入参数的时候不需要指定类型，可以是任何数据类型；而类似的方式在Java或者C#里面需要通过继承来实现同一方法传入不同数据类型的参数。

class F1:

    pass

class S1(F1):

    def show(self):

        print ('S1.show')

class S2(F1):

    def show(self):

        print ('S2.show')

def Func(obj):

    print (obj.show())

s1\_obj = S1()

Func(s1\_obj)

s2\_obj = S2()

Func(s2\_obj)

-------------

S1.show

None

S2.show

None

**第二章 Ruby**

**一、Ruby设计思想：**减少编程过程中受到的挫折，让人在编程中付出最小的劳动，这是作者设计Ruby的主要目的。Ruby和Python给编程人员提供的能力几乎是相同的，所以作者在设计Ruby时重点不在于强调它能做什么，而在于用户在使用它时感觉如何。作者想设计一门不同于Python的能够给用户充分选择自由的语言，同时是简洁紧凑的能让编程人员感到幸福的语言。Ruby继承了Perl的理念，同一件事情可以用多于一种方法完成，Ruby的使用乐趣还在于它的写法类似于伪代码，这类似于Python。同时作者在文本处理上也下了很大功夫，Ruby可以调用Unix中的所有系统调用和Windows API中的大多数系统调用，这就把操作系统的能力和功能带到解释性语言环境中，因此用户可以进行日常的系统管理和文本处理编程，减少程序员的工作。

**二、Ruby面向对象特征：**满足备注中所有的条件。①完全面向对象，在Ruby语言中，任何东西都是对象，包括其他语言中的基本数据类型，比如整数。②变量没有类型，Ruby的变量可以保存任何类型的数据。③任何东西都有值，不管是数学或者逻辑表达式还是一个语句，都会有值。

**1）背景知识：**

Ruby类的定义：当定义一个类时，实际是定义了一个数据类型的蓝图。这实际上并没有定义任何的数据，而是定义了类的名称意味着什么，也就是说，定义了类的对象将由什么组成，以及在该对象上能执行什么操作。

类定义以关键字 class 开始，后跟类名称，最后以一个 end 进行分隔表示终止该类定义。例如，我们使用关键字 class 来定义 Box 类，如下所示：

class Box  
   code  
end

名称必须以大写字母开始，按照约定名称中包含多个单词，每个单词没有分隔符（驼峰式）一起执行。

定义Ruby的对象：我们声明一个类的对象使用new关键字。下面的语句声明了两个对象，Box 类：

box1 = Box.new

box2 = Box.new

initialize方法：initialize 方法是一个标准的 Ruby 类方法，是类的构造函数，与其他面向对象编程语言中的 constructor 工作原理类似。当想要在创建对象的同时初始化一些类变量，initialize 方法就派上用场了。该方法带有一系列参数，与其他 Ruby 方法一样，使用该方法时，必须在前面放置 def 关键字，如下所示：

class Box

   def initialize(w,h)

      @width, @height = w, h

   end

end

实例变量：实例变量是类属性，它们在使用类创建对象时就变成对象的属性。每个对象的属性是单独赋值的，和其他对象之间不共享值。在类的内部，是使用 @ 运算符访问这些属性，在类的外部，则是使用称为访问器方法的公共方法进行访问。下面我们以上面定义的类 Box 为实例，把 @width 和 @height 作为类 Box 的实例变量。

def initialize(w,h)

      # assign instance avriables

      @width, @height = w, h

   end

end

**2）总结：**Ruby和大多数别的语言之间有一个很大的区别。例如在Java中，是通过调用另外一个函数来得到某些成员数据的绝对值，并把结果传回数据本身。你可以写成

number= Math.abs(number) //Java 代码

而在Ruby里，确定绝对值的能力内建在数字中--处理细节在内部实现中。只要发送abs消息到一个数字对象，让它去得到绝对值即可。

number = number.abs//Ruby代码

Ruby语言已经了一个类Object,如果在定义新类的时候没有指明新类的父类，那么Ruby解释器认为，新类的父类是Object类，类中含有new方法，initialize方法，只要不重写这些方法，就可以自然使用类Objectd的方法。Ruby语言，只有重写（override），没有其他语言具有的严格意义上的重载（overload）。Ruby语言只支持单继承，每一个类都只能有一个直接父类。这样避免了多继承的复杂度。但同时，Ruby提供了mixin的机制可以用来实现多继承。可以使用super关键字调用对象父类的方法，当super省略参数时，将使用当前方法的参数来进行调用。

**3）Ruby的封装：**

在面向对象的设计理念中，为了提升封装性，会禁止直接的对象属性操作，只给出相应的get和set方法。而在Ruby中，从语言层级中就禁止了直接访问对象属性的途径，只存在方法调用。我们平常常见的a.name和a.name = 'Hello'的调用方式，看上去很像是直接操作属性，其实不是的。因为Ruby的方法调用可以省略括号，所以a.name等价于a.name(),而a.name=‘hello’的形式实际上是a.name=(‘hello’)的简写。

**4）Ruby的继承和多态代码示例：**

代码1：

class Person//创建Person父类

  def initialize( name,age=18 )

    @name = name

    @age = age

    @motherland = "China"

  end

  def talk

    puts "my name is "+@name+", age is "+@age.to\_s

    if @motherland == "China"

      puts "I am a Chinese."

    else

      puts "I am a foreigner."

    end

  end

  attr\_writer :motherland

end

class Student < Person//用“ < ”表示 Student类是Person类的子类。//Person 类的一切，Student 类都能继承。但是 Student 类重写了 talk 方//法，所以我们看到了不同的运行结果。子类继承父类的时候，除了重写方法//；也可以添加一些新的方法；或是增强父类的方法(用关键字 super指明)。

  def talk

    puts "I am a student. my name is "+@name+", age is "+@age.to\_s

  end

end

class Worker < Person// Worker类是Person类的子类。Worker类与 //Student类同样继承自Person类，亲缘关系是兄弟，当他们talk时，能准//确表明自己身份，因为他们都重写了各自的talk方法。不同的子类继承一个//父类，不仅子类和父类的行为有变异，而且子类彼此的行为也有差异，这就//是多态。

  def talk

    puts "I am a worker. my name is "+@name+", age is "+@age.to\_s

  end

end

p3=Student.new("kaichuan",25);

p3.talk

p4=Student.new("Ben");

p4.talk

p5=Worker.new("kaichuan",30);

p5.talk

p6=Worker.new("Ben");

p6.talk

执行结果：

my name is Kaichuan,age is 20

I am a China.

my name is Ben,age is 18

I am a foreigner.

I am a student. my name is Chengang,age is 25

I am a student. my name is Ben,age is 18

I am a worker. my name is Hiong,age is 30

I am a worker. my name is Ben,age is 18

代码2：

/\*使用super关键字调用对象父类的方法，当super省略参数时，将使用当前方法的参数来进行调用\*/

class Base

  def meth(info)

    puts "This is Base #{info}"

  end

end

class Derived < Base

  def meth(info)

    puts "This is derived #{info}"

    super

  end

end

obj1 = Derived.new

obj1.meth("test")

执行结果：

This is derived test

This is Base test

/\*如果传入的参数被修改再调用super的话，那么将会使用使用修改后的值。\*/

Eg：

class Base

  def meth(info)

    puts "This is Base #{info}"

  end

end

class Derived < Base

  def meth(info)

    puts "This is derived #{info}"

    info = "over"

    super

  end

end

obj1 = Derived.new

obj1.meth("test")

执行结果为：

This is derived test

This is Base over

**第三章 Java**

**一、Java设计思想：**有一条语言设计的准则，同时也是Java设计过程中的一个重要原则——保守。具体说来，就是每当你添加一个新特性的时候，可能你也造成了新的问题。如果你的语言有200种特性，而这个时候你想再加入一个，我需要检验它和所有其他特性的交互情况，所以Java采用不同于Scala的做法：不去频繁的加入新的特性。

当James解释他设计Java的初衷时说：当我在设计的时候，我知道人们想要自动内存管理，人们想要強型式，但是这些功能会吓跑蓝领工人。比如说Smalltalk，这是一门很优秀的语言，但是它太先进了，它和现实中开发者们在构建应用时的思维脱离开来。所以Java继承了其中的一些理念，并将其简化，把这些理念放入一种语言和格式中。这些事解释了这门语言设计的基本动机。

我们可以认为Java是一门冗长的语言，但是作者认为它额外的内容可以方便阅读，特别是当读者是初级或者中级的程序员时，所以作者并不认为他是一门冗长的语言。

**二、Java语言特征：**不满足备注中第四条和第五条。其基本数据类型并不是对象，基本的算数运算是内置的运算符，而不是对象的消息。

**1）封装**

封装原则：

将不需要对外提供的内容都隐藏起来。

把属性都隐藏，提供公共方法对其访问。

Java中可以通过对类的成员设置一定的访问权限，实现类中成员的信息隐藏。

private：类中限定为private的成员，只能被这个类本身访问。如果一个类的构造方法声明为private,则其它类不能生成该类的一个实例。

default：类中不加任何访问权限限定的成员属于缺省的（default）访问状态，可以被这个类本身和同一个包中的类所访问。

protected：类中限定为protected的成员，可以被这个类本身、它的子类（包括同一个包中以及不同包中的子类）和同一个包中的所有其他的类访问。

public：类中限定为public的成员，可以被所有的类访问。

如下面的这个student类，就是使用了封装，将类中的属性name 、age和score私有化，使外部不能直接访问他们，只能通过public类型的对他们方法进行操作。

class Student{

private String name;//声明属性，并设为私有类型，起到封装的作用

private int age;

private int score;

public Student(String name,int age,int score){

this.name=name;

this.age=age;

this.score=score;

}

public String getName(){//设置get方法，因为变量设为了私有，对象无法访问类中的属性，需要用get方法获取变量

return name;

}

public void setName(String name){//用set方法对变量赋值

this.name=name

}

public int getAge(){

return age;

}

public void setAge(int age){

this.age=age;

}

public int getScore(){

return score;

}

public void setScore(int score){

this.score=score;

}

}

**2)继承**

继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。

新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。

子类可以直接访问父类中的非私有的属性和行为。通过 extends 关键字让类与类之间产生继承关系。如下：class SubDemo extends Demo{}

继承提高了代码的复用性。

继承的出现让类与类之间产生了关系，提供了多态的前提

继承的特点：

Java只支持单继承，不支持多继承。

一个类只能有一个父类，不可以有多个父类。

class SubDemo extends Demo{} //这是对的

class SubDemo extends Demo1,Demo2...//这是错的

Java支持多层继承(继承体系)

class A{}

class B extends A{}

class C extends B{}

定义继承需要注意：

不要仅为了获取其他类中某个功能而去继承

类与类之间要有所属关系，xx1是xx2的一种

class Ren{//父类

int age=20;

String name=”wk”;

Super.show

public void show(){

System.out.println(“姓名：”+name+“年龄：”+age)；

}

}

class Student extends Ren{//子类Student继承了父类Ren

int score=90；//增加了新的属性

int age=super.age+1;//使用super标记父类中的元素，修改了Ren中的age

public void show(){//重写了父类中的方法（覆盖时，子类方法权限一定要大于等于父类方法权限，静态只能覆盖静态。）

System.out println（“姓名：”+name+“年龄：”+age+“分数：”+score）；//Student继承了Ren，所以Ren中的属性name和age在Student可以直接使用

}

}

public class Person{

public static void main(String[] args){

Ren ren=new Ren();//声明一个Ren类

Students student=new Student();声明一个Student类

ren.show()

student.show();

}

}

super和this的用法相同：this代表本类引用；super代表父类引用。当字符类出现同名成员时，可以用super进行区分，子类要调用父类构造函数时，可以使用super语句。在子类覆盖方法中，继续使用被覆盖的方法可以通过super。函数名获取。

注意：

1.子类中所有的构造函数默认都会访问父类中空参数的构造函数，因为每一个构造函数的第一行都有一条默认的语句super（）；子类会具备父类中的数据，所以要先声明父类是如何对这些数据初始化的。当父类中没有空参数的构造函数时，子类的构造函数必须通过this或者super语句指定要访问的构造函数。

2.覆盖时子类方法权限一定要大于等于父类方法权限，静态只能覆盖静态，父类中的私有方法不可以被覆盖。

3.被final修饰的类是一个最终类，不可以被继承。被final修饰的方法是一个最终方法，不可以被覆盖。被final修饰的变量是一个常量，只能赋值一次，内部类只能访问被final修饰的局部变量。

继承与重载：一是子类与父类的关系，二是重载方法的调用问题。

**3）多态**

方法的重写、重载与动态连接构成多态性。java只允许单继承，这样做虽然保证了继承关系的简单明了，但是功能上有很大的限制，所以java引入了多态性的概念。此外，抽象类和接口也是解决单继承规定限制的重要手段，同时，多态性也是面向对象编程的精髓所在。

多态性：发送消息给某个对象，让该对象自行决定响应何种行为。通过将子类对象引用赋值给超类对象引用变量来实现动态方法调用。要理解多态性，首先要知道什么是“向上转型”。

子类Cat继承了Animal类，那么后者就是前者的父类。

　　Cat c = new Cat();//实例化一个Cat的对象，

Animal a = new Cat();//定义了一个Animal类型的引用，指向新建的Cat类型的对象

由于Cat是继承自它的父类Animal，所以Animal类型的引用是可以指向Cat类型的对象的。那么这样做的意义是什么：因为子类是对父类的一个改进和扩充，所以一般子类在功能上较父类更强大，属性较父类更独特，

定义一个父类类型的引用指向一个子类的对象既可以使用子类强大的功能，又可以抽取父类的共性。

所以，父类类型的引用可以调用父类中定义的所有属性和方法，但是对于子类中定义而父类中没有的方法，它是无可奈何的；

同时，父类中的一个方法只有在在父类中定义而在子类中没有重写的情况下，才可以被父类类型的引用调用；

对于父类中定义的方法，如果子类中重写了该方法，那么父类类型的引用将会调用子类中的这个方法，这就是动态连接。

class Father{

public void func1(){

func2();

}

public void func2(){//这是父类中的func2()方法，因为下面的子类中//重写了该方法，所以在父类类型的引用中调用时，这个方法将不再有效，取而//代之的是将调用子类中的重写的func()方法

System.out.println(“AAA”);

}

}

class Child extends Father{

public void func1(int i){

System.out.println(“BBB”);

}

//func1(int i)是对func1()方法的一个重载，由于在父类中没有定义这个方法，//所以它不能被父类类型的引用调用，所以在下面的main方法中//child.func1(68)是不对的

public void func2(int i){

System.out.println(“CCC”);

}

//func2()重写了父类Father中的func2()方法

//如果父类类型的引用中调用了func2()方阿飞，那么必然是子类中重写的这个//方

}

public class PolymorphismTest{

public static void main(String[] args){

Father child=new Child();

child.func1();//打印结果？CCC

}

}

这是个很典型的多态的例子，子类Child继承了父类Father，并重载了父类的func1的方法，重写了父类func2()方法。重载后的func1()和func1(int i)不再是同一个方法，由于父类中没有func1(int i)，那么父类类型的引用child就不能调用func（int i）方法，而子类重写了func2()，那么父类类型的引用child

在调用该方法时将会调用子类中重写的func2().

覆盖，是指子类重新定义父类的虚函数的做法。它是覆盖了一个方法并且对其重写，以求达到不同的作用。在覆盖要注意以下的几点：

1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；

4、被覆盖的方法不能为private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

重载，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，在使用重载要注意以下的几点：

1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是fun(int,float)，但是不能为fun(int,int)）；

2、不能通过访问权限、返回类型、抛出的异常进行重载；

3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

多态也有弊端：当父类引用指向子类对象时，虽然提高了扩展性，但是只能访问父类中具备的方法，不可以访问子类中特有的方法。(前期不能使用后期产生的功能，即访问的局限性)