程序设计语言的OO特性调研报告

面向对象其实是现实世界模型的自然延伸。现实世界中任何实体都可以看作是对象。对象之间通过消息相互作用。另外，现实世界中任何实体都可归属于某类事物，任何对象都是某一类事物的实例。如果说传统的面向过程式编程语言是以过程为中心以算法为驱动的话，面向对象的编程语言则是以对象为中心以消息为驱动。用公式表示，过程式编程语言为：程序=算法+数据；面向对象编程语言为：程序=对象+消息。

对于面向对象语言应满足的标准，没有统一的答案。但大多数研究认为，面向对象语言所应具有以下六个性质：

1. Encapsulation/Information Hiding （封装/信息隐藏）
2. Inheritance （继承）
3. Polymorphism/Dynamic Binding （多态/动态绑定）
4. All pre-defined types are Objects （所有预定义类型皆对象）
5. All operations performed by sending messages to Objects （所有操作都由向对象发送消息实现）
6. All user-defined types are Objects （所有用户定义的类型都是对象）

如果一门编程语言满足了所有这些性质，我们就可以认为这门语言是“纯粹的”面向对象语言。一门“混合型”语言可能支持这些性质中的某几条，但不是全部。特别的，许多语言支持前三条性质，但不支持后三条。下表列出了几种常见的编程语言对这6条性质的满足情况对比：

Eiffel	Smalltalk	Ruby	Java	C#	C++	Python	Perl	Visual Basic	
Encapsulation / Information Hiding	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes?	Yes?
Inheritance	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes?	No
Polymorphism / Dynamic Binding	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes?	Yes (through delegation)
All pre-defined types are Objects	Yes	Yes	Yes	No	No	No	Yes	No	No
All operations are messages to Objects	Yes	Yes	Yes	No	No	No	No	No	No
All user-defined types are Objects	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No

从上表可以看出，Smalltalk，Ruby，Java是“纯粹的”面向对象语言，C#，C++，Python，Perl， Visual Basic并没有满足这六条规则，但是都满足了前三条，因而并不是“纯粹的”面向对象语言。本文主要调查了Java，C++，Python 这三种语言在面向对象方面的特性，以期更好地理解程序设计语言对于面向对象的支持。

一、Java 对于面向对象的支持

Java 是完全面向对象的，Java中的所有东西都必须置入一个类。不存在全局函数、全局数据，也没有像结构、枚举或者联合这种东西，一切只有“类”！现实世界中的对象均有属性和行为，映射到计算机程序上，属性则表示对象的数据，行为表示对象的方法（其作用是处理数据或同外界交

互）。所谓封装，就是用一个自主式框架把对象的数据和方法联在一起形成一个整体。可以说，对象是支持封装的手段，是封装的基本单位。Java语言的封装性较强，因为Java无全局变量，无主函数，在Java中绝大部分成员是对象，只有简单的数字类型、字符类型和布尔类型除外。而对于这些类型，Java也提供了相应的对象类型以便与其他对象交互操作。

1.封装性：

面向对象里面类的提出就是为了实现封装，我们将一类同样的对象的一些相同或相似的属性及方法抽象出来，封装在一起就实现了封装。封装的目的就是为了实现一些数据及方法的内部隐藏（通过修饰符来实现），并提供一些通用的方法供外部使用，而具体的内部实现细节无需对外公开，加强了安全性。

定义了Car类：

```
1   package com.mct.main;
2   public class Car {
3       protected String name;  // 一般不将类的属性设置为public，这里设置为protected，因为这两个属性肯定是需要被子类继承的。
4       protected String color;
5       public String getName() {  // 通过get,set方法让外部访问并设置属性。
6           return name;
7       }
8       public void setName(String name) {
9           this.name = name;
10      }
11      public String getColor() {
12          return color;
13      }
14      public void setColor(String color) {
15          this.color = color;
16      }
17      public Car(){}  // 如果提供了有参数的构造函数，则最好再提供一个无参的构造函数，即使是空函数也好。
18      public Car(String name, String color){
19          this.name = name;
20          this.color = color;
21      }
22      public void speedUp(){
23          System.out.println("运行这里加速 ");
24      }
25      public void speedDown(){
26          System.out.println("运行这里减速");
27      }
28  }
```

2.继承

继承是使用已经存在的类为基础实现新的类，继承使用extends关键字。在Java里面只允许实现单继承，子类将完全继承父类中允许继承的属性和方法(具体依据修饰符的访问权限而定)，另外子类也可添加自己的属性和方法，但不能选择性的继承父类。如果父类的某些功能无法满足子类要求，可以使用下面要讲到的覆写或重载（有些著作中认为父类与子类之间不存在重载关系）的方法来实现自己特有的方法。

```
public class Animal {
    private String name;
    private int id;
    public Animal(String myName, int myid) {
        name = myName;
        id = myid;
    }
    public void eat(){
        System.out.println(name+"正在吃");
    }
    public void sleep(){
        System.out.println(name+"正在睡");
    }
    public void introduction() {
        System.out.println("大家好! 我是"         + id + "号" + name + ".");
```

```
    }
}


public class Penguin extends Animal {
    public Penguin(String myName, int myid) {
        super(myName, myid);
    }
}

public class Mouse extends Animal {
    public Mouse(String myName, int myid) {
        super(myName, myid);
    }
}
```

3. 多态

多态是同一个行为具有多个不同表现形式或形态的能力。多态就是同一个接口，使用不同的实例而执行不同操作。

多态的优点

1. 消除类型之间的耦合关系
2. 可替换性
3. 可扩充性
4. 接口性
5. 灵活性
6. 简化性

多态存在的三个必要条件

1. 继承
2. 重写
3. 父类引用指向子类对象

```
public class Test {
    public static void main(String[] args) {
      show(new Cat());  // 以 Cat 对象调用 show 方法
      show(new Dog());  // 以 Dog 对象调用 show 方法

      Animal a = new Cat();  // 向上转型
      a.eat();               // 调用的是 Cat 的 eat
      Cat c = (Cat)a;        // 向下转型
      c.work();        // 调用的是 Cat 的 catchMouse
  }

    public static void show(Animal a)  {
      a.eat();
        // 类型判断
        if (a instanceof Cat)  {  // 猫做的事情
            Cat c = (Cat)a;
            c.work();
        } else if (a instanceof Dog) { // 狗做的事情
            Dog c = (Dog)a;
            c.work();
        }
    }
}

abstract class Animal {
    abstract void eat();
}

class Cat extends Animal {
    public void eat() {
        System.out.println("吃鱼");
    }
    public void work() {
        System.out.println("抓老鼠");
    }
}
```

```java
class Dog extends Animal {
    public void eat() {
        System.out.println("吃骨头");
    }
    public void work() {
        System.out.println("看家");
    }
}
```

二、C++ 对于面向对象的支持

C++ 在 C 语言的基础上增加了面向对象编程，C++ 支持面向对象程序设计。类是 C++ 的核心特性，通常被称为用户定义的类型。类用于指定对象的形式，它包含了数据表示法和用于处理数据的方法。类中的数据和方法称为类的成员。函数在一个类被称为类的成员。

1. C++ 定义类

```
class Box
{
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

2. C++ 定义类的对象

```
Box Box1;          // 声明 Box1, 类型为 Box
Box Box2;          // 声明 Box2, 类型为 Box
```

3.访问数据成员

类的对象的公共数据成员可以使用直接成员访问运算符 (.) 来访问。

```
#include <iostream>

using namespace std;

class Box
{
   public:
      double length;   // 长度
      double breadth;  // 宽度
      double height;   // 高度
};

int main( )
{
   Box Box1;        // 声明 Box1, 类型为 Box
   Box Box2;        // 声明 Box2, 类型为 Box
   double volume = 0.0;     // 用于存储体积

   // box 1 详述
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 详述
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;

   // box 1 的体积
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Box1 的体积: " << volume <<endl;
```

```
    // box 2 的体积
    volume = Box2.height * Box2.length * Box2.breadth;
    cout << "Box2 的体积: " << volume <<endl;
    return 0;
}
```

三、Python 对于面向对象的支持

Java和C#来说只支持面向对象编程，而python比较灵活即支持面向对象编程也支持函数式编程。

1. 类与对象的创建

```
# 创建类
class Foo:

    def Bar(self):
        print 'Bar'

    def Hello(self, name):
        print 'i am %s' %name

# 根据类Foo创建对象obj
obj = Foo()
obj.Bar()            #执行Bar方法
obj.Hello('wupeiqi') #执行Hello方法
```

2. 从某处调用被封装的内容

调用被封装的内容时，有两种情况：通过对象直接调用；通过self间接调用。

1、通过对象直接调用被封装的内容

```
class Foo:

    def __init__(self, name, age):
        self.name = name
        self.age = age

obj1 = Foo('wupeiqi', 18)
print obj1.name    # 直接调用obj1对象的name属性
print obj1.age     # 直接调用obj1对象的age属性

obj2 = Foo('alex', 73)
print obj2.name    # 直接调用obj2对象的name属性
print obj2.age     # 直接调用obj2对象的age属性
```

3. 通过self间接调用被封装的内容

执行类中的方法时，需要通过self间接调用被封装的内容。

```
class Foo:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def detail(self):
        print self.name
        print self.age

obj1 = Foo('wupeiqi', 18)
obj1.detail()  # Python默认会将obj1传给self参数，即：obj1.detail(obj1)，所以，此时方法内部的 self = obj1，即：
self.name 是 wupeiqi ；self.age 是 18

obj2 = Foo('alex', 73)
obj2.detail()  # Python默认会将obj2传给self参数，即：obj1.detail(obj2)，所以，此时方法内部的 self = obj2，即：
self.name 是 alex ；  self.age 是 78
```

四、Java、C++ 、Python 对于面向对象的支持比较

面向对象技术中的对象就是现实世界中某个具体的实体在计算机逻辑中的映射和体现，而类则是同种对象的集合与抽象。所有的Java程序都是由类或者说是类的定义组成的。在我们的学习过程中应该自始至终地牢记这一点，因为这意味着Java是一种完全的面向对象语言。Java中的所有东西都必须置入一个类。不存在全局函数、全局数据，也没有像结构、枚举或者联合这种东西，一切只有“类”！

然而C++则不同，比如C++的main方法是置于所有的类之外的，除此之外还可以在类外定义其它的函数。在C++中，全局变量、结构、枚举、联合等一些列源于C的概念仍然存在。对于在这个问题上的区别，不同的人会有不同的看法，C++的优点是灵活机动，而且比较利于C程序员接受，因为在C中成立的事情在C++中基本上不会出现差错，他们只需要了解C++多了哪些东西便可以了，然而也正因为如此，C++杂糅了面向对象和面向过程的双重概念，由此产生出的很多机制在强化某部分功能的同时破坏了程序的整体结构。

与此相比，Java语言去除了C++中为了兼容C语言而保留的非面向对象的内容，对于对C比较习惯的人来说不是十分友好，在很多问题的处理上也显得有些弃简就繁，但是它以此换来了严谨和可靠的结构，以及在维护和管理上的方便。因此对两种语言的总体比较可以得出的结论是：C++更加灵活而Java更加严谨。

（1）在Java中，类定义采取几乎和C++一样的形式，只不过没有标志结束的分号。

（2）Java中的所有方法都是在类的主体定义的而C++并非如此。在Java中我们必须将函数的定义置于类的内部，这种禁止在类外对方法定义的规定和Java的完全面向对象特性是吻合的。

（3）Java中没有作用域范围运算符“::”。Java利用“.”做所有的事情，但可以不用考虑它，因为只能在一个类里定义元素。即使那些方法定义，也必须在一个类的内部，所以根本没有必要指定作用域的范围。而对于static方法的调用，也是通过使用ClassName.methodName()就可以了。

在C++中，同时支持函数重载和运算符重载，而Java具有方法重载的能力，但不允许运算符重载。C++在语法上直接支持多继承，其格式为：class 派生类名：访问控制关键字 1 基类名1，访问控制关键字 2 基类名2，...Java出于简化程序结构的考虑，取消了语法上对多继承的直接支持，而是用接口来实现多重继承功能的结构。

Python经常被标榜为一种面向对象语言，但它对面向对象的支持似乎是被“贴上去的”。一些操作以方法的形式实现，而另一些则用的是全局函数。另外，方法参数中需要显式地添加“self”参数也很令人尴尬。有一些人抱怨Python缺乏“private”或者“隐藏”属性，这违背了封装/信息隐藏的原则，而另一些人则认为“通过约定实现的私有”在不增加额外限制的同时，也提供了语言强制的封装带来的优势。